МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РЕСПУБЛИКИ КАЗАХСТАН

Казахский национальный исследовательский
технический университет имени К.И. Сатпаева

Б. А. Ахметов, А. Г. Корченко,
В. П. Сиденко, Ю. А. Дрейс, Н. А. Сейлова

ПРИКЛАДНАЯ КРИПТОЛОГИЯ:

методы шифрования

Учебное пособие

Рекомендовано Учебно-методическим объединением
Республиканского учебно-методического совета
Министерства образования и науки Республики Казахстан

Алматы – 2015

УДК 004.056.55 (075.8)
ББК 32.97 Я73
П 75

Р е ц е н з е н т ы:
Доктор технических наук, профессор Тукеев А. Т.
Доктор технических наук, профессор Утепбергенов Е. Т.
Доктор технических наук, профессор Джурунтаев Д. З.

П 75 Прикладная криптология: методы шифрования : учебное пособие / Б. С. Ахметов, А. Г. Корченко, В. П. Сиденко и др. – Алматы : КазНИТУ имени К. И. Сатпаева, 2015. – 496 с.: ил.

ISBN 978-601-228-879-7

Учебное пособие содержит общие сведения: основы криптологии, криптографии и криптографического анализа; традиционно исторические шифры подстановки и перестановки; блочные и составные шифры и атаки на них; потоковые шифры и генераторы псевдослучайных чисел; стандарты криптографического шифрования и преобразования; блочные симметричные криптографические алгоритмы; принципы построения симметричных и асимметричных криптографических систем шифрования, которые используются для обеспечения конфиденциальности данных в информационно-телекоммуникационных системах, а также примеры с решениями, контрольные задания с ответами.

Пособие подготовлено авторским коллективом Казахского национального исследовательского технического университета имени К.И. Сатпаева и Национального авиационного университета Украины (совместно с учеными Житомирского военного института имени С. П. Королева) в соответствии с Меморандумом о сотрудничестве, и предназначено для бакалавров, магистрантов, аспирантов и докторантов высших учебных заведений, обучающихся в области знаний «Информационная безопасность».

УДК 004.056.55 (075.8)
ББК 32.97 Я73

ISBN 978-601-228-879-7

Печатается по плану издания Министерства образования и науки Республики Казахстан на 2015 год

СОДЕРЖАНИЕ

СПИСОК СОКРАЩЕНИЙ

AES - Advanced Encryption Standard (Усовершенствованный стандарт шифрования)
ASCII - American Standard Code for Information Interchange (Американский стандарт кодов для обмена информацией)
ATM - Asynchronous Transfer Mode (Асинхронный метод переноса)
BBS - Algorithm Blum – Blum – Shub (Алгоритм Блюм-Блюма-Шуба)
CBC - Cipher Block Chaining (режим “сцепления блоков зашифрованных данных”)
CFB - Cipher Feedback (режим “обратной связи по зашифрованными данными”)
CTR - Counter Mode (режим “счетчика”)
DES - Data Encryption Standard (Стандарт шифрования данных)
ECB - Electronic Code Book (режим “электронной кодовой книги”)
FIPS - Federal Information Processing Standards (Федеральный стандарт обработки информации)
GSM - Global System for Mobile Communications (Глобальный цифровой стандарт мобильной сотовой связи)
IBM - International Business Machines (Корпорация международных бизнес-машин)
IDEA - International Data Encryption Algorithm (Международный алгоритм шифрования данных)
IPES - Improved Proposed Encryption Standard (Усовершенствованный предложенный стандарт шифрования)
IST - Information Societies Technology (Информационные общественные технологии)
IV - Initialization Vector (Вектор инициализации)
KSA - Key-Scheduling Algorithm (алгоритм ключевого расписания)
LFSR - linear feedback shift register (регистр сдвига с линейной обратной связью)
MA - Multiplication/Addition (умножение/сложение)
MAC - Message Authentication Code (код аутентификации сообщения)

MDC	-	Modification/Manipulation Detection Code (код обнаружения изменений/манипуляций)
MIT	-	Massachusetts Institute of Technology (Массачусетский технологический институт)
NFS	-	Number Field Sieve (Общий метод решета числового поля – метод факторизации целых чисел)
NIST	-	National Institute of Standards and Technology (Национальный институт стандартов и технологии)
OFB	-	Output Feedback (режим “обратной связи по выходу”)
PES	-	Proposed Encryption Standard (Предложенный стандарт шифрования)
RSA	-	Rivest, Shamir i Adleman (криптографический алгоритм с открытым ключом)
SPA	-	Software Publishers Association (Ассоциация издателей программного обеспечения)
TCP	-	Transmission Control Protocol (Протокол управления передачей)
АПШ	-	асинхронные потоковые шифры
АС		автоматизированная система
ГПСЧ	-	генератор псевдослучайных чисел
ГППСЧ	-	генератор последовательности псевдослучайных чисел
ДК	-	дифференциальный криптографический анализ
ЭВМ	-	электронная вычислительная машина
ЭЦП	-	электронная цифровая подпись
ИТС	-	информационно-телекоммуникационная система
КЗУ	-	ключевое запоминающее устройство
ЛК	-	линейный криптографический анализ
ППСЧ	-	последовательность псевдослучайных чисел
ПСП	-	псевдослучайная последовательность
ПВО	-	противовоздушная оборона
СПШ	-	синхронные потоковые шифры
СССР	-	Союз Советских Социалистических Республик
ФАПСИ	-	Федеральное агентство правительственной связи и информации при Президенте РФ

ВВЕДЕНИЕ

На сегодня первичным фактором, влияющим на политическую и экономическую составляющие национальной безопасности, является *степень защищенности информации и информационной среды*. Вот почему важное значение приобретают вопросы обеспечения защиты информации в автоматизированных (информационных) и телекоммуникационных системах во время обработки в различных сферах деятельности.

Под *обработкой информации* в системе понимают выполнение одной или нескольких операций, в частности: сбора, ввода, записи, преобразования, считывания, хранения, уничтожения, регистрации, приема, получения, передачи, которые осуществляются в системе с помощью технических и программных средств. Такие программные средства (или программное обеспечение), как и сама информация, которая обрабатывается, являются *объектами защиты*. В целом в защите нуждается *информация с ограниченным доступом* (критическая информация), а именно: *конфиденциальная*, *служебная* или *секретная информация* (например, государственная тайна, банковская тайна или иная тайна). Во время обработки информации с ограниченным доступом должна обеспечиваться ее защита от: несанкционированного и неконтролируемого ознакомления, модификации, уничтожения, копирования, распространения, т.е. обеспечения основных ее свойств защищенности - конфиденциальности, целостности, доступности и наблюдательности. В частности, передача служебной и секретной информации из одной системы в другую осуществляется в зашифрованном виде или по защищенным каналам связи с помощью *технической* и *криптографической защиты информации*. Последний способ защиты в свою очередь реализуется программными, программно-аппаратными и аппаратными средствами путем преобразования информации с использованием специальных (ключевых) данных с целью скрытия/восстановления

содержания информации, подтверждение ее подлинности, целостности и авторства и т.д. Совокупностью средств криптографической защиты информации, необходимой ключевой, нормативной, эксплуатационной, а также другой документации (в том числе определяет меры безопасности), использование которых обеспечивает должный уровень защищенности информации, обрабатываемой, хранимой и/или передаваемой называют *криптографической системой* (криптосистемой). А совокупность органов, подразделений, групп, деятельность которых направлена на обеспечение криптографической защиты информации, предприятий, учреждений и организаций, разрабатывающих, производящих, эксплуатирующих и/или распространяющих криптосистемы и средства криптографической защиты информации – *системой криптографической защиты информации*.

Формирование общих принципов построения систем криптографической защиты информации (систем шифрования) на основе использования современных алгоритмов симметричного и асимметричного шифрования для обеспечения конфиденциальности данных в *информационно-телекоммуникационной системе* (*ИТС*) является *целью данного учебного пособия*, который состоит из таких разделов:

первый раздел содержит общие сведения об основах криптологии, криптографии и криптографического анализа. В нем раскрыты основы защиты информации в *ИТС*, роль криптографических протоколов в общей задаче обеспечения информационной безопасности, основы теории засекреченной связи *К. Шеннона*, приведена классификация методов шифрования сообщений;

второй раздел посвящен традиционно историческим шифрам подстановки (моноалфавитные, многоалфавитные) и перестановки;

третий раздел определяет принципы построения современных симметричных криптографических систем, использующих блочные и составные шифры с криптографическим анализом атак на блочные шифры;

в *четвертом разделе* раскрыты потоковые шифры (*A5, RC4*) и генераторы псевдослучайных чисел, основанных на алгоритме *BBS* и регистров сдвига с обратной связью;

пятый раздел раскрывает суть стандарта блочного симметричного шифрования данных *Data Encription Standard* (*DES*): история

создания, принципы построения, структура, примеры шифрования и криптографический анализ возможных атак;

шестой раздел содержит режимы выполнения криптографических алгоритмов блочного симметричного шифрования данных;

в *седьмом разделе* подробно рассматривается стандарт криптографического преобразования *ГОСТ 28147-89* в режиме простой замены, режиме гаммирования, режиме гаммирования с обратной связью и режиме формирования имитовставки;

восьмом раздел посвящен блочным симметричным криптографическим алгоритмам *Rijndael* и *Advanced Encryption Standard* (*AES*): структура стандарта, раунд преобразования и т.п.;

девятый раздел раскрывает суть международных алгоритмов шифрования данных *International Data Encryption Algorithm* (*IDEA*) и *Camelia*: историю создания, принципы построения, структуру, примеры шифрования и криптографический анализ возможных атак;

в *десятом разделе* рассмотрены асимметричные криптографические системы: *Диффи-Хеллмана, Шамира, Эль-Гамаля, RSA, Рабина*; методы взлома криптосистем, основанных на дискретном логарифмировании.

К каждому разделу прилагаются *контрольные вопросы и задания* для самоконтроля усвоенных знаний, а также ответы на контрольные задания для самопроверки правильности их выполнения.

Структура и содержание учебного пособия определено, исходя из содержания рабочих учебных программ специальности «Ситемы информационной безопасности» ее содержательных модулей, теоретического и практического опыта авторов.

Учебное пособие адресовано бакалаврам и магистрантам для учебного процесса в отрасли знаний «Информационная безопасность», в первую очередь для направления подготовки специалистов по специальности «Системы информационной безопасности», специалистам, которые связаны с эксплуатацией и использованием криптографических систем, комплексов и средств защиты информации в *ИТС*, а также может быть использован докторантами и учеными при проведении научных исследований.

Раздел 1

ОСНОВЫ КРИПТОЛОГИИ, КРИПТОГРАФИИ И КРИПТОГРАФИЧЕСКОГО АНАЛИЗА

1.1. ОБЩИЕ СВЕДЕНИЯ О ЗАЩИТЕ ИНФОРМАЦИИ В ИНФОРМАЦИОННО-ТЕЛЕКОММУНИКАЦИОННЫХ СИСТЕМАХ

Стремительное развитие средств вычислительной техники и открытых сетей передачи данных обусловило их широкое распространение в повседневной жизни и предпринимательской деятельности. Мощные вычислительные возможности и оперативность передачи информации не только повлияли на принципы ведения бизнеса, сложившиеся в большинстве традиционных отраслей, но и открыли новые направления развития предпринимательской деятельности [21].

Однако последние достижения человеческой мысли в области компьютерных технологий связаны с появлением не только персональных компьютеров, сетей передачи данных и электронных денег, но и таких понятий, как хакер, информационное оружие, компьютерные вирусы и др.

На практике угрозы *ИТС* могут быть реализованы непосредственным воздействием на информацию, представляющую интерес для конечных пользователей подобных систем, так и на информационные ресурсы и телекоммуникационные службы, которые обеспечиваемые в рамках этой *ИТС*. Например, существует распространенный вид атаки через *Internet* - шторм ложных запросов на *TCP* (*Transmission Control Protocol - протокол управления передачей*) – соединение, приводящий к тому, что система временно прекращает обслуживание удаленных пользователей.

Под *информационной безопасностью* будем понимать состояние защищенности обрабатываемых, хранимых и передаваемых в *ИТС* данных от незаконного ознакомления, преобразования и уничтожения (как крайний случай модификации), а также состояние защищенности информационных ресурсов от воздействий, направленных на нарушение их работоспособности.

Основными задачами защиты информации, которая предназначена для пользователя, является обеспечение [21]:

- конфиденциальности информации;
- целостности информации;
- достоверности информации;
- оперативности доступа к информации;
- юридической значимости информации, представленной в виде электронного документа;
- неотслеживаемости действий клиента.

Конфиденциальность информации - это ее свойство быть доступной только ограниченному кругу пользователей *ИТС*, в которой циркулирует эта информация.

Под *целостностью информации* понимают свойство ее или программного обеспечения сохранять свою структуру и/или содержание в процессе передачи и/или хранения. Рассматривая вопросы передачи информации в виде сообщения через сеть, можно прийти к заключению, что каждое сообщение по своему смысловому содержанию образует некоторый класс. Другими словами, смысл конечного сообщения останется таким же, как и начального, даже если форма представления информации в электронном виде существенно изменится. Таким образом, каждое сообщение на каком-либо языке будет иметь свой класс эквивалентности, и для данного случая свойство *сохранения целостности* можно сформулировать следующим образом: переданное сообщение *M* считается сохранившим целостность, если полученное в результате передачи сообщение *M'* принадлежит классу эквивалентности сообщения *M*.

Достоверность информации – это свойство, которое выражается в строгой принадлежности объекту, который является ее источником, или тому объекту, от которого эта информация принята.

Оперативность – это способность информации или некоторого информационного ресурса быть доступным для конечного пользователя в соответствии с его временными потребностями.

Юридическая значимость означает, что документ имеет юридическую силу. С этой целью субъекты, которые нуждаются в подтверждении юридической значимости передаваемого сообщения, договариваются о повсеместном принятии некоторых атрибутов информации, выражающих ее способность быть юридически значимой. Данное свойство информации особенно актуально в системах электронных платежей, где осуществляется операция по переводу денежных средств. Исходя из сказанного, можно сформулировать некоторые требования к атрибутам информации, выражающим ее свойство быть юридически значимой. Информацию необходимо сформировать таким образом, чтобы с формальной точки зрения было определено ясно, что только отправитель, которому принадлежит данный платежный документ, мог его создать.

Неотслеживаемость – это способность совершать некоторые действия в *ИТС* незаметно для других объектов. Актуальность данного требования стала очевидной благодаря появлению таких понятий, как *электронные деньги* и *Internet-banking*. Так, для авторизации доступа к электронной платежной системе пользователь должен предоставить некоторые сведения, однозначно его идентифицирующие. По мере стремительного развития данных систем может появиться реальная опасность, что, например, все платежные операции будут контролироваться, тем самым возникнут условия для тотального слежения за пользователями *ИТС*.

Существует несколько путей решения проблемы неотслеживаемости:

- запрещение с помощью законодательных актов всякого тотального слежения за пользователями *ИТС*;
- применение криптографических методов для поддержания неотслеживаемости.

Как уже говорилось, информационная безопасность может рассматриваться не только по отношению к некоторым конфиденциальным сведениям, но и по отношению к способности *ИТС* выполнять заданные функции.

Основные задачи, решаемые в рамках информационной безопасности по отношению к работоспособности *ИТС*, должны обеспечивать защиту от:

- нарушения функционирования телекоммуникационной системы, выражающегося в воздействии на информационные каналы, каналы сигнализации, управления и удаленной загрузки баз данных

коммутационного оборудования, системное и прикладное программное обеспечение;

- несанкционированного доступа к информационным ресурсам и от попыток использования ресурсов сети, приводящих к утечки данных, нарушению целостности сети и информации, изменению функционирования подсистемы распределения информации, доступности баз данных;

- разрушения встраиваемых и внешних средств защиты;

- неправомочных действий пользователей и обслуживающего персонала сети.

Приоритеты среди перечисленных задач информационной безопасности определяются индивидуально для каждой конкретной *ИТС* и зависят от требований, предъявляемых непосредственно к информационным системам.

Следует учесть, что с точки зрения государственных структур мероприятия по защите в первую очередь призваны обеспечить конфиденциальность, целостность и доступность информации. Понятно, что для режимных государственных организаций на первом месте всегда стоит конфиденциальность сведений, а целостность понимается исключительно как их неизменность. Коммерческим структурам, вероятно, важнее всего целостность и доступность данных и услуг по их обработке. По сравнению с государственными, коммерческие организации более открыты и динамичны, поэтому вероятные угрозы для них отличаются не только количеством, но и качеством.

Для решения задачи информационной безопасности в *ИТС* необходимо:

- защитить информацию при ее хранении, обработке и передаче по сети;

- подтвердить подлинность объектов данных и пользователей (аутентификация сторон, устанавливающих связь);

- обнаружение и предупреждение нарушение целостности объектов данных;

- защитить конфиденциальную информацию от утечки и от внедренных электронных устройств съема информации;

- защитить программные продукты от внедрения программных закладок и вирусов;

- защитить от несанкционированного доступа к информационным ресурсам и техническим средствам сети, в том числе и к средствам

управления, чтобы предотвратить снижения уровня защищенности информации и самой сети в целом;

- организовать требующиеся мероприятия, направленные на обеспечение сохранности конфиденциальных данных;

- защитить технические устройства и помещения.

Конкретная реализация общих принципов обеспечения информационной безопасности может выражаться в организационных либо технических мерах защиты информации.

Следует отметить, что объем мероприятий по защите обрабатываемых и передаваемых данных зависит, прежде всего, от величины предполагаемого ущерба. Эта величина может выражаться в прямой (например, затраты на покупку нового программного обеспечения в случае нарушения его целостности) или в опосредованной (например, затраты от простоя информационной системы банка) форме. Правда, в некоторых ситуациях рассчитать величину ущерба затруднительно (например, случае утечки государственной тайны).

1.2. РОЛЬ КРИПТОГРАФИЧЕСКИХ ПРОТОКОЛОВ В ОБЩЕЙ ЗАДАЧЕ ОБЕСПЕЧЕНИЯ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Основу обеспечения информационной безопасности в *ИТС* составляют *криптографические методы* и *средства защиты информации*. Следует учесть, что наиболее надежную защиту можно обеспечить только с помощью комплексного подхода, то есть решение задачи должно представлять собой совокупность организационно-технических и криптографических мероприятий.

В основе криптографических методов лежит понятие *криптографического преобразования информации*, производимого по определенным математическим законам, с целью исключить доступ к данной информации посторонних пользователей, а также с целью обеспечения невозможности бесконтрольного изменения информации со стороны тех же самых лиц.

Применение криптографических методов защиты обеспечивает решение основных задач информационной безопасности. Этого можно добиться путем реализации следующих криптографических методов защиты как конфиденциальной и служебной информации, так и информационных ресурсов в целом:

- шифрование всего информационного трафика, передающегося через открытые сети передачи данных, и отдельных сообщений;

- криптографическая аутентификация устанавливающих связь разноуровневых объектов (имеется в виду уровни модели взаимодействия открытых систем);

- защита несущего данные трафика *средствами имитозащиты* (защиты от навязывания ложных сообщений) и *электронно-цифровой подписи* (*ЭЦП*) с целью обеспечения целостности и достоверности передаваемой информации;

- шифрование данных, представленных в виде файлов либо хранящихся в базе данных;

- контроль целостности программного обеспечения путем применения криптографически стойких контрольных сумм;

- применение *ЭЦП* для обеспечения юридической значимости платежных документов; применение *затемняющей цифровой подписи* для обеспечения неотслеживаемости действий клиента в платежных системах, основанных на понятии электронных денег.

При реализации большинства из приведенных методов криптографической защиты возникает необходимость обмена некоторой информацией. Например, аутентификация объектов *ИТС* сопровождается обменом идентифицирующей и аутентифицирующей информации.

В общем случае взаимодействие объектов (субъектов) подобных систем всегда сопровождается соблюдением некоторых договоренностей, называемых *протоколом*. Формально протоколом будем считать последовательность действий объектов (субъектов) для достижения определенной цели. Она в данном случае определяет структуру и специфику применения протокола.

В свою очередь, *криптографическими протоколами* будем называть те, в которых участники для достижения определенной цели используют криптографические преобразования информации.

Перечислим основные задачи обеспечения информационной безопасности, которые решаются с помощью криптографических протоколов:

- обмен ключевой информации с последующей установкой защищенного обмена данными; при этом не существует никаких предположений, общались ли предварительно между собой стороны, обменивающиеся ключами (например, без использования криптографических протоколов невозможно было создать системы распределения ключевой информации в распределенных сетях передачи данных);

- аутентификация сторон, устанавливающих связь;

- авторизация пользователей при доступе к телекоммуникационным и информационным службам.

На сегодняшний день благодаря повсеместному применению открытых сетей передачи данных, таких как *Internet*, и построенных на их основе сетей – *intranet* и *extranet* – криптографические протоколы нашли широкое применение для решения разнообразного круга задач и обеспечения, постоянно расширяющихся услуг, предоставляемых пользователям таких сетей.

Кроме вышеперечисленных классических областей применения протоколов существует широкий круг специфических задач, также решаемых с помощью соответствующих криптографических протоколов. Это, прежде всего раскрытие части сведений без обнародования самого секрета в его подлинном объеме, а также частичное раскрытие секрета. Так, например, участники могут для достижения какой-то общей цели сообщить друг другу часть своей информации или объединить усилия для раскрытия секрета, неизвестного каждому из них в отдельности.

Стремительное развитие криптографических протоколов в большей степени стимулируется развитием систем электронных платежей, интеллектуальных карточек, появлением электронных денег и т.д.

Поскольку на сегодняшний день основным криптографическим средством защиты информации в *Internet* являются протоколы, можно констатировать, что развитие подобных средств защиты коммерческих тайн будет продолжаться и в количественном, и в качественном отношении.

Многогранность применения криптографических протоколов в решении задачи обеспечения информационной безопасности, как в локальных, так и в распределенных информационных системах, приводит к необходимости детального рассмотрения их основных типов, вопросов практического применения таких протоколов и построения на их основе специальных информационных систем.

1.3. ОБЩИЕ СВЕДЕНИЯ О КЛАССИЧЕСКОЙ КРИПТОЛОГИИ, КРИПТОГРАФИИ И КРИПТОГРАФИЧЕСКОМ АНАЛИЗЕ

1.3.1. Общие сведения о криптологии

Криптология - это область знаний, изучающая тайнопись (*криптографию*) и методы её раскрытия (*криптографический анализ*), которая по меткому выражению *Рональда Ривеста*, (профессора Массачусетского технологического института – *MIT*) и одного из авторов знаменитой криптографической системы *RSA*, является “повивальной бабкой всей *computer science* вообще” [2, 12]. Название криптологии произошло от греческого языка (*криптос* – тайный, *логос* – наука или слово).

Построение современной криптологии как науки основывается на совокупности фундаментальных понятий и фактов математики, физики, теории информации и сложности вычислений, природно очень сложных для всестороннего и глубокого осмысления даже профессионалами. Однако, несмотря на органически присущую ей сложность, многие теоретические достижения криптологии сейчас широко используются в нашей насыщенной информационными технологиями жизни, например: в пластиковых *smart*-картах, в электронной почте, в системах банковских платежей, при электронной торговле через *Internet*, в системах электронного документооборота, при ведении баз данных, системах электронного голосования и др. Подобное соотношение общей внутренней сложности и практической применимости для теоретической науки, по-видимому, уникально.

Криптография – это раздел прикладной математики, изучающий модели, методы, алгоритмы, программные и аппаратные средства преобразования информации (шифрования) в целях скрытия её содержания, предотвращения её изменения или несанкционированного использования. Другими словами, криптограф пытается найти методы обеспечения секретности и/или аутентичности (подлинности) сообщении.

Без криптографии принципиально нельзя обойтись при защите данных, передаваемых по открытым электронным каналам связи, а также там, где необходимо подтверждать целостность электронной информации или доказывать ее авторство.

Криптографический анализ объединяет математические методы нарушения конфиденциальности и аутентичности информации без знания ключей.

Существует ряд смежных, но не входящих в криптологию отраслей знания. Так обеспечением скрытности информации в информационных массивах занимается *стеганография*. Обеспечение целостности информации в условиях случайного воздействия находится в ведении *теории помехоустойчивого кодирования*. Наконец, смежной областью по отношению к криптологии являются математические методы сжатия информации.

В этом учебном пособии будут рассматриваться только основы криптографии и частично криптографический анализ для определения стойкости систем, использующих криптографические протоколы.

1.3.2. История развития криптологии

В истории криптологии условно можно выделить четыре этапа [6]: наивный, формальный, научный; компьютерный.

Для *наивной криптологии* (до начала XVI в.) характерно использование любых, обычно примитивных, способов запутывания противника относительно содержания шифруемых сообщений. На начальном этапе для защиты информации использовались методы *кодирования* и *стеганографии,* которые родственны, но не тождественны криптологии.

Большинство из используемых шифров сводились к *перестановке* или *моноалфавитной подстановке.* Одним из первых зафиксированных примеров является *шифр Цезаря*, состоящий в замене каждой буквы исходного сообщения на другую, отстоящую от нее в алфавите на определенное число позиций. Другой шифр, *полибианский квадрат*, авторство которого приписывается греческому писателю *Полибию*, является общей моноалфавитной подстановкой, которая проводится с помощью случайно заполненной алфавитом квадратной таблицей (для греческого алфавита размер составляет 5×5). Каждая буква исходного сообщения заменяется на букву, стоящую в квадрате снизу от нее.

Этап *формальной криптологии* (конец XV - начало XX вв.) связан с появлением формализованных и относительно стойких к ручному криптографическому анализу шифров. В европейских странах это произошло в эпоху Возрождения, когда развитие науки и тор-

говли вызвало спрос на надежные способы защиты информации. Важная роль на этом этапе принадлежит *Леону Батисте Альберти*, итальянскому архитектору, который одним из первых предложил *многоалфавитную подстановку.* Данный шифр, получивший имя дипломата XVI в. *Блеза Вижинера*, состоял в последовательном "*сложении*" букв исходного сообщения с ключом (процедуру можно облегчить с помощью специальной таблицы). Его работа "*Трактат о шифре*" (1466 г.) считается первой научной работой по криптологии.

Одной из первых печатных работ, в которой обобщены и сформулированы известные на тот момент алгоритмы шифрования, является труд "Полиграфия" (1508 г.) немецкого аббата *Иоганна Тритемиуса*. Ему принадлежат два небольших, но важных открытия: способ заполнения полибианского квадрата (первые позиции заполняются с помощью легко запоминаемого ключевого слова, остальные – оставшимися буквами алфавита) и шифрование пар букв (биграмм).

Простым, но стойким способом многоалфавитной замены (подстановки биграмм) является *шифр Плейфера*, который был открыт в начале XIX в. *Чарльзом Уитстоном*. Уитстону принадлежит и важное усовершенствование – шифрование "двойным квадратом". Шифры *Плейфера* и *Уитстона* использовались вплоть до первой мировой войны, так как с трудом поддавались ручному криптографическому анализу.

В XIX в. голландец *Огюст Керкгоффс* сформулировал главное требование к криптографическим системам, которое остается актуальным и поныне: *секретность шифров должна быть основана на секретности ключа, но не алгоритма.*

Наконец, последним словом в донаучной криптографии, которое обеспечило еще более высокую криптографическую стойкость, а также позволило автоматизировать (в смысле механизировать) процесс шифрования стали *роторные криптосистемы.*

Одной из первых подобных систем стала изобретенная в 1790 г. *Томасом Джефферсоном*, будущим президентом США, механическая машина. Многоалфавитная подстановка с помощью роторной машины реализуется вариацией взаимного положения вращающихся роторов, каждый из которых осуществляет "прошитую" в нем подстановку.

Практическое распространение роторной машины получили только в начале XX в. Одной из первых практически используемых машин, стала немецкая *Enigma,* разработанная в 1917 г. *Эдвардом Хеберном* и усовершенствованная *Артуром Кирхом*. Роторные машины активно использовались во время второй мировой войны. Помимо немецкой машины *Enigma* использовались также устройства *Sigaba* (США), *Typex* (Великобритания), *Red, Orange* и *Purple* (Япония). Роторные системы – вершина формальной криптографии, так как относительно просто реализовывали очень стойкие шифры. Успешные криптографические атаки на роторные системы стали возможны только с появлением *электронных вычислительных машин* (*ЭВМ*) в начале 40-х гг. XX в.

Главная отличительная черта *научной криптологии* (1930–60-е гг.) – появление криптографических систем со строгим математическим обоснованием криптографической стойкости. К началу 30-х гг. окончательно сформировались разделы математики, являющиеся научной основой криптологии: теория вероятностей и математическая статистика, общая алгебра, теория чисел, начали активно развиваться теория алгоритмов, теория информации, кибернетика. Своеобразным водоразделом стала работа *Клода Шеннона* "*Теория связи в секретных системах*" (1949 г.), которая подвела научную базу под криптографию и криптографический анализ. С этого времени стали говорить о *криптологии* – как о науке преобразовании информации для обеспечения ее секретности. Этап развития криптографии и криптографического анализа до 1949 г. стали называть *донаучной криптологией*. *Шеннон* ввел понятия "*рассеивание*" и "*перемешивание*", обосновал возможность создания сколь угодно стойких криптографических систем.

В 1960-х гг. ведущие криптографические школы подошли к созданию *блочных шифров,* еще более стойких по сравнению с роторными криптографическими системами, однако допускающих практическую реализацию только в виде цифровых электронных устройств.

Компьютерная криптография (с 1970-х гг.) обязана своим появлением вычислительным средствам с производительностью, достаточной для реализации криптографических систем, обеспечивающих при большой скорости шифрования на несколько порядков более высокую криптографическую стойкость, чем *ручные* и *механические шифры*.

Первым классом криптографических систем, практическое применение которых стало возможно с появлением мощных и компактных вычислительных средств, стали блочные шифры. В 70-е гг. XX в. был разработан *американский стандарт шифрования DES – Data Encryption Standard (Стандарт шифрування даних)* принятый в 1978 г. Один из его авторов, *Хорст Фейстель* (сотрудник *IBM),* описал модель блочных шифров, на основе которой были построены другие, более стойкие симметричные криптосистемы, в том числе *стандарт шифрования ГОСТ 28147-89*, который был еще разработан в СССР в 1989 г.

С появлением *DES* обогатился и криптографический анализ, для атак на американский алгоритм был создано несколько новых видов криптографического анализа (линейный, дифференциальный и т.д.), практическая реализация которых опять же была возможна только с появлением мощных вычислительных систем.

В середине 70-х гг. XX столетия произошел настоящий прорыв в современной криптографии – *появление асимметричных криптографических систем,* которые не требовали передачи секретного ключа между сторонами. Здесь отправной точкой принято считать работу, опубликованную *Уитфилдом Диффи* и *Мартином Хеллманом* в 1976г. под названием “Новые направления в современной криптографии”. В ней впервые сформулированы принципы обмена шифрованной информацией без обмена секретным ключом. Независимо к идее асимметричных криптографических систем подошел *Ральф Меркли*. Несколькими годами позже *Рон Ривест, Ади Шамир* и *Леонард Адлеман* открыли систему *RSA* – первую практическую асимметричную криптографическую систему, стойкость которой была основана на проблеме факторизации больших чисел. *Асимметричная криптография* открыла сразу несколько новых прикладных направлений, в частности системы *электронной цифровой подписи* и *электронных денег.*

В 1980-90-е гг. появились совершенно новые направления криптографии: *вероятностное шифрование, квантовая криптография* и другие. Осознание их практической ценности еще впереди. Актуальной остается и задача совершенствования симметричных криптографических систем. В этот же период были разработаны нефейстелевские шифры *(SAFER, RC6* и др.), а в 2000г. после открытого международного конкурса был принят новый национальный стан-

дарт шифрования США – *AES(Advanced Encryption Standard* – Усовершенствованный стандарт шифрования).

Криптография является одним из наиболее мощных средств обеспечения конфиденциальности и контроля целостности информации. Во многих отношениях она занимает центральное место среди программно-технических регуляторов безопасности. Например, для портативных компьютеров, физически защитить которые крайне трудно, только криптография позволяет гарантировать конфиденциальность информации даже в случае кражи.

1.3.3. Основные определения, используемые в криптологии

Способы и методы преобразования (шифрования) информации с целью её защиты от незаконных пользователей (от несанкционированного доступа) называются *шифрами*.

Шифрование (зашифрование) – процесс применения шифра к защищаемой информации, т.е. преобразование защищаемой информации (открытого сообщения, данных) в шифрованное сообщение (шифрованные данные) с помощью определенных правил, содержащихся в шифре.

Расшифрование – процесс, обратный шифрованию, т.е. преобразование шифрованного сообщения в защищаемую информацию с помощью определенных правил, содержащихся в шифре.

Ключ – это важнейший компонент шифра, отвечающий за выбор преобразования, применяемого для зашифрования конкретного сообщения. Обычно ключ представляет собой некоторую буквенную или числовую последовательность. Эта последовательность как бы "настраивает" алгоритм шифрования.

В некоторых источниках отдельно выделяют термин "*дешифрование*", подразумевая под этим восстановление исходного сообщения на основе шифрованной без знания ключа, то есть методами криптографического анализа. В дальнейшем будем считать расшифрование и дешифрование синонимами. Также термин шифрование (в узком смысле) используется как синоним зашифрования. Однако неверно в качестве синонима шифрования использовать термин "*кодирование*" (а вместо "*шифра*" – "*код*"), так как под кодированием обычно понимают представление информации в виде знаков (букв алфавита).

Протокол – это последовательность шагов, которые предпринимают две или большее количество сторон для совместного решения задачи. Все шаги следуют в порядке строгой очередности, и ни один из них не может быть сделан прежде, чем закончится предыдущий. Кроме того, любой протокол подразумевает участие, по крайней мере, двух сторон. И, наконец, протокол обязательно предназначен для достижения какой-то цели.

Протокол, основе которого лежит криптографический алгоритм называется *криптографическим протоколом*. Однако, целью криптографического протокола зачастую является не только сохранение информации втайне от посторонних. Участники криптографического протокола могут быть близкими друзьями, у которых нет друг от друга секретов, но могут являться настолько непримиримыми врагами, что каждый из них отказывается сообщить другому, например, какое сегодня число. Тем не менее, им может понадобиться поставить сноп подписи под совместным договором или удостоверить свою личность. В этом случае криптография нужна, чтобы предотвратить или обнаружить, подслушивание посторонними лицами, не являющимися участниками протокола, а также не допустить мошенничества. Поэтому часто требуется, чтобы криптографический протокол обеспечивал следующее: его участники не могут сделать или узнать больше того, что определено протоколом.

Основой любого криптографического протокола является так называемые *криптографические алгоритмы*.

Каждое преобразование однозначно определяется *ключом* и описывается некоторым *криптографическим алгоритмом*. Один и тот же криптографический алгоритм может применяться для шифрования в различных режимах. Тем самым реализуются различные способы шифрования. Каждый режим шифрования имеет как свои преимущества, так и недостатки. Поэтому выбор режима зависит от конкретной ситуации. При расшифрования используется криптографический алгоритм, который в общем случае может отличаться от алгоритма, применяемого для зашифрования сообщения. Соответственно могут различаться *ключи зашифрования* и *расшифрования*. Пару алгоритмов зашифрования и расшифрования обычно называют *криптографической системой*, а реализующие их устройства – шифровальными устройствами.

1.3.4. Обобщенная схема криптографической системы

Криптографическая система – это система, реализованная программно, аппаратно или программно – аппаратно и осуществляющая криптографическое преобразование информации с целью ее защиты [8, 22].

Предположим, что отправитель хочет послать сообщение получателю. Более того, этот отправитель хочет послать свое сообщение безопасно: он хочет быть уверен, что злоумышленник перехвативший носитель не сможет узнать содержание сообщения. Само передаваемое сообщение называется *открытым сообщением (текстом, данными)*. Изменение вида сообщения с целью скрытия его сути называется *зашифрованием*. Зашифрованное сообщение называется *шифротекстом (данными, шифрограммой*). Процесс преобразования зашифрованного сообщения в открытое сообщение называется *расшифрованием* (дешифрованием при криптографическом анализе).

Обобщенная схема криптографического преобразования, обеспечивающая зашифрование передаваемой информации (сообщений, текстов, данных) и ее расшифрование, показана на рис. 1.1.

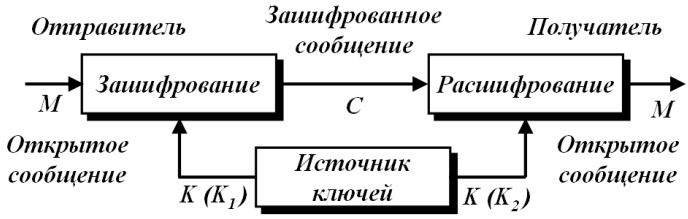


Рис. 1.1. Обобщенная схема криптографического преобразования

Обозначим открытый текст как *M* (от *message* – сообщение), или *P* (от *plaintext* – открытый текст). Это может быть поток битов, текстовый файл, битовое изображение, оцифрованный звук, цифровое видеоизображение и др.

Обозначим шифротекст как *C* (от *ciphertext* – *зашифрованный текст*). Это тоже двоичные данные, иногда того же размера, что и *M,* иногда больше. Если шифрование сопровождается сжатием, *C* может быть меньше *M*. Однако само шифрование не обеспечивает

сжатие информации. *Функция шифрования E* действует на открытый текст, создавая зашифрованный текст:

$$C = E(M). \tag{1.1}$$

Процесс восстановления открытого текста по зашифрованному тексту называется *расшифрованием* и выполняется с помощью функции расшифрования *D*:

$$M = D(C). \tag{1.2}$$

Поскольку смыслом зашифрования и последующего расшифрования сообщения является восстановление первоначального открытого текста, то должно выполняться следующее равенство:

$$M = D(E(M)).$$

Криптографический алгоритм, также называемый шифром, представляет собой математическую функцию (например, функции *E* и *D* в выражениях (1.1) и (1.2)), используемые для зашифрования и расшифрования.

Если безопасность алгоритма основана на сохранении самого алгоритма в тайне, это *ограниченный алгоритм*. Ограниченные алгоритмы представляют только исторический интерес, но они совершенно не соответствуют сегодняшним стандартам. Большая или изменяющаяся группа пользователей не может использовать такие алгоритмы, так как всякий раз, когда пользователь покидает группу, ее члены должны переходить на другой алгоритм. Алгоритм должен быть заменен, если кто-нибудь извне случайно узнает секрет.

Современная криптография решает эти проблемы с помощью ключа *K*. *Ключ* – это конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования данных, обеспечивающее выбор только одного варианта из всех возможных для данного алгоритма. Множество возможных ключей называют *пространством ключей*.

С учетом использования ключа, функции зашифрования (1.1) и расшифрования (1.2) запишутся как:

$$C = E_K(M) \tag{1.3}$$

и

$$M = D_K(C), \tag{1.4}$$

при этом должно выполняться:

$$M = D_K(E_K(M)). \quad (1.5)$$

Для некоторых алгоритмов при зашифровании и расшифровании используются различные ключи. В этом случае выражения (1.3) и (1.4) запишутся в таком виде:

$$C = E_{K_1}(M), \quad M = D_{K_2}(C), \quad (1.6)$$

а равенство (1.5):

$$M = D_{K_2}(E_{K_1}(M)).$$

В качестве информации, подлежащей зашифрованию и расшифрованию будут рассматриваться тексты (сообщения), построенные на некотором *алфавите*. Под этими терминами понимается следующее. *Алфавит* – конечное множество используемых для кодирования информации знаков. *Текст (сообщение, данные)* – упорядоченный набор из элементов алфавита.

В качестве примеров алфавитов, используемых в современных информационных системах можно привести следующие:

- алфавит Z_{26} – *26* букв английского алфавита (исключая пробел);
- алфавит Z_{32} – *32* буквы русского алфавита (исключая "ё") и пробел;
- алфавит Z_{256} – *256* символов, входящие в стандартные коды *ASCII*;
- алфавит Z_{16} – *16* символов *{0,1,…,f}* шестнадцатеричного алфавита;
- алфавит Z_2 – *2* символа *{0,1}* двоичного алфавита.

Безопасность алгоритмов, описываемых выражениями (1.3), (1.4) и (1.6), полностью основана на ключах, а не на деталях алгоритма. Это значит, что алгоритм может быть опубликован и проанализирован. Продукты, использующие этот алгоритм, могут широко тиражироваться. Фундаментальное правило криптографического анализа, впервые сформулированное голландцем *О. Керкгоффсом* еще в XIX веке, заключается в том, что стойкость шифра (криптографической системы) должна определяться только степенью секретности ключа. Иными словами, *правило Керкгоффса* состоит в том, что весь алгоритм зашифрования и расшифрования,

кроме секретного ключа, известен криптографическому аналитику противника. Это обусловлено тем, что криптографическая система, реализующая семейство криптографических преобразований, обычно рассматривается как открытая система. Такой подход отражает очень важный принцип технологии защиты информации: защищенность системы не должна зависеть от секретности чего-либо такого, что невозможно быстро изменить в случае утечки секретной (конфиденциальной) информации. Обычно криптографическая система представляет собой совокупность аппаратных и программных средств, которую можно изменить только при значительных затратах времени и средств, тогда как ключ является легко изменяемым объектом. Именно поэтому стойкость криптографической системы должна определяться только степенью секретности ключа и стойкости самого шифра противостоять различного рода атакам.

1.4. ОСНОВЫ ТЕОРИИ ЗАСЕКРЕЧЕННОЙ СВЯЗИ К. ШЕННОНА

Прежде чем перейти к рассмотрению криптографических алгоритмов, необходимо уделить внимание вопросам, которые в рамках криптографии давно признаются классическими, а именно – основам построения *систем засекреченной связи*.

Под *системой засекреченной связи* будем понимать систему передачи информации, в которой смысл передаваемой информации скрывается с помощью криптографических преобразований [34]. При этом сам факт передачи информации не утаивается. В основе каждой системы засекреченной связи – использование алгоритмов шифрования как основного средства сохранения конфиденциальности.

К. Шеннон рассмотрел модель (рис. 1.2), в которой источник сообщений порождает открытый текст *M*. Источник ключей генерирует ключ *K*.

Зашифрующее устройство преобразовывает открытый текст *M* с помощью ключа *K* в зашифрованный текст *C: C* = $E_K(M)$. Устройство расшифрования, получив зашифрованное сообщение *C,* выполняет обратную операцию: $M = D_K(C)$.

Задачей криптографического аналитика противника является получение открытого текста и/или ключа на основе анализа зашифрованного текста.

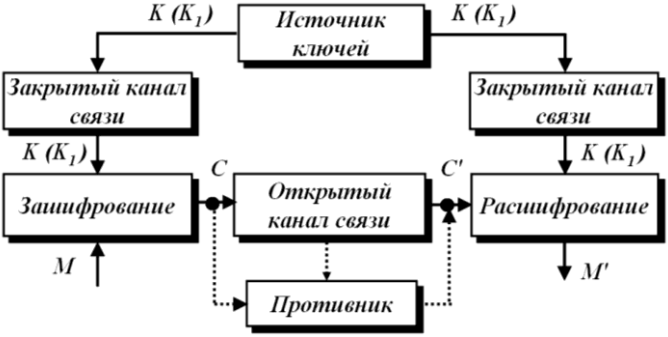


Рис. 1.2. Модель системы передачи шифрованных сообщений

Шеннон рассмотрел вопросы теоретической и практической секретности. Для определения теоретической секретности Шеннон сформулировал следующие вопросы:

1. Насколько устойчива система, если криптографический аналитик противника не ограничен временем и обладает всеми необходимыми средствами для анализа шифрограмм?

2. Имеет ли шифрограмм единственное решение?

3. Какой объем шифрограмм необходимо перехватить криптографическому аналитику, чтобы решение стало единственным?

Для ответа на эти вопросы *Шеннон* ввел понятие *совершенной секретности* с помощью следующего условия: для всех *C* апостериорные вероятности равны априорным вероятностям, т.е. перехват зашифрованного сообщения не дает криптографическому аналитику противника никакой информации. По *теореме Байеса*:

$$p(M,C) = p(M)\cdot p(C/M) = p(C)\cdot p(M/C), \qquad (1.7)$$

где $p(M)$ – априорная вероятность сообщения M; $p(C/M)$ – условная вероятность криптограммы C при условии, что выбрано сообщение M, т.е. сумма вероятностей всех тех ключей, которые переводят сообщение M в шифрограмму C; $p(C)$ – вероятность получения шифрограммы C; $p(M/C)$ – апостериорная вероятность сообщения M при условии, что перехвачена шифрограмма C.

Из выражения (1.7) следует, что:

$$p(M/C) = \frac{p(M)\cdot p(C/M)}{p(C)}. \qquad (1.8)$$

Для совершенной секретности криптографической системы значения $p(M/C)$ и $p(M)$ должны быть равны для всех M и C, т.е. $p(M/C) = p(M)$. Следовательно, из анализа (1.8), должно быть выполнено одно из равенств:

- $p(M) = 0$ (это решение должно быть отброшено, так как требуется, чтобы равенство осуществлялось при любых значениях $p(M)$);

- $p(C/M) = p(C)$ для любых M и C.

Наоборот, если $p(C/M) = p(C)$, то $p(M/C) = p(M)$, и система совершенно секретна. Таким образом, можно сформулировать следующее: необходимое и достаточное условие для совершенной секретности заключается в том, что $p(C/M) = p(C)$ для всех M и C, то есть $p(C/M)$ не должно зависеть от M.

Другими словами, полная вероятность всех ключей, переводящих сообщение M_i, в данное зашифрованное сообщение C, равна полной вероятности всех ключей, переводящих сообщение M_j в то же самое зашифрованное сообщение C для всех M_i, M_j и C.

Далее, должно существовать по крайней мере столько же зашифрованных сообщений C, сколько и сообщений M, так как для фиксированного i отображение E_i дает взаимно-однозначное соответствие между всеми M и некоторыми из C. Для совершенно секретных систем для каждого из этих C и любого M

$$p(C/M) = p(C) \neq 0.$$

Следовательно, найдется по крайней мере один ключ, отображающий данное M в любое из C. Но все ключи, отображающие фиксированное M в различные C, должны быть различными. Поэтому, *число различных ключей должно быть не меньше числа сообщений M.*

Как показывает следующий пример, можно получить совершенную секретность, когда число сообщений точно равно числу ключей. Пусть M_i пронумерованы числами от 1 до n, так же как и C_i, и пусть используются n ключей. Тогда:

$$E_i(M_j) = C_s,$$

где $s = (i + j) \bmod n$.

В этом случае оказывается справедливым равенство $p(M/C) = 1/n = p(C)$ и система является совершенно секретной. Один пример такой системы показан на рис. 1.3, где $s = (i + j - 1) \bmod 5$.

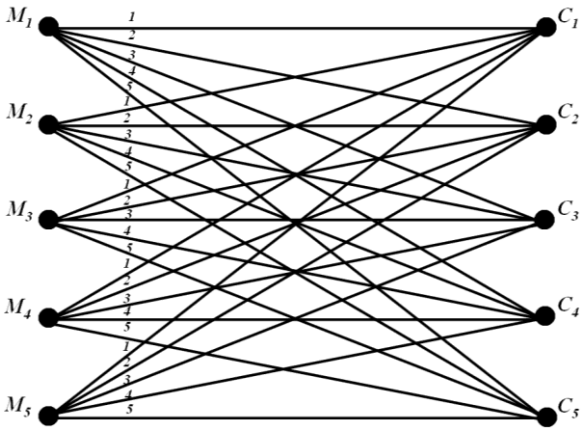


Рис. 1.3. Совершенная криптографическая система

Совершенно секретные системы, в которых число зашифрованных сообщений равно числу исходных сообщений, а также числу ключей, характеризуются следующими двумя свойствами:

1) каждое *M* связывается с каждым *C* только одной линией;

2) все ключи равновероятны.

Таким образом, матричное представление такой системы является "*латинским квадратом*".

В "*Математической теории связи*" показано, что количественно информацию удобно измерять с помощью энтропии. Если имеется некоторая совокупность возможностей с вероятностями p_1, $p_2,...,p_n$, то энтропия представляется выражением

$$H = -\sum p_i \cdot log\, p_i \,.$$

Секретная система включает в себя два статистических выбора: выбор сообщения и выбор ключа. Можно измерять количество информации, создаваемой при выборе сообщения, через *H(M)*

$$H(M) = -\sum P(M) \cdot log\, P(M),$$

где суммирование выполняется по всем возможным сообщениям.

Аналогично, неопределенность, связанная с выбором ключа, представляется выражением

$$H(K) = -\sum P(K) \cdot log\, P(K),$$

В совершенно секретных системах описанного выше типа количество информации в сообщении равно самое большее *log n* (эта величина достигается для равновероятных сообщений). Эта информация может быть скрыта полностью лишь тогда, когда неопределенность ключа не меньше *log n*. Это является первым примером общего принципа, который будет часто встречаться ниже: существует предел, которого нельзя превзойти при заданной неопределенности ключа – количество неопределенности, которое может быть введено в решение, не может быть больше, чем неопределенность ключа.

Положение несколько усложняется, если число сообщений бесконечно. Предположим, например, что сообщения порождаются соответствующим *марковским процессом* в виде бесконечной последовательности букв. Ясно, что никакой конечный ключ не даст совершенной секретности. Предположим тогда, что источник ключа порождает ключ аналогичным образом, т.е. как бесконечную последовательность символов.

Предположим далее, что для зашифрования и расшифрования сообщения длины L_M требуется только определенная длина ключа L_K. Пусть логарифм числа букв в алфавите сообщений будет R_M, а такой же логарифм для ключа – R_K. Тогда из рассуждений для конечного случая, очевидно, следует, что для совершенной секретности требуется, чтобы выполнялось неравенство

$$R_M \cdot L_M \leq R_K \cdot L_K.$$

Эти выводы делаются в предположении, что априорные вероятности сообщений неизвестны или произвольны. В этом случае ключ, требуемый для того, чтобы имела место совершенная секретность, зависит от полного числа возможных сообщений.

Можно было бы ожидать, что если в пространстве сообщений имеются фиксированные известные статистические связи, так что имеется определенная скорость создания **с**ообщений R в смысле, принятом в "*Математической теории связи*", то необходимый объем ключа можно было бы снизить в среднем в R/R_M раз, и это действительно верно. В самом деле, сообщение можно пропустить через преобразователь, который устраняет избыточность и уменьшает среднюю длину сообщения как раз во столько раз. Затем к результату можно применить *шифр Вернама*. Очевидно, что объем ключа,

используемого на букву сообщения, статистически уменьшается на множитель R/R_M, и в этом случае источник ключа и источник сообщений в точности согласован – один бит ключа полностью скрывает один бит информации сообщения. С помощью методов, использованных в "*Математической теории связи*", легко также показать, что это лучшее, чего можно достигнуть.

1.5. КЛАССИФИКАЦИЯ МЕТОДОВ ШИФРОВАНИЯ СООБЩЕНИЙ

Классическая или одноключевая криптография опирается на использование симметричных алгоритмов шифрования, в которых зашифрование и расшифрование отличаются только порядком исполнения и направлением некоторых шагов. Эти алгоритмы используют тот самый секретный элемент (ключ), и второе действие (расшифрование) является простым обращением первой (зашифрование). Поэтому, обычно каждый из участников обмена может, как зашифровать, так и расшифровать сообщение. Схематическая структура такой системы показана на рис. 1.1.

На передающей стороне есть источник сообщений *M* и источник ключей *K*. Источник ключей выбирает конкретный ключ *K* среди всех возможных ключей данной системы. Этот ключ *K* передается некоторым способом принимающей стороне, причем предполагается, что его невозможно перехватить, например, ключ передается специальным курьером (поэтому симметричное шифрование называется также *шифрованием с закрытым ключом*). Источник сообщений формирует некоторое сообщение *M*, которое затем зашифровывается с использованием выбранного ключа *K*. В результате процедуры зашифрования получается зашифрованное сообщение *C* (называемое также *криптограммой*). Далее криптограмма *C* передается по каналу связи. Поскольку канал связи открытый (незащищенный), например, радиоканал или компьютерная сеть, то переданное сообщение может быть перехвачено противником. На принимающей стороне криптограмму *C* с помощью ключа *K* расшифровывают и получают входящее сообщение *M*.

Через большую избыточность естественных языков непосредственно в зашифрованное сообщение чрезвычайно сложно внести осмысленную смену, поэтому классическая криптография обеспечивает также защиту от навязывания ложных данных. Если же есте-

ственной избыточности оказывается недостаточно для надежной защиты сообщение от модификации, избыточность может быть искусственно увеличена путем добавления к сообщению специальной контрольной комбинации, называемой *имитовставкой*.

Известны различные методы шифрования (рис. 1.4).

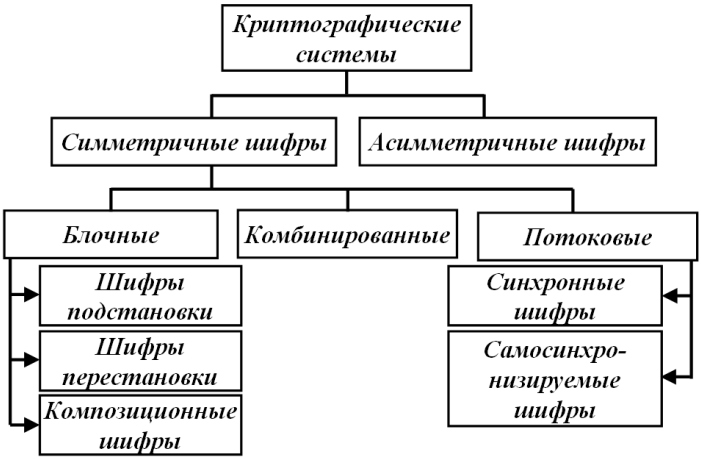


Рис. 1.4. Классификация методов шифрования сообщений

На практике часто используются алгоритмы перестановки, подстановки, а также комбинированные методы.

Криптографические системы по типу алгоритма шифрования подразделяются на две большие группы (рис. 1.4): *симметричные* и *асимметричные*. Кроме того, алгоритмы делятся по типу преобразования, по способу обработки информации.

В *симметричных* криптографических системах зашифрования и расшифрования проводятся с помощью того же ключа. И соответственно этот ключ необходимо хранить в тайне (отсюда другое название симметричных криптографических систем – *криптографические системы с секретным ключом*).

В *асимметричных* криптографических системах существуют два разных ключа: один используется для зашифрования, который еще называют *открытым*, другой – для расшифрования, который называют *закрытым*. Главное отличие асимметричных криптосистем заключается в том, что даже тот, кто с помощью открытого ключа зашифровал сообщение, не сможет его самостоятельно

расшифровать без секретного ключа. Поэтому эти системы называются *асимметричными*, или *системами с открытым ключом*.

Гибридными принято называть криптографические системы, объединяющие оба типа криптографических систем. В них, как правило, текст сообщение зашифровывается с использованием симметричной криптографической системы, а секретный ключ, использованной симметричной криптографической системой, зашифровывается с использованием асимметричной криптографической системы.

По типу обработки входной информационной последовательности криптографические системы делятся на *потоковые*, в которых каждый символ или бит открытого текста преобразуется в символ шифрованного текста потоком, и *блочные*, в которых сообщения обрабатываются в виде блоков определенной длины.

В *методах перестановки* символы входного текста меняются местами друг с другом по определенному правилу. В *методах подстановки* (или замены) символы открытого текста заменяются некоторыми эквивалентами зашифрованного текста. По сути, шифры перестановки и подстановки являются кирпичиками, из которых строятся различные другие более стойкие шифры.

С целью повышения надежности зашифрования текст, зашифрованный с помощью одного метода, может быть еще раз зашифрованый с помощью другого метода. В таком случае получается *комбинированный* или *композиционный шифр*. Применяемые на практике в настоящее время блочные или симметричные потоковые шифры также относятся к комбинированным, поскольку в них используется несколько операций для зашифрования сообщения.

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

Идея, лежащая в основе с*оставных*, или *композиционных шифров*, заключается в построении криптографичностойкой системы путем многократного применения относительно простых криптографических преобразований. *К. Шеннон* предложил использовать в качестве преобразования *подстановки* (*substitution*) и *транспозиции* (*permutation*). Многократное использования этих преобразований позволяет обеспечить два свойства, которые должны быть присущи устойчивым шифрам: *рассеивание* (*diffusion*) и *перемешивание* (*confusion*).

Рассеивание предусматривает распространение влияния одного знака открытого текста, а также одного знака ключа на значительное количество знаков зашифрованного сообщения. Наличие в шифре этого свойства, с одной стороны, позволяет скрывать статистическую зависимость между знаками открытого текста, иначе говоря, перераспределить избыточность входного языка посредством распространения ее на весь текст, а с другой – не позволяет восстановить неизвестный ключ частями. Например, обычная перестановка символов позволяет скрыть частоты появления биграмм, триграмм и т.д.

Цель *перемешивания* – сделать как можно сложнее зависимость между ключом и зашифрованным сообщением. Криптографический аналитик на основе статистического анализа перемешанного текста не должен получить любое количество информации о использованном ключе. Обычно перемешивание осуществляется с помощью подстановки. Применение рассеивания и перемешивания отдельности не обеспечивает необходимую устойчивость, стойкая криптографическая система получается только в результате их совместного использования.

1.6. КРИПТОГРАФИЧЕСКИЙ АНАЛИЗ

Как уже отмечалось, криптография – наука и искусство создания секретных кодов, криптографический анализ – наука и искусство взлома этих кодов. В дополнение к изучению методов криптографии необходимо также изучить методы криптографического анализа. Это необходимо не для того, чтобы взламывать коды других людей, а чтобы оценить уязвимые места своих криптографических систем. Изучение криптографического анализа поможет создавать лучшие секретные коды. Есть четыре общих типа атак криптографического анализа, которые показаны на рис. 1.5 [8].

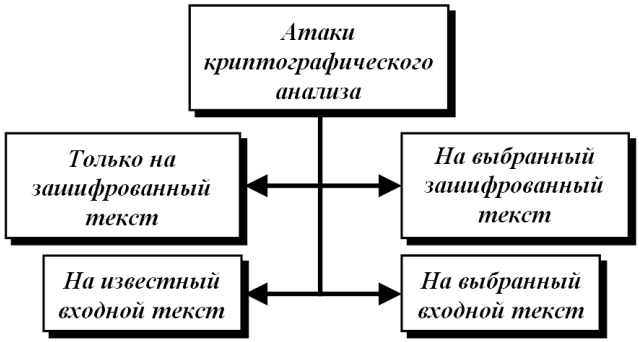


Рис. 1.5. Классификация атак криптографического анализа

1.6.1. Атака только на зашифрованный текст

В атаке только на зашифрованный текст криптографический аналитик имеет доступ только к некоторому зашифрованному тексту. Он пытается найти подходящий ключ и входной текст. При этом согласно предположению, криптографический аналитик знает алгоритм и может перехватить зашифрованный текст. Атака только на зашифрованный текст – самая вероятная, потому что криптографическому аналитику для нее нужен только сам зашифрованный текст. Шифр должен серьезно препятствовать этому типу атаки и не позволить дешифрования сообщения противником. Рис. 1.6 иллюстрирует процесс атаки.

В атаке только на зашифрованный текст могут использоваться различные методы. Рассмотрим некоторые из них.

Атака “грубой силы”

С помощью *метода “грубой силы”*, или *метода исчерпывающего ключевого поиска*, криптографический аналитик пытается использовать все возможные ключи. Предполагаем, что он знает алгоритм и знает множество ключей (список возможных ключей). Путем использование этого метода перехватывается зашифрованный текст и задействуются все возможные ключи, пока не получится входной текст. Создание атаки “грубой силы” было в прошлом трудной задачей; сегодня с помощью компьютера это стало проще.

Чтобы предотвратить атаки этого типа, число возможных ключей должно быть очень большим.

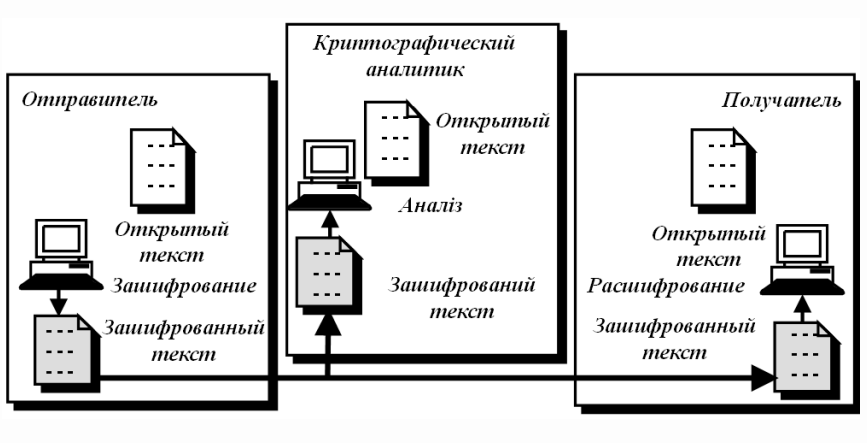


Рис. 1.6. Пояснение процесса атаки только на зашифрованный текст

Статистическая атака

Криптографический аналитик может извлечь выгоду из некоторых присущих языку входного текста характеристик, чтобы начать *статистическую атаку*. Например, известно, что буква *E* – наиболее часто используемая буква в английском тексте. Криптографический аналитик находит наиболее часто используемый символ в зашифрованном тексте и принимает, что это подходящий символ входного текста – *E*. После определения нескольких пар аналитик может найти ключ и расшифровать сообщение. Чтобы предотвратить атаки такого типа, шифр должен скрывать характеристики языка.

Атака по образцу

Некоторые шифры скрывают характеристики языка, но создают некоторые образцы в зашифрованном тексте. Криптографический аналитик может использовать атаку по образцу, чтобы взломать шифр. Поэтому важно использовать шифры, которые сделали бы просматриваемый зашифрованный текст, насколько это возможно неопределенным (абстрактным).

1.6.2. Атака на известный входной текст

При *атаке на известный входной текст* криптографический аналитик имеет доступ к некоторым пар "*входной/зашифрованный текст*" в дополнение к перехваченному зашифрованному тексту, который он хочет взломать, как показано на рис. 1.7.

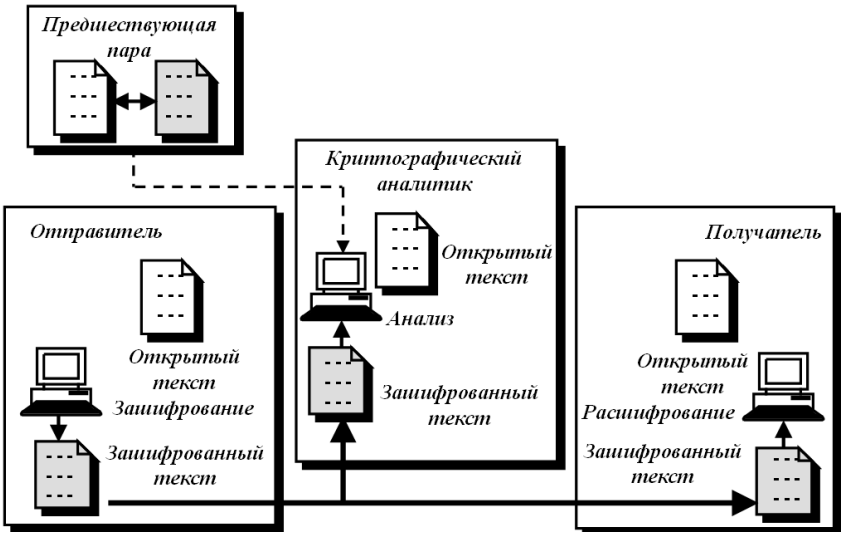


Рис. 1.7. Пояснение процесса атаки на известный входной текст

Пары входного/зашифрованного текста были собраны раньше. Например, отправитель передал секретное сообщение получателю, но позже он открыл содержание сообщения посторонним. Криптографический аналитик хранил и зашифрованный текст, и входной текст, чтобы использовать их, когда понадобится взломать следующее секретное сообщение от отправителя к получателю, предполагая, что отправитель не изменит свой ключ. Криптографический аналитик использует отношения между предыдущей парой, чтобы анализировать текущий зашифрованный текст.

Те же методы, которые используются в атаке только на зашифрованный текст, могут быть применены и здесь. Но эту атаку осуществить проще, потому что криптографический аналитик имеет больше информации для анализа. Однако может случиться, что отправитель

изменил свой ключ или не раскрывал содержания любых предыдущих сообщений, – тогда подобная атака станет невозможной.

1.6.3. Атака на выбранный входной текст

Атака с выборкой входного текста подобная атаки на входной текст, но пары "*входной/зашифрованный текст*" были выбраны и изготовленные самим нападающим. Этот процесс иллюстрирует рис. 1.8.

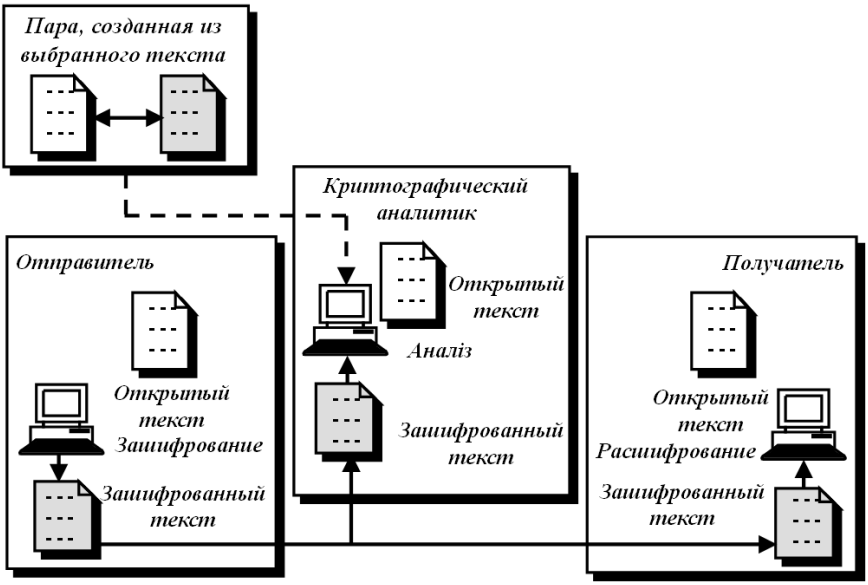


Рис. 1.8. Пояснение процесса атаки на выбранный входной текст

Это может произойти, например, если криптографический аналитик имеет доступ к компьютеру отправителя. Он выбирает определенный входной текст и создает с помощью компьютера зашифрованный текст. Конечно, он не имеет ключа, потому что ключ обычно содержится в программном обеспечении, используемом отправителем.

Этот тип атаки гораздо проще осуществить, но он наименее вероятен, поскольку подразумевает слишком много "*если*".

1.6.4. Атака на выбранный зашифрованный текст

Атака на выбранный зашифрованный текст подобна атаки на выбранный входной текст, за исключением того, что криптографический аналитик выбирает определенный зашифрованный текст и расшифровывает его, чтобы сформировать пару "*зашифрованный/исходный текст*" (это случается, когда аналитик имеет доступ к компьютеру отправителя). Рис. 1.9 показывает этот процесс.

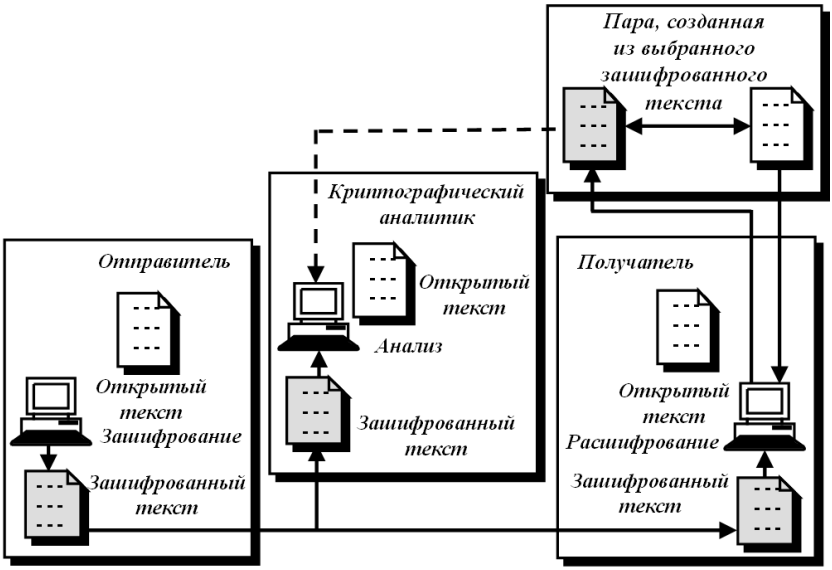


Рис. 1.9. Пояснение процесса атаки на выбранный зашифрованный текст

Криптографические атаки с использование современных криптографических систем будет рассмотрены во время их изучения в следующих разделах.

Контрольные вопросы и задания

1. Назовите проблемы, при решении которых могут использоваться криптографические методы.
2. Перечислите основные задачи защиты информации, которая предназначена для пользователя.

3. Дайте определение свойств информации: конфиденциальность, целостность; достоверность.

4. Дайте определение: шифра; ключа шифрования; криптографического алгоритма; криптографической системы.

5. Дайте определение криптографического протокола.

6. Дайте объяснение необходимости выполнения условия (1.5) при шифровании данных.

7. Сформулируйте необходимое и достаточное условия для совершенной секретности криптографической системы.

8. Дайте объяснение сущности рассеивания данных в процессе их шифрования.

9. Что является целью перемешивания данных в процессе их шифрования?

10. Что такое криптографическая атака?

11. Какие типы криптографических атак существуют?

12. Дайте характеристику атаки только на зашифрованный текст. Объясните сущность атаки “грубой силы”.

13. Дайте характеристику атаки только на зашифрованный текст. Объясните сущность статистической атаки.

14. Дайте характеристику атаки только на зашифрованный текст. Объясните сущность атаки по образцу.

15. Дайте характеристику атаки на известный входной текст.

16. Дайте характеристику атаки на выбранный входной текст.

17. Дайте характеристику атаки на выбранный зашифрованный текст.

Раздел 2

ТРАДИЦИОННЫЕ ИСТОРИЧЕСКИЕ ШИФРЫ

Традиционные шифры с симметричным ключом можно разделить на две обширные категории: *шифры подстановки* и *шифры перестановки*. В шифре подстановки один символ в зашифрованном тексте заменяется на другой символ; в шифре перестановки – меняеются местами позиции символов в исходном тексте.

2.1. ШИФРЫ ПОДСТАНОВКИ

Шифр подстановки заменяет один символ другим. Если символы в исходном тексте – символы алфавита, то одна буква заменяется другой. Например, можно заменить букву *A* буквой *D* используя английский алфавит, а букву *T* – буквой *Z*. Если символы – цифры (от *0* до *9*), то можно, например, заменить *3* на *7* и *2* на *6*. Шифры подстановки могут быть разбиты на две категории: *моноалфавитные* (*одноалфавитные*) и *многоалфавитные* шифры.

Для шифров подстановки длина алфавита открытого текста (данных) (n_M) и длина алфавита зашифрованного текста (данных) (n_C) одинакова, то есть $n_M = n_C$.

При использовании этих шифров буквы алфавита удобно отождествлять с их порядковыми номерами.

2.1.1. Моноалфавитные шифры подстановки

В *моноалфавитных шифрах* подстановки буква (или символ) в исходном тексте всегда изменяется на одну и ту же самую букву (или символ) в зашифрованном тексте независимо от его позиции в тексте. Например, если алгоритм определяет, что буква *A* в исходном тексте изменяется на букву *D*, то при этом каждая буква *A* изменяется на букву *D*. Другими словами, буквы в исходном тексте и зашифрованном тексте находятся в отношении один к одному.

Пример 2.1. Исходный текст *hello* в результате шифрования превратился в *KHOOR*. Определить, каким шифром обеспечивалось шифрование. Используем строчные символы, чтобы показать исходный текст, и заглавные буквы (символы верхнего регистра), чтобы получить зашифрованный текст.

Решение. Так как оба *l* зашифрованы как *O*, то шифр моноалфавитный.

Пример 2.2. Исходный текст *hello* в результате шифрования превратился в *ABNZF*. Определить, каким шифром обеспечивалось шифрование.

Решение. Шифр не является моноалфавитным, потому что каждая буква *l* (эль) зашифрована различными символами. Первая буква *l* (эль) зашифрована как *N*; вторая – как *Z*.

Аддитивные шифры подстановки

Самый простой моноалфавитный шифр – *аддитивный шифр*, его иногда называют *шифром сдвига*, а иногда - *шифром Цезаря*, но термин *аддитивный шифр* лучше показывает его математический смысл. Предположим, что исходный текст состоит из маленьких букв (от *a* до *z*) и зашифрованный текст состоит из заглавных букв (от *A* до *Z*). Чтобы обеспечить применение математических операций к исходному и зашифрованному текстам, присвоим каждой букве числовое значение (для нижнего и верхнего регистра), как это показано на рис. 2.1 для английского (*26* букв) и на рис. 2.2 русского языка (*32* буквы).

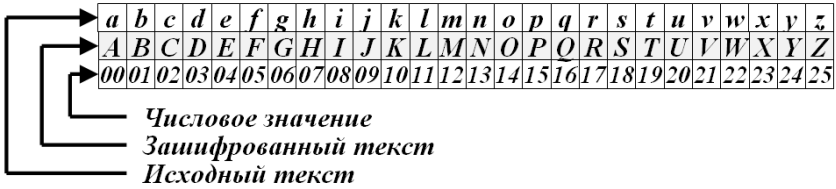


Рис. 2.1. Представление букв исходного и зашифрованного текста в Z_{26} для английского языка

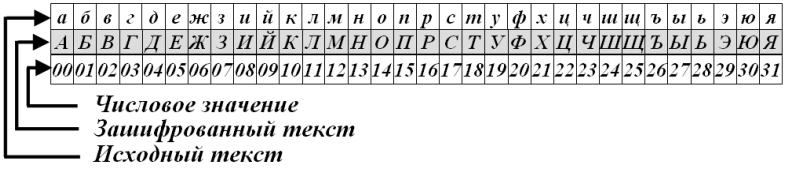


Рис. 2.2. Представление букв исходного и зашифрованного текста в Z_{32} для русского языка

На рис. 2.1 каждому символу (нижний регистр или верхний регистр) сопоставлено целое число из Z_{26}. Ключ засекречивания между отправителем и получателем – также целое число в Z_n. Алгоритм зашифрования прибавляет ключ к символу исходного текста; алгоритм расшифрования вычитает ключ из символа зашифрованного текста. Все операции проводятся в конечном поле Z_n.

Рис. 2.3 показывает процесс зашифрования и расшифрования.

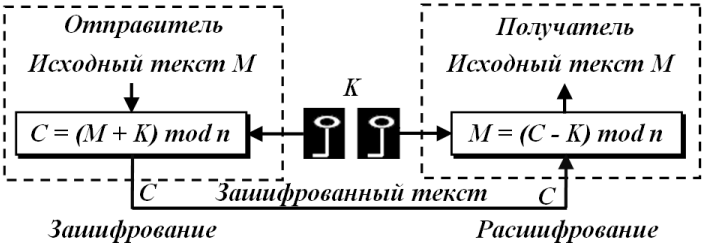


Рис. 2.4. Пояснение процессов зашифрования и расшифрования сообщений с помощью аддитивного шифра подстановки

Зашифрование текстовых сообщений с помощью аддитивного шифра подстановки осуществляется с помощью выражения:

$$C = (M + K) \bmod n, \qquad (2.1)$$

где n – длина алфавита (для английского алфавита $n = 26$); M и C – исходный и зашифрованный текст соответственно; K – ключ зашифрования (расшифрования), а расшифрования

$$M = (C - K) \bmod n. \qquad (2.2)$$

Можно легко показать, что процессы зашифрования и расшифрования являются инверсными друг другу, потому что исходный

текст, созданный получателем (M_1), тот же самый, что и тот, который передан отправителем (M):

$$M' = (C - K) \bmod n = (M + K - K) \bmod n = M.$$

Пример 2.3. Зашифровать сообщение *hello* с использованием аддитивного шифра подстановки с ключом $K = 15$.

Решение. Применяем алгоритм шифрования (2.1) к исходному тексту, буква за буквой:

$m_1 = h \rightarrow 07;$ $\quad c_1 = (07 + 15) \bmod 26 = 22 \rightarrow W;$
$m_2 = e \rightarrow 04;$ $\quad c_2 = (04 + 15) \bmod 26 = 19 \rightarrow T;$
$m_3 = l \rightarrow 11;$ $\quad c_3 = (11 + 15) \bmod 26 = 0 \rightarrow A;$
$m_4 = l \rightarrow 11;$ $\quad c_4 = (11 + 15) \bmod 26 = 0 \rightarrow A;$
$m_5 = o \rightarrow 14;$ $\quad c_4 = (14 + 15) \bmod 26 = 3 \rightarrow D,$

и в результате получим зашифрованное сообщение – *WTAAD*.

Обратите внимание, что шифр моноалфавитный, потому что два отображения одной и той же буквы исходного текста (*l*) зашифрованы как один и тот же символ (*A*).

Пример 2.4. Используя аддитивный шифр с ключом $K = 15$ расшифровать сообщение *WTAAD*.

Решение. Применяя алгоритм расшифрования (2.2) к зашифрованному тексту буква за буквой:

$c_1 = W \rightarrow 22;$ $\quad m_1 = (22 - 15) \bmod 26 = 07 \rightarrow h;$
$c_2 = T \rightarrow 19;$ $\quad m_2 = (19 - 15) \bmod 26 = 04 \rightarrow e;$
$c_3 = A \rightarrow 00;$ $\quad m_3 = (00 - 15) \bmod 26 = 11 \rightarrow l;$
$c_4 = A \rightarrow 00;$ $\quad m_4 = (00 - 15) \bmod 26 = 11 \rightarrow l;$
$c_5 = D \rightarrow 03;$ $\quad m_5 = (03 - 15) \bmod 26 = 14 \rightarrow o,$

и в результате получим исходное сообщение – *hello*.

Обратите внимание, что операции проводятся по модулю *26*, отрицательный результат должен быть отображен в Z_{26} (например, *-15* становится *11*).

Исторически аддитивные шифры назывались *шифрами сдвига*, по той причине, что алгоритм зашифрования может интерпретироваться как "*клавиша сдвига буквы вниз*", а алгоритм расшифрования может интерпретироваться как "*клавиши сдвига буквы вверх*". Например, если ключ $K = 15$, алгоритм зашифрования сдвигает букву на *15* букв вниз (к концу алфавита). Алгоритм расшифрования сдвигает букву на *15* букв вверх (к началу алфавита). Конечно,

когда достигается конец или начало алфавита, нужно двигаться по кольцу к началу (объявленные свойства операции по модулю *n*).

Юлий Цезарь использовал аддитивный шифр, чтобы связаться со своими чиновниками. По этой причине аддитивные шифры упоминаются иногда как *шифры Цезаря*. Цезарь для своей связи использовал в качестве ключа цифру *3* (*K = 3*).

Аддитивные шифры подстановки уязвимы к атакам только на зашифрованный текст, когда используется исчерпывающий перебор ключей (атака "*грубой силы*"). Множество ключей аддитивного шифра подстановки очень мало – их только *26* (для английского алфавита) и *32* (для русского алфавита). Один из ключей, нулевой, является бесполезным (зашифрованный текст будет просто соответствовать исходному тексту). Следовательно, остается только *25* возможных ключей для английского и *31* для русского алфавита. Поэтому злоумышленник может легко начать атаку "*грубой силы*" на зашифрованный текст.

Пример 2.5. Пусть злоумышленник перехватил зашифрованный текст *UVACLYFZLJBYL*. Покажем, как он может взломать шифр, используя атаку "грубой силы"

Решение. Злоумышленник пробует раскрыть текст и последовательно перебирает ключи, начиная с первого. С помощью ключа номер *7* (*K = 7*) он получает осмысленный текст "*not very secure*" (*не очень безопасный*).

Зашифрованный текст: *UVACLYFZLJBYL*

K = 1	Исходный текст: *tubkxeykiaxk*
K = 2	Исходный текст: *styajwdxjhzwj*
K = 3	Исходный текст: *rsxzivewigyvi*
K = 4	Исходный текст: *qrwyhubvhfhuh*
K = 5	Исходный текст: *pqvxgtaugewtg*
K = 6	Исходный текст: *opuwfsztfdvst*
K = 7	Исходный текст: *notverysecure*

Аддитивные шифры подстановки также могут быть объектами статистических атак. Это особенно реально, если противник перехватил зашифрованный текст большой длины. Он может воспользоваться знаниями о частоте употребления символов в конкретном языке. Табл. 2.1 показывает частоту появления определенных букв для английского текста длиной в *100* символов [29, 32].

Таблица 2.1

Частота появления букв в английском тексте

Буква	*Частота*	*Буква*	*Частота*	*Буква*	*Частота*
E	*12,7*	*D*	*4,3*	*B*	*1,0*
T	*9,1*	*L*	*4,0*	*V*	*0,09*
A	*8,2*	*C*	*2,8*	*K*	*0,08*
O	*7,5*	*U*	*2,8*	*J*	*0,02*
I	*7,0*	*W*	*2,3*	*Q*	*0,01*
N	*6,7*	*F*	*2,2*	*X*	*0,01*
S	*6,3*	*G*	*2,0*	*Z*	*0,01*
H	*6,1*	*Y*	*1,9*		
R	*6,0*	*P*	*1,5*		

Однако информации о частоте единственного символа недостаточно, и это затрудняет анализ зашифрованного текста, основанного на анализе частоты появления букв. Весьма желательно знать частоту появления комбинаций символов, а также необходимо знать частоту появления в зашифрованном тексте комбинаций с двумя или с тремя символами и сравнивать ее с частотой в языке, на котором написан исходный документ.

Наиболее употребляемые группы с двумя символами (диаграмма (*diagrams*)) и группы с тремя символами (триграмма (*trigrams*)) для английского текста показаны в табл. 2.2.

Таблица 2.2

Группы диаграмм и триграмм, основанные на их частоте появления в английском языке

Диграмма	*TH,HE,IN,ER,AN,RE,ED,ON,ES,ST,EN,AT,TO,NT,HA, ND,OU,EA,NG,AS,OR, TI,IS,ET,IT,AR,TE,SE,HI,OF*
Триграмма	*THE,ING,AND,HER,ERE,ENT,THA,NTH,WAS, ETH, FOR,DTH*

Пример 2.6. Злоумышленник перехватил следующий зашифрованный текст

XLILSYWIMWRSAJSVWEPIJSVJSYVQMPPMSRH
SPPEVWMXMWASVX-LQSVILY-
VVCFIJSVIXLIWIPPIVVIGIMZIWQSVISJJIVW.

Используя статистическую атаку, необходимо найти исходный текст.

Решение. Когда злоумышленник составит таблицу частоты букв в этом зашифрованном тексте, он получит: *I = 14, V = 13, S = 12,* и так далее. Самый частый символ – *I* – имеет *14* появлений. Это показывает, что символ *I* в зашифрованном тексте, вероятно, соответствует символу *e* в исходном тексте. Тем самым, ключ *K = 4.* Поэтому злоумышленник, используя ключ *K = 4,* расшифровывает текст и получает:

the house is now for sale for four million dollars it is worth more hurry before the seller receives more offers
(*дом теперь продается за четыре миллиона долларов, стоит поспешить, пока продавец не получил больше предложений*).

Мультипликативные шифры подстановки

В *мультипликативном шифре* подстановки алгоритм зашифрования применяет умножение исходного текста ключом, а алгоритм расшифрования применяет деление зашифрованного текста ключом, как показано на рис. 2.5.

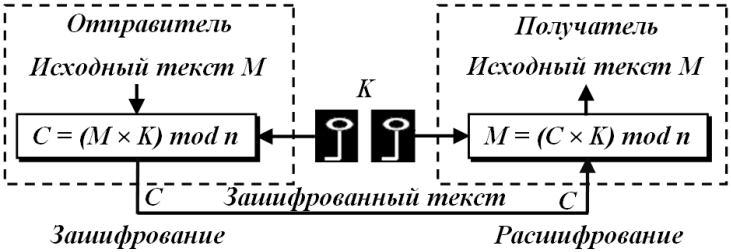


Рис. 2.5. Зашифрование и расшифрование сообщений с помощью мультипликативного шифра подстановки

Поскольку операции в мультипликативном шифре подстановки проводятся в конечном поле Z_{26}, расшифрование здесь означает умножение на мультипликативную инверсию ключа. Обратите внимание, что ключ должен принадлежать набору Z^*_n – это гарантирует, что алгоритмы зашифрования и расшифрования будут инверсны друг другу.

Зашифрование текстовых сообщений с помощью мультипликативного шифра подстановки осуществляется с помощью выражения:

$$C = (M \times K) \bmod n, \tag{2.3}$$

а расшифрование:

$$M = (C \times K^{-1}) \bmod n, \tag{2.4}$$

где K^{-1} – мультипликативная инверсия ключа K.

Пример 2.7. Определить каково множество ключей для мультипликативного шифра подстановки при использовании английского алфавита?

Решение. Ключ должен быть в конечном поле Z^*_{26}. Это множество имеет только *12* элементов: *1*, *3*, *5*, *7*, *9*, *11*, *15*, *17*, *19*, *21*, *23* и *25*. Объясняется это тем, что для значений *2*, *4*, *6*, *8*, *10*, *12*, *13*, *14*, *16*, *18*, *20*, *22* и *24* не существует мультипликативных инверсий набора Z^*_{26}.

Пример 2.8. Используя мультипликативный шифр, зашифровать сообщение *hello* с ключом $K = 7$.

Решение. Применяя алгоритм зашифрования (2.3) к зашифрованному тексту буква за буквой:

$m_1 = h \rightarrow 07;$ $\quad c_1 = (07 \times 07) \bmod 26 = 23 \rightarrow X;$
$m_2 = e \rightarrow 04;$ $\quad c_2 = (04 \times 07) \bmod 26 = 02 \rightarrow C;$
$m_3 = l \rightarrow 11;$ $\quad c_3 = (11 \times 07) \bmod 26 = 25 \rightarrow Z;$
$m_4 = l \rightarrow 11;$ $\quad c_4 = (11 \times 07) \bmod 26 = 25 \rightarrow Z;$
$m_5 = o \rightarrow 14;$ $\quad c_5 = (14 \times 07) \bmod 26 = 20 \rightarrow U,$

получим зашифрованное сообщение – *XCZZU*.

Можно комбинировать аддитивные и мультипликативные шифры подстановки, чтобы получить то, что названо *аффинным шифром* – комбинацией обоих шифров с парой ключей. Первый ключ используется мультипликативным шифром, второй – аддитивным шифром. Рис. 2.6 показывает, что аффинный шифр – фактически два шифра, применяемые один за другим.

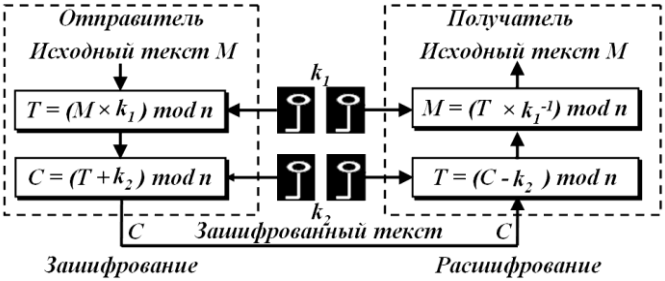


Рис. 2.6. Зашифрование и расшифрование сообщений с помощью аффинного шифра подстановки

При использовании аффинного шифра подстановки отношение между исходным M и зашифрованным текстом C, а также ключами k_1 и k_2 определяется, как это показано ниже:

$$C = (M \times k_1 + k_2) \bmod n, \tag{2.5}$$

и

$$M = ((C + (-k_2)) \times k_1^{-1}) \bmod 26, \tag{2.6}$$

где k_1^{-1} – мультипликативная инверсия k_1, а $-k_2$ – аддитивна инверсия k_2.

Можно было бы показать только одну комплексную операцию для зашифрования или расшифрования на рис. 2.6, такую, как (2.5). Однако на рис. 2.6 используется промежуточный результат (T) и указаны две отдельных операции, показывая тем самым, что всякий раз, когда используется комбинация шифров, нужно убедиться, что каждый из них имеет инверсию на другой стороне линии и что они используются в обратном порядке во время зашифрования и расшифрования. Если сложение – последнее действие во время зашифрования, то вычитание должно быть первым при расшифровании.

Пример 2.9. Аффинный шифр подстановки использует пару ключей, в которой первый ключ из Z^*_{26}, а второй – из Z_{26}. Определить каково множество ключей для аффинного шифра подстановки при использовании английского алфавита?

Решение. В примере 2.7 определено, что для английского алфавита множество ключей k_1^{-1} равно *12*. Поэтому область существования ключей для аффинного шифра будет равна

$$K = Z_{26} \times Z^*_{26} = 25 \times 12 = 300.$$

Пример 2.10. Используя аффинный шифр подстановки зашифровать сообщение *hello* с ключевой парой $k_1 = 7$ и $k_2 = 2$.

Решение. Используем $k_1 = 7$ для мультипликативного ключа и $k_2 = 2$ для аддитивного ключа, а также применяя алгоритм зашифрования подстановки (2.5) к сообщению буква за буквой:

$m_1 = h \rightarrow 7;$ $\quad c_1 = (7 \times 7 + 2) \bmod 26 = 25 \rightarrow Z;$
$m_2 = e \rightarrow 4;$ $\quad c_2 = (4 \times 7 + 2) \bmod 26 = 4 \rightarrow E;$
$m_3 = l \rightarrow 11;$ $\quad c_3 = (11 \times 7 + 2) \bmod 26 = 1 \rightarrow B;$
$m_4 = l \rightarrow 11;$ $\quad c_4 = (11 \times 7 + 2) \bmod 26 = 1 \rightarrow B;$
$m_5 = o \rightarrow 14;$ $\quad c_5 = (14 \times 7 + 2) \bmod 26 = 22 \rightarrow W,$

получим зашифрованное сообщение *ZEBBW*.

Пример 2.11. Используя аффинный шифр подстановки расшифровать сообщение *ZEBBW* с ключевой парой $k_1 = 7$ и $k_2 = 2$.

Решение. Чтобы найти символы исходного текста, прибавим аддитивную инверсию от $(-k_2) \bmod 26 = (-2) \bmod 26 = 24$ к полученному зашифрованному тексту. Потом умножим результат на мультипликативную инверсию от $k_1^{-1} \bmod 26 = 7^{-1} \bmod 26 = 15$. Поскольку k_2 имеет аддитивную инверсию в Z_{26} и k_1 имеет мультипликативную инверсию в Z^*_{26}, исходный текст – точно тот, что использовался в примере 2.10:

$c_1 = Z \rightarrow 25;$ $\quad m_1 = ((25 + 24) \times 15) \bmod 26 = 7 \rightarrow h;$
$c_2 = E \rightarrow 4;$ $\quad m_2 = ((4 + 24) \times 15) \bmod 26 = 4 \rightarrow e;$
$c_3 = B \rightarrow 1;$ $\quad m_3 = ((1 + 24) \times 15) \bmod 26 = 11 \rightarrow l;$
$c_4 = B \rightarrow 1;$ $\quad m_4 = ((1 + 24) \times 15) \bmod 26 = 11 \rightarrow l;$
$c_5 = W \rightarrow 22;$ $\quad m_5 = ((22 + 24) \times 15) \bmod 26 = 14 \rightarrow o.$

Следовательно, получим исходное сообщение *hello*.

Аддитивный шифр подстановки – частный случай аффинного шифра подстановки, при котором $k_1 = 1$. Мультипликативный шифр подстановки – частный случай аффинного шифра подстановки, в котором $k_2 = 0$.

Хотя метод "*грубой силы*" и статистической атаки могут использоваться только для зашифрованного текста, попробуем атаку на выбранный исходный текст. Предположим, что злоумышленник перехватывает следующий зашифрованный текст на английском языке:

PWUFFOGWCHFDWIWEJOUUNJORSMDWRHVCMWJUPVCCG.

Злоумышленник также очень ненадолго получает доступ к компьютеру отправителю сообщений и время, достаточное лишь для того, чтобы напечатать исходный текст с двумя символами: *et*. Тогда он пробует зашифровать короткий исходный текст, используя два различных алгоритма, потому что не уверен, какой из них является аффинным шифром подстановки. В результате он получает первый набор данных "*исходный текст/зашифрованный текст*" – $et \rightarrow WC$ и второй набор данных $et \rightarrow WF$.

Для того чтобы найти ключ, злоумышленник использует следующую стратегию:

1. Злоумышленник знает, что если первый алгоритм является аффинным шифром подстановки, он может составить следующие уравнения, основанные на первом наборе данных:

$m_1 = e \rightarrow 4;$ $\quad c_1 = (4 \times k_1 + k_2) \bmod 26 = 22 \rightarrow W;$
$m_2 = t \rightarrow 19;$ $\quad c_2 = (19 \times k_1 + k_2) \bmod 26 = 2 \rightarrow C.$

Эти два уравнения сравнения могут быть решены (могут быть найдены значения k_1 и k_2)

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 22 \\ 2 \end{pmatrix} \bmod 26 = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \cdot \begin{pmatrix} 22 \\ 2 \end{pmatrix} \bmod 26 = \begin{pmatrix} 16 \\ 10 \end{pmatrix}.$$

Откуда $k_1 = 16$, а $k_2 = 10$. Однако этот ответ неприемлем, потому что $k_1 = 16$ не может быть первой частью ключа. Его значение, *16*, не имеет мультипликативной инверсии в Z^*_{26}.

2. Злоумышленник теперь пробует использовать результат второго набора данных:

$m_1 = e \rightarrow 4;$ $\quad c_1 = (4 \times k_1 + k_2) \bmod 26 = 22 \rightarrow W;$
$m_2 = t \rightarrow 19;$ $\quad c_2 = (19 \times k_1 + k_2) \bmod 26 = 5 \rightarrow F.$

Квадратная матрица и ее инверсия – те же самые, что и в предыдущем примере:

$$\begin{pmatrix} k_1 \\ k_2 \end{pmatrix} = \begin{pmatrix} 4 & 1 \\ 19 & 1 \end{pmatrix}^{-1} \cdot \begin{pmatrix} 22 \\ 5 \end{pmatrix} \bmod 26 = \begin{pmatrix} 19 & 7 \\ 3 & 24 \end{pmatrix} \cdot \begin{pmatrix} 22 \\ 5 \end{pmatrix} \bmod 26 = \begin{pmatrix} 11 \\ 4 \end{pmatrix}.$$

Теперь злоумышленник получает $k_1 = 11$ и $k_2 = 4$, эта пара является приемлемой, потому что k_1 имеет мультипликативную инвер-

сию в Z^*_{26}. Она пробует пару ключей (*19*, *22*), которые являются инверсией пары (*11*, *4*):

$c_1 = W \rightarrow 22;$ $\quad m_1 = ((22 + 22) \times 19) \bmod 26 = 04 \rightarrow e,$
$c_2 = F \rightarrow 5;$ $\quad m_2 = ((5 + 22) \times 19) \bmod 26 = 19 \rightarrow t$

и расшифровывает сообщение

PWUFFOGWCHFDWIWEJOUUNJORSMDWRHVCMWJUPVCCG.

Исходный текст при этом будет

Best time of the year is spring when flower bloom
(*Самое лучшее время года - весна, когда цветут цветы*).

Одноалфавитный шифр подстановки

Поскольку аддитивные, мультипликативные и аффинные шифры подстановки имеют малое множество ключей, они очень уязвимы к атаке "*грубой силы*". Отправитель и получатель согласовывают единственный ключ, который они используют, чтобы зашифровать каждую букву в исходном тексте или расшифровать каждую букву в зашифрованном тексте. Другими словами, ключ независим от передаваемых букв.

Лучшее решение состоит в том, чтобы создать отображение каждой буквы исходного текста на соответствующий символ зашифрованного текста. Отправитель и получатель могут договориться об отображении для каждой буквы и записать его в виде таблицы. Рис. 2.7 показывает пример такого отображения.

Пример 2.13. Используя ключ, показанный на рис. 2.7, зашифровать сообщение

Исходный текст

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

Зашифрованный текст

Рис. 2.7. Пример ключа для одноалфавитного шифра подстановки английского алфавита

This message is easy to encrypt but hard to find the key
(*Это сообщение просто зашифровать, но трудно найти ключ, которым зашифрован текст*).

Решение. Зашифрованное сообщение имеет вид

ICFVQRVVNEFVRNVSIYRGAHSLIOJICNHTIYBFGTICRXRS.

Размер ключевого пространства для одноалфавитного шифра подстановки – число перестановок из *26*, т.е. *26!* (почти $4 \cdot 10^{26}$). Это делает атаку "грубой силы" чрезвычайно трудной для злоумышленника, даже если он использует мощный компьютер. Однако он может применить метод статистической атаки, основанный на частоте появления символов, потому что данный шифр не изменяет частоту появления символов.

2.1.2. Многоалфавитные шифры подстановки

Использование многоалфавитного шифра подстановки приводит к тому, что каждое появление символа может иметь различную замену. Отношения между символом в исходном тексте и символом в зашифрованном тексте – "*один ко многим*". Например, буква *a* может быть зашифрована как *D* в начале текста, но как *N* – в середине. Многоалфавитные шифры подстановки имеют преимущество: они скрывают частоту появления символа основного языка. Злоумышленник не может использовать статистическую частоту отдельного символа, чтобы взломать зашифрованный текст.

Чтобы создать многоалфавитный шифр подстановки, нужно сделать каждый символ зашифрованного текста зависящим от соответствующего символа исходного текста и позиции символа исходного текста в сообщении. Это подразумевает, что ключ должен быть потоком подключей, в которых каждый подключ так или иначе зависит от позиции символа исходного текста, который используется для выбора подключа шифрования. Другими словами, нужно иметь ключевой поток $k = (k_1, k_2, k_{3,}...)$, в котором k_i применяется, чтобы зашифровать *i*-ый символ в исходном тексте и создать *i*-ый символ в зашифрованном тексте.

Автоключевой шифр подстановки

Чтобы понять зависимость ключа от позиции, обсудим простой многоалфавитный шифр подстановки, названный *автоключевым*. В этом шифре ключ – поток подключей, в котором каждый подключ используется, чтобы зашифровать соответствующий символ

в исходном тексте. Первый подключ – определенное заранее значение, тайно согласованное отправителем и получателем. Второй подключ – значение первого символа исходного текста (между *0* и *25*). Третий – *i*-ое значение второго исходного текста. И так далее. Пусть исходный текст: $M = m_1, m_2, m_3, \ldots$. Зашифрованный текст: $C = c_1, c_2, c_3, \ldots$. Ключ: $K = (k_1, k_2 = m_1, k_3 = m_2, k_4 = m_3, \ldots)$.

Шифрование текстовых сообщений с помощью автоключевого шифра подстановки осуществляется с помощью выражения:

$$c_i = (m_i + k_i) \bmod n, \tag{2.7}$$

а расшифрование:

$$m_i = (c_i - k_i) \bmod n. \tag{2.8}$$

Название шифра подстановки, *автоключевой*, подразумевает, что подключи создаются автоматически в зависимости от символов шифра исходного текста в процессе шифрования.

Пример 2.14. Предположим, что отправитель и получатель согласились использовать автоключевой шифр с начальным ключевым значением $k_1 = 12$. Теперь отправитель хочет передать получателю сообщение *Attack is today* (*Атака сегодня*).

Решение. Зашифрование проводится символ за символом. Каждый символ в исходном тексте сначала заменяется его значением целого числа, как показано на рис. 2.1, первый подключ прибавляется, чтобы создать первый символ зашифрованного текста. Остальная часть ключа создается по мере чтения символов исходного текста. Результат зашифрования приведен в табл. 2.3.

Обратите внимание, что шифр является многоалфавитным, потому что эти три появления *a* в исходном тексте зашифрованы различно. Три возникновения *t* также зашифрованы различно.

Автоключевой шифр подстановки действительно скрывает статистику частоты появления отдельного символа. Однако он так же уязвим при атаке “грубой силы”, как и аддитивный шифр подстановки. Первый подключ может быть только одним из *25* значений (*1 – 25*).

Таблица 2.3

Результат зашифрования для примера 2.14

Исходный текст	*a*	*t*	*t*	*a*	*c*	*k*	*i*	*s*	*t*	*o*	*d*	*a*	*y*
Значения M	*00*	*19*	*19*	*00*	*02*	*10*	*08*	*18*	*19*	*14*	*03*	*00*	*24*
Поток ключей K	*12*	*00*	*19*	*19*	*00*	*02*	*10*	*08*	*18*	*19*	*14*	*03*	*00*
Значения C	*12*	*19*	*12*	*19*	*02*	*12*	*18*	*00*	*11*	*07*	*17*	*03*	*24*
Зашифрованный текст	*M*	*T*	*M*	*T*	*C*	*M*	*S*	*A*	*L*	*H*	*R*	*D*	*Y*

Потребность в многоалфавитных шифрах подстановки, которые не только скрывают характеристики языка, но и имеют большие множества ключей.

Шифр подстановки Плейфеера

Другой пример многоалфавитного шифра подстановки – *шифр Плейфеера*, использовавшийся британской армией в течение Первой мировой войны. Ключ засекречивания в этом шифре сделан из *25* букв алфавита, размещенных в матрице 5×5 (для латинського алфавіту, для кириличного алфавіту – в матриці 6×6).

Для создания матрицы и использование шифра достаточно запомнить ключевое слово и четыре простые правила. Чтобы составить ключевую матрицу, в первую очередь нужно заполнить пустые ячейки матрицы, буквами ключевого слова (не записывая повторяющиеся символы). Затем заполнить пустые ячейки матрицы, которые остались свободными символами алфавита, что не встречаются в ключевом слове, по порядку (в английских текстах обычно опускается символ *q*, чтобы уменьшить алфавит, в других версиях *i* и *j* объединяются в одну ячейку). Ключевое слово может быть записано в верхней строке матрицы слева направо, или по спирали с левого верхнего угла к центру. Ключевое слово, дополненное символами алфавита, составляет матрицу 5×5 и является ключом шифра. С помощью различных договоренностей о размещении букв в матрице можно создать много разных ключей засекречивания.

Для того, чтобы зашифровать сообщение необходимо разбить его на биграммы (группы из двух символов), например *hello world* становится *he ll ow or ld*, и отыскать эти биграммы в таблице. Два символа биграммы соответствуют углам прямоугольника в ключевой матрицы. Определяем положение углов этого прямоугольника относитель-

но друг друга. Затем, руководствуясь следующими четырьмя правилами, зашифровываем пары символов входного текста:

1. Если два символа биграммы совпадают, добавляем после первого символа фиктивный символ *x*, зашифровываем новую пару символов и продолжаем. В некоторых вариантах шифра *Плейфера* вместо символа *x* используется символ *q* или _.

2. Если символы биграммы входного текста встречаются в одной строке, то эти символы заменяются на символы, расположенные в ближайших столбцах справа от соответствующих символов. Если символ является последним в строке, то он заменяется на первый символ этой самой строки.

3. Если символы биграммы входного текста встречаются в одном столбце, то они превращаются в символы того же столбца, находящихся непосредственно под ними. Если символ является нижним в столбце, то он заменяется на первый символ этого самого столбца.

4. Если символы биграммы входного текста находятся в разных столбцах и разных строках, то они заменяются на символы, которые находятся в тех же строках, но соответствуют другим углам прямоугольника в ключевой матрице.

Для расшифрования зашифрованного сообщения необходимо использовать инверсию этих четырех правил, отбрасывая символы *x* (или *q*), если они не несут смысла во входном сообщении.

Шифр *Плейфеера* соответствует критериям для многоалфавитного шифра. *Ключ* – поток подключей, в котором они создаются по два одновременно. В шифре *Плейфеера* поток ключей и поток шифра – те же самые. Это означает, что вышеупомянутые правила можно представить как правила для создания потока ключей. Алгоритм зашифрования берет пару символов из исходного текста и создает пару подключей, следуя вышеуказанным правилам. Можно сказать, что поток ключей зависит от позиции символа в исходном тексте. Зависимость от позиции имеет здесь различную интерпретацию: подключ для каждого символа исходного текста зависит от следующего или предыдущего. Рассматривая шифр *Плейфеера*, таким образом, можно сказать, что зашифрованный текст – это фактически поток ключей.

Пусть исходный текст: $M = m_1, m_2, m_3, \ldots$. Зашифрованный текст: $C = c_1, c_2, c_3, \ldots$. Ключ: $K = [(k_1,k_2), (k_3,k_4), \ldots]$.

Зашифрование текстовых сообщений с помощью шифра *Плейфеера* осуществляется с помощью выражения:

$$c_i = k_i, \tag{2.9}$$

а расшифрование:

$$m_i = k_i. \tag{2.10}$$

Пример 2.15. Зашифровать исходный текст *hello world* (*Привет мир*) с помощью шифра *Плейфера* с ключевым словом *playfair example*. В таком случае матрица засекречивания (или секретный ключ) примет вид, как показано на рис. 2.8.

Секретний ключ =

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

Рис. 2.8. Пример секретного ключа шифра *Плейфера*

Решение. Когда сгруппируем по парам буквы, то получим *he ly lo wo rl d*. Для обеспечения последнего символа парой нужно добавить символ *x* за символом *y*, после чего получим *he ly lo wo rl dy*. Превращая входной текст с помощью таблицы (рис. 2.8), получим:

1. Биграмма *he* формирует прямоугольник, заменяем ее на *DM*.
2. Биграмма *ly* формирует прямоугольник, заменяем ее на *AF*.
3. Биграмма *lo* расположена в одном столбце, заменяем ее на *YK*.
4. Биграмма *wo* расположена в одном столбце, заменяем ее на *WY*.
5. Биграмма *rl* формирует прямоугольник, заменяем ее на *CR*.
6. Биграмма *dy* расположена в одном столбце, заменяем ее на *GA*.

Итак, зашифрованный текст будет представлен в виде:

DM AF YK WY CR GA.

Этот пример показывает, что шифр *Плейфера* – фактически многоалфавитный шифр: три появления символа *l* (эль) исходного

текста зашифрованы как *A, Y* и *R* соответственно; два появления символа *о* исходного текста зашифрованы как *K* и *Y* соответственно.

Для использования кириллического алфавита необходимо увеличить размер матрицы до *6×6*. Используются 32 буквы алфавита и дополнительно четыре символы: . (точка); , (запятая); – (тире); _ (знак подчеркивания). Пример матриц засекречивания для российского алфавита (без использования ключевого слова) приведен на рис. 2.9.

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

Рис. 2.9. Пример матрицы засекречивания для русского алфавита

Символ _ (знак подчеркивания) используется в случае, когда в биграммах появляются одинаковые символы (он вставляется между ними) или для обеспечения последнего символа парой.

Очевидно, атака "грубой силы" шифра *Плейфеера* очень трудна. Размер домена – *n!* (*n* факториал). Кроме того, зашифрованный текст скрывает частоту появления отдельных букв.

Однако частоты двухбуквенных комбинаций (биграмм) сохранены до некоторой степени из-за вставки наполнителя, так что криптографический аналитик может использовать атаку только на зашифрованный текст, основанную на испытании частоты биграмм, чтобы найти ключ.

Шифр подстановки Виженера

Еще одним интересным видом многоалфавитного шифра подстановки является *шифр Виженера* созданный *Блезом де Виженером*, французским математиком шестнадцатого столетия [5, 32]. Шифр *Виженера* использует различную стратегию создания потока ключей. *Поток ключей* – повторение начального потока ключа засекречивания длины *m*. Шифр может быть описан следующим образом: $(k_1, k_2, \ldots, k_n)$ – первоначальный ключ засекречивания, согласованный отправителем и получателем.

Пусть исходный текст: $M = m_1, m_2, m_3, \ldots$ Зашифрованный текст: $C = c_1, c_2, c_3, \ldots$. Поток ключей: $K = k_1, k_2, k_3, \ldots$.

Зашифрование текстовых сообщений с помощью шифра *Виженера* осуществляется с помощью выражения

$$c_i = (m_i + k_i) \bmod n, \tag{2.11}$$

а расшифрование:

$$m_i = (c_i - k_i) \bmod n. \tag{2.12}$$

Одно важное отличие между шифром подстановки *Виженера* и другими двумя многоалфавитными шифрами подстановки, которые были рассмотрены: поток ключей шифра подстановки Виженера не зависит от символов исходного текста; он зависит только от позиции символа в исходном тексте. Другими словами, поток ключей может быть создан без знания сути исходного текста.

Пример 2.16. Зашифровать сообщение *She is listening* (*Она слушает*), используя ключевое слово на *6* символов *PASCAL*.

Решение. Начальный поток ключей – это *(15, 0, 18, 2, 0, 11)*. Поток ключей – повторение этого начального потока ключей (столько раз, сколько необходимо). Процесс зашифрования представлен табл. 2.4.

Таблица 2.4

Процесс зашифрования для примера 2.16

Исходный текст	*s*	*h*	*e*	*i*	*s*	*l*	*i*	*s*	*t*	*e*	*n*	*i*	*n*	*g*
Значения M	*18*	*07*	*04*	*08*	*18*	*11*	*08*	*18*	*19*	*04*	*13*	*08*	*13*	*06*
Поток ключей	*15*	*00*	*18*	*02*	*00*	*11*	*15*	*00*	*18*	*02*	*00*	*11*	*15*	*00*
Значения C	*07*	*07*	*22*	*10*	*18*	*22*	*23*	*18*	*11*	*6*	*13*	*19*	*02*	*06*
Зашифрованный текст	*H*	*H*	*W*	*K*	*S*	*W*	*X*	*S*	*L*	*G*	*N*	*T*	*C*	*G*

Таким образом, зашифрованный текст будет представлен в виде: *HHWKSWXSLGNTCG*.

Шифр подстановки *Виженера* может рассматриваться как комбинации аддитивных шифров. Рис. 2.10 показывает, что исходный текст предыдущего примера можно рассматривать как состоящий из нескольких частей по шесть элементов в каждом (хотя в одном не хватило букв исходного текста), где каждый из элементов зашифрован отдельно.

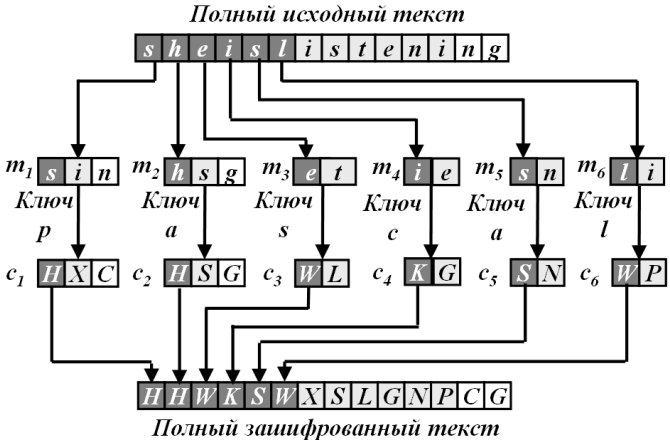


Рис. 2.10. Шифр подстановки *Виженера* как комбинация аддитивных шифров

Рисунок поможет позже понять криптографический анализ шифра подстановки *Виженера*. Имеется *n* частей исходного текста, каждая из которых зашифрована различным ключом, чтобы разделить зашифрованный текст на *m* частей.

Аддитивный шифр – частный случай шифра подстановки Виженера, в котором *m = 1*.

Другой способ рассмотрения подстановки *Виженера* – с помощью того, что названо *списком или таблицей Виженера* (*Vigenere tableau*) и показано в табл. 2.5.

Первая строка показывает символы исходного текста, который будет зашифрован. Первая колонка содержит столбец символов, которые используются ключом. Остальная часть таблицы показывает символы зашифрованного текста. Чтобы найти зашифрованный текст для исходного *she listening*, используя слово *PASCAL* как ключ, можно найти символ *s* из исходного текста в первой строке, символ ключевого слова *P* в первом столбце и на пересечении строки и столбца – символ из зашифрованного текста *H*. Находим *h* в первой строке и *A* в первом столбце, на пересечении строки и столбца – символ *H* из зашифрованного текста. И повторяем те же действия, пока все символы зашифрованного текста не будут найдены.

Таблица 2.5

Список *Виженера*

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

Чтобы найти исходный текст для зашифрованного *HHWKSWXSLGNTCG*, используя слово *PASCAL* как ключ, нужно найти символ *P* из ключевого слова в первом столбце; далее, двигаясь вправо по строке, находим символ *H* зашифрованного текста; на пересечении столбца, в котором находится символ *H* и первой строки – символ исходного текста *s*. Находим символ *A* из ключевого слова в первом столбце; далее, двигаясь вправо по строке находим символ *H* зашифрованного текста; на пересечении столбца, в котором находится символ *H* и первой строки – символ исходного тек-

ста *h*. Повторяем те же действия, пока все символы исходногоо текста не будут найдены.

Шифр подстановки *Виженера*, подобно всем многоалфавитным шифрам, не сохраняет частоту появления символов. Однако злоумышленник может использовать некоторые методы для того, чтобы расшифровать перехваченный зашифрованный текст. Криптографический анализ в данном случае состоит из двух частей: находят длину ключа и потом непосредственно находят ключ.

1. Были найдены несколько методов, чтобы найти длину ключа. Один метод рассмотрим ниже. В так называемом *тесте Касиского (Kasiski)* криптографический аналитик в зашифрованном тексте ищет повторные сегменты по крайней мере из трех символов [20, 32]. Предположим, что найдены два сегмента, и расстояние между ними – *d*. Криптографический аналитик предполагает, что $d|m$, где *m* – длина ключа. Если можно найти больше повторных сегментов с расстоянием $d_1, d_2, \ldots, d_n$, тогда $gcd\,(d_1, d_2, \ldots, d_m\ldots)/m$ (обозначение *gcd* для наибольшого общего делителя происходит от английских слов *greatest common divisor – наибольший общий делитель* – и принято в современной литературе). Это предположение логично, потому что если два символа одинаковы и $k \times m$ ($k = 1, 2, \ldots$) – символы, выделенные в исходном тексте, то одинаковы и $k \times m$ символы, выделенные в зашифрованном тексте. Криптографический аналитик использует сегменты по крайней мере из трех символов, чтобы избежать случаев, где символы имеют один и тот же ключ.

2. После того как длина ключа была найдена, криптографический аналитик использует аддитивный шифр – частный случай шифру подстановки *Виженера*, в котором $m = 1$. Зашифрованный текст делится на *m* различных частей и применяется метод, используемый в криптографическом анализе аддитивного шифра, включая статистическую атаку частоты появления символов. Каждая *m* часть зашифрованного текста может быть расшифрована и соединена с другими *m*, чтобы создать целый исходный текст. Весь зашифрованный текст не сохраняет частоты появления отдельных букв исходного текста, но каждая часть делает это.

Пример 2.17 может помочь понять эти рассуждения.

Пример 2.17. Предположим, что злоумышленник перехватил следующий зашифрованный текст:

LIOMWGFEGGDVWGHHCQUCRHRWAGWIOWQLKGZETK – KMEVLWPCZVG'TH VTSGXQOVGCSVETQLTJSUMVWVEU – VLXEWSLGFZMVVWLGYHCUSWXQHKVGSHEEVFLCFDGV – SUMPHKIRZDMPHHBVWVWJWIXGFWLTSHGJOUEEHHVUC – FVGOWICQLIJSUXGLW.

Злоумышленнику необходимо дешифровать этот зашифрованный текст.

Решение. Тест *Касиского* на повторение сегментов на три символа приводит к результатам, показанным в табл. 2.6.

Таблица 2.6

Тест Касиского для примера 2.17

Комбинация	*Первое расстояние*	*Второе расстояние*	*Разность*
JSU	*68*	*168*	*100*
SUM	*69*	*117*	*48*
VWV	*72*	*132*	*60*
MPH	*119*	*127*	*8*

Наибольший общий делитель – *4*, что означает длину ключа, пропорциональную четырем. Сначала злоумышленник пробует $m = 4$. Делит зашифрованный текст на четыре части. Часть c_1 будет состоять из символов *1, 5, 9, ...* ; часть c_2 будет состоять из символов *2, 6, 10, ...* и так далее. Злоумышленник использует статистическую атаку каждой части отдельно. Он перебирает расшифровывающиеся части по одному символу одновременно, чтобы получить целый исходный текст как показано в табл. 2.7.

Если исходный текст не имеет смысла, злоумышленник пробуем с другим *m*. В рассматриваемом случае исходный текст имеет смысл (читаем по столбцам):

Julius Caesar used a cryptosystem in his wars, which is now referred to as Caesar chipper. It is an additive chipper with the key set to three. Each character in the plaintext is shift three characters to create ciphertext.

Перевод этого текста:

Юлий Цезарь использовал в своих войнах криптографическую систему, которая упоминается теперь как шифр Цезаря. Это аддитив-

ный шифр с ключом, установленным на три. Каждый символ в исходном тексте сдвинут на три символа, чтобы создать зашифрованный текст.

Таблица 2.7

Процес дешифрования зашифрованного текста
с помощью теста Касиского для примера 2.17

c_1	*LWGW*	*CRAO*	*KTEP*	*GTQC*	*TJVU*	*EGVG*
m_1	*jueu*	*apym*	*ircn*	*eroa*	*rhts*	*thin*
c_1	*UQGE*	*CVPR*	*PVJG*	*TJEU*	*GCJG*	
m_1	*ytra*	*hcie*	*ixst*	*hcar*	*rehe*	
c_2	*IGGG*	*QHGW*	*GKVC*	*TSOS*	*QSWV*	*WFVY*
m_2	*usss*	*ctsl*	*swho*	*feae*	*ceih*	*cete*
c_2	*SHSV*	*FSHZ*	*HWWF*	*SOHC*	*OQSL*	
m_2	*soec*	*atnp*	*nkhe*	*rhck*	*esex*	
c_3	*OFDN*	*URWQ*	*ZKLZ*	*HGVV*	*LUVL*	*SZWH*
m_3	*lcae*	*rotn*	*whiw*	*edss*	*irsi*	*irh*
c_3	*WKHF*	*DUKD*	*HVIW*	*HUHF*	*WLUW*	
m_3	*eteh*	*retl*	*tiid*	*eatr*	*airt*	
c_4	*MEVN*	*CWIL*	*EMWV*	*VXGE*	*TMEX*	*LMLC*
m_4	*iard*	*yseh*	*aisr*	*rtca*	*piaf*	*pwte*
c_4	*XVEL*	*GMIM*	*BWHL*	*GEVV*	*ITX*	
m_4	*thec*	*arha*	*esft*	*erec*	*tpt*	

Для использования зашифрования и расшифрования сообщений с использованием русского алфавита можно воспользоваться табл. 2.8.

Шифр Хилла

Другой интересный пример многоалфавитного шифра – *шифр Хилла*, изобретенный *Лестером С. Хиллом* [2, 32]. В отличие от других многоалфавитных шифров, которые были уже рассмотрены, здесь исходный текст разделен на блоки равного размера. Блоки зашифрованы по одному таким способом, что каждый символ в блоке вносит вклад в шифрование других символов в блоке. По этой причине шифр *Хилла* принадлежит к категории шифров, названных *блочными шифрами*.

Таблица 2.8

Список Виженера для русского алфавита																															
	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
А	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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	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

Окончание табл. 2.8

Список Виженера для русского алфавита																																
	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*
Р	*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	*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	*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	*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	*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	*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	*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	*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	*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	*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	*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	*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	*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	*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	*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	*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*

Другие шифры, которые были изучены до сих пор, принадлежат к категории, называемых *шифрами потока*. Отличие между шифрами блока и шифрами потока будут обсуждены в разд. 3 и 4.

В шифре *Хилла* ключ – квадратная матрица размера $n \times n$, в котором n является размером блока. Если вызывается ключевая матрица K, то каждый элемент $k_{i,j}$ определяется матрицей, как показано на рис. 2.11.

$$\begin{pmatrix} k_{11} & k_{12} & \dots & k_{1g} \\ k_{21} & k_{22} & \dots & k_{2g} \\ \dots & \dots & \dots & \dots \\ k_{g1} & k_{g2} & \dots & k_{gg} \end{pmatrix}$$

Рис. 2.11. Ключ в шифре Хилла

Покажем, как получается один блок зашифрованного текста. Если обозначим n символов блоков исходного текста $m_1, m_2, ..., m_g$, соответствующие символы в блоках зашифрованного текста будут $c_1, c_2, ..., c_g$. Тогда будем иметь:

$$c_1 = m_1 \cdot k_{11} + m_2 \cdot k_{21} + \dots + m_g \cdot k_{g1};$$

$$c_2 = m_1 \cdot k_{12} + m_2 \cdot k_{22} + \dots + m_g \cdot k_{g2};$$

$$\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots\dots$$

$$c_g = m_1 \cdot k_{1g} + m_2 \cdot k_{2g} + \dots + m_g \cdot k_{gg}.$$

Уравнения показывают, что каждый символ зашифрованного текста, такой, как c_1, зависит от символов всего исходного текста в блоке ($m_1, m_2, ..., m_g$). Однако необходимо знать, что не все квадратные матрицы имеют мультипликативные инверсии в Z_g, так что отправитель и получатель должны быть осторожны в выборе ключа. Получатель не сможет расшифровать зашифрованный текст, передаваемый отправителем, если матрица не имеет мультипликативной инверсии. *Ключевая матрица в шифре Хилла должна иметь мультипликативную инверсию.*

Уравнения показывают, что каждый символ зашифрованного текста, такой, как c_1, зависит от символов всего исходного текста в блоке ($m_1, m_2, ..., m_g$). Однако необходимо знать, что не все квадратные матрицы имеют мультипликативные инверсии в Z_g, так что отправитель и получатель должны быть осторожны в выборе ключа.

Получатель не сможет расшифровать зашифрованный текст, передаваемый отправителем, если матрица не имеет мультипликативной инверсии. *Ключевая матрица в шифре Хилла должна иметь мультипликативную инверсию.*

Использование матриц позволяет отправителю зашифровать весь исходный текст. В этом случае исходный текст – $l \times g$ – матрица, в которой l является номером блоков. Зашифрование текстовых сообщений с помощью шифра *Хилла* осуществляется по правилу:

$$C(l \times g) = M(l \times g) \times K(g \times g) \bmod n, \qquad (2.13)$$

а расшифрование:

$$M(l \times g) = C(l \times g) \times K^{-1}(g \times g) \bmod n. \qquad (2.14)$$

Пример 2.18. Пусть исходный текст *code is ready* (*код готов*) необходимо зашифровать, а затем расшифровать зашифрованный текст с помощью шифра *Хилла*, используя ключевую матрицу K (*GYBNQKURP* в буквеном виде):

$$K = \begin{pmatrix} 06 & 24 & 01 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}.$$

Решение. Исходный текст может быть представлен как матрица размером 4×3 при добавлении дополнительного фиктивного символа *z* к последнему блоку и удалении пробелов. Исходный текст для шифрования в таком случае будет: *codeisreadyz*. Числовой эквивалент данного сообщения представляется в виде: *02, 14, 03, 04, 08, 18, 17, 00, 03, 24, 25*.

Зашифрование сообщения с помощью шифра *Хилла* показано на рис. 2.12.

$$\underset{C}{\begin{pmatrix} 20 & 11 & 05 \\ 20 & 10 & 16 \\ 24 & 04 & 05 \\ 24 & 23 & 20 \end{pmatrix}} = \underset{M}{\begin{pmatrix} 02 & 14 & 03 \\ 04 & 08 & 18 \\ 17 & 04 & 00 \\ 03 & 24 & 25 \end{pmatrix}} \times \underset{K}{\begin{pmatrix} 06 & 24 & 01 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix}}.$$

Рис. 2.12. Зашифрование сообщений с помощью шифра *Хилла* для примера 2.18

Как видим, в результате зашифрования получено зашифрованное сообщение: *20, 11, 05, 20, 10, 16, 24, 04, 05, 24, 23, 20*. Это зашифрованное сообщение соответствует *ULFUKQYEFYXU* с использованием английского алфавита.

Получатель может расшифровать сообщение, используя мультипликативно инверсную (обратную) матрицу-ключ K^{-1}, такую, что $(K \times K^{-1}) \bmod n = I$, где I – единичная матрица. Для нашего примера умножения матрицы-ключа K на мультипликативно обратную матрицу-ключ K^{-1} будет равна:

$$K \times K^{-1} = \begin{pmatrix} 06 & 24 & 01 \\ 13 & 16 & 10 \\ 20 & 17 & 15 \end{pmatrix} \times \begin{pmatrix} 08 & 05 & 10 \\ 21 & 08 & 21 \\ 21 & 12 & 08 \end{pmatrix} = \begin{pmatrix} 01 & 00 & 00 \\ 00 & 01 & 00 \\ 00 & 00 & 01 \end{pmatrix} = I.$$

Расшифрование зашифрованного сообщения в примере 2.18 с помощью шифра *Хилла* показано на рис. 2.13.

$$\underset{M}{\begin{pmatrix} 02 & 14 & 03 \\ 04 & 08 & 18 \\ 17 & 04 & 00 \\ 03 & 24 & 25 \end{pmatrix}} = \underset{C}{\begin{pmatrix} 20 & 11 & 05 \\ 20 & 10 & 16 \\ 24 & 04 & 05 \\ 24 & 23 & 20 \end{pmatrix}} \times \underset{K^{-1}}{\begin{pmatrix} 08 & 05 & 10 \\ 21 & 08 & 21 \\ 21 & 12 & 08 \end{pmatrix}}$$

Рис. 2.13. Расшифрование зашифрованого сообщения с помощью шифра *Хилла* для примера 2.18

Из рис. 2.13 следует, что в результате расшифрования получено исходное сообщение: *02, 14, 03, 04, 08, 18, 17, 04, 00, 03, 24, 25*, которое соответствует *codeisreadyz* с использованием английского алфавита. Отбрасывая последний символ, как результат получаем с*odeisready*, что соответствует входному тексту.

Криптографический анализ только для зашифрованного шифрами *Хилла* текста труден. Во-первых, атака "грубой силы" при использовании шифра *Хилла* чрезвычайно сложна, потому что матрица-ключ – $g \times g$. Каждый вход может иметь одно из n значений. Это означает что размер ключа $n^{g \times g}$. Однако, не все матрицы имеют мультипликативно обратную матрицу-ключ. Поэтому область существования ключей все же не такая огромная.

Во-вторых, шифр *Хилла* не сохраняют статистику обычного текста. Злоумышленник не может провести статистическую атаку частоты появления отдельных букв из двух или трех букв. Анализ частоты слов размера g мог бы сработать, но очень редко исходный текст имеет много одинаковых строк размера g. Злоумышленник, однако, может провести атаку на шифр, используя метод знания

исходного текста, если он знает значение g и знает пары *исходный текст/зашифрованный текст*, по крайней мере m блоков. Блоки могут принадлежать тому же самому сообщению или различным сообщениям, но должны быть различны. Злоумышленник может создать две $g \times g$ матрицы, M (*исходный текст*) и C (*зашифрованный текст*), в котором соответствующие строки представляют известные пары *исходный текст/зашифрованный текст*. Поскольку $C = M \times K$, злоумышленник может использовать отношения $K = M^{-1} \times C$, чтобы найти ключ, если M является обратимою. Если M не является обратимою, то злоумышленник должен задействовать различные наборы g пар *исходный текст/зашифрованный текст*.

Если злоумышленник не знает значение g, он может попробовать различные значения при условии, что m не является очень большим.

Пример 2.19. Предположим, что злоумышленник знает, что $g = 3$. Он перехватил три пары блока *исходный текст/зашифрованный текст* (не обязательно из того же самого сообщения), как показано на рис. 2.14.

$$\begin{array}{cccc} (05 & 07 & 10) & \longleftrightarrow & (03 & 06 & 00) \\ (13 & 17 & 07) & \longleftrightarrow & (14 & 16 & 19) \\ (00 & 05 & 04) & \longleftrightarrow & (03 & 17 & 11) \\ \multicolumn{3}{c}{M} & & \multicolumn{3}{c}{C} \end{array}$$

Рис. 2.14. Формирования зашифрованного текста с помощью шифра *Хилла* для примера 2.19

Решение. Злоумышленник составляет матрицы M и C из пар, представленных на рис. 2.12. Поскольку в данном случае матрица M обратима, он инвертирует эту матрицу и умножает ее на C, что дает матрицу ключей K, как это показано на рис. 2.15.

Теперь он имеет ключ и может взломать любой шифрованный текст, где применен этот ключ.

$$\underset{K}{\begin{pmatrix} 02 & 03 & 07 \\ 05 & 07 & 09 \\ 01 & 02 & 11 \end{pmatrix}} = \underset{M^{-1}}{\begin{pmatrix} 21 & 14 & 01 \\ 00 & 08 & 25 \\ 13 & 03 & 08 \end{pmatrix}} \times \underset{C}{\begin{pmatrix} 03 & 06 & 00 \\ 14 & 16 & 09 \\ 03 & 17 & 11 \end{pmatrix}}$$

Рис. 2.15. Получение матрицы-ключа K шифра *Хилла* для примера 2.19

Одноразовая система шифрования

Одна из целей криптографии – идеальная секретность. Почти все применяемые на практике шифры характеризуются как *условно надежные*, поскольку они могут быть раскрыты при наличии неограниченных вычислительных возможностей. *Абсолютно надежные шифры* нельзя разрушить даже при использовании неограниченных вычислительных возможностей.

Существует единственный такой шифр, изобретенный в 1917 г. американцами *Г. Вернамом* и *Д. Моборном* [3, 9] и используемый на практике, – *одноразовый блокнот*. Характерной особенностью такого шифра является однократное использование ключевой последовательности. В литературе эту систему шифрования часто называют *шифром Вернама*. Ключ данного шифра имеет такую же длину, что и входной текст, и избирается совершенно случайно.

В 1949 г. было опубликовано работу *К. Шеннона*, в которой доказано абсолютная стойкость шифра *Вернама*. Работа *Шеннона* показывает, что не существует других шифров с подобными свойствами. Это и привело к выводу с таким утверждения: шифр *Вернама* – самая безопасная криптографическая система из всех имеющихся.

Исследования *К. Шеннона* показали, что идеальная секретность может быть достигнута при выполнение следующих правил:

- ключ для шифрования выбирается случайно;
- длина ключа должна равняться длине шифруемого текста;
- ключ должен использоваться только один раз.

Известно, что аддитивный шифр можно легко взломать, если использовать один и тот же ключ. Но даже и этот шифр может быть идеальным если для шифрования каждого символа применять ключ, который выбирается случайно из множества ключей (*00, 01, 02, ..., n-1*): если первый символ, зашифрованный с помощью ключа *04,* второй символ – ключа *02*, третий – ключа *21* и так далее. Тогда атака только на зашифрованный текст становится невозможной. В случае если отправитель меняет ключ, используя каждый раз другую случайную последовательность целых чисел, то и другие типы атак также будут невозможны.

Для реализации такой системы подстановок иногда используют *одноразовый блокнот*. Этот блокнот состоит из отрывных листов, на каждом из которых напечатана таблица со случайными числами

(ключами) k_i. Блокнот выполняется в двух экземплярах: один используется отправителем, а другой – получателем. Для каждого символа m_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 нового листа.

Этот шифр является абсолютно надежным, если набор ключей K действительно случайный и непредсказуемый. Даже, если криптографический аналитик попытается использовать все возможные наборы ключей заданных для зашифрованных данных и восстановит варианты входных данных, то они окажутся равновероятными. То есть не имеет возможности выбрать входные данные, которые были посланы.

Казалось бы, что благодаря такому преимуществу одноразовые системы следует применять во всех случаях, которые требуют абсолютной информационной безопасности. Однако возможности применения одноразовой системы ограничены чисто практическими аспектами. Существенной особенностью есть требование одноразового использования случайной ключевой последовательности. Ключевая последовательность с длиной, не меньшей длины сообщения, должна передаваться получателю сообщения заранее или отдельно по некоторому закрытому каналу. Это требование не является слишком обременительным для передачи действительно важных одноразовых сообщений. Однако такое требование почти невыполнимо для современных систем обработки информации, где требуется шифровать много миллионов 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 известного отправителю и получателю сообщения. Ключевая последовательность начинается с указанного места этой книги и используется так же, как в системе *Виженера*. Иногда такой шифр называют *шифром с бегущим ключом*. Управление ключевой последовательностью в таком варианте шифра намного проще, поскольку длинная ключевая последовательность может быть представлена в компактной форме. Но с другой стороны, эти ключи не будут случайными. Поэтому, у криптографического аналитика появляется возможность исполь-

зовать информацию о частоте символов входного естественного языка и реализовать соответствующую атаку.

Система шифрования *Вернама* является, по сути, частный случаем системы шифрования *Виженера* со значением модуля $n = 2$. Конкретная версия этого шифра, предложенная в 1926 г. *Г. Вернамом* сотрудником фирмы AT&T (США), использует двоичное представление символов входных данных.

Каждый символ входного открытого текста из английского алфавита $\{A, B, C, D, \ldots, Z\}$, расширенного шестью вспомогательными символами (пробел, возврат каретки и т.д.), сначала кодировался в пятибитовый блок (b_0, b_1, b_2, b_3, b_4) телеграфного *кода Бодо*.

Случайная последовательность двоичных пятибитовых ключей $\{ k_0, k_1, k_2, \ldots \}$ предварительно записывалась на бумажной ленте.

Схему передачи сообщения с использованием шифрования методом *Вернама* показано на рис. 2.16.

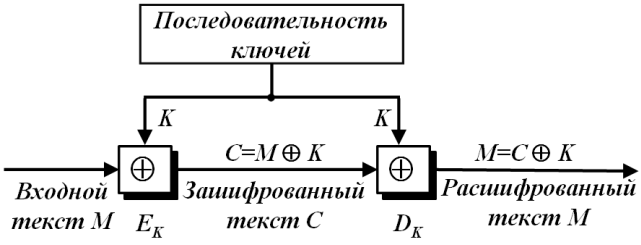


Рис. 2.16. Схема зашифрования и расшифрования сообщений методом *Вернама*

Зашифрования входных данных, предварительно преобразованных в последовательность двоичных символов n, осуществляется путем сложения по модулю 2 символов n с последовательностью ключей k.

Символы зашифрованных данных преобразуется следующим образом:

$$c_i = m_i \oplus k_i,\ i = 1, 2, 3, \ldots, n. \qquad (2.15)$$

Расшифрование заключается в сложении по модулю 2 символов зашифрованных данных c_i с той же последовательностью ключей k_i:

$$m_i = c_i \oplus k_i = m_i \oplus k_i \oplus k_i = m_i,\ i = 1, 2, 3, \ldots, n. \qquad (2.16)$$

При этом последовательности ключей, используемые во время зашифрования и расшифрования, компенсируют друг друга (при сложении по модулю *2*), и вследствие этого восстанавливаются символы входных данных m_i.

Разрабатывая свою систему *Вернам* проверял ее с помощью закольцованных лент, установленных на передающей и приемной сторонах для того, чтобы использовалась та же последовательность ключей (рис. 2.17).

В реальных системах изначально готовят две одинаковые ленты со случайными цифрами ключа. Одна остается у отправителя, а другая – передается *неперехватываемым* способом, например, курьером с охраной, законному получателю. Когда отправитель хочет передать сообщение, он сначала превращает его в двоичную форму и помещает в устройство, к каждой цифре сообщения добавляет по модулю *2* цифры, определенные в ключевой ленте.

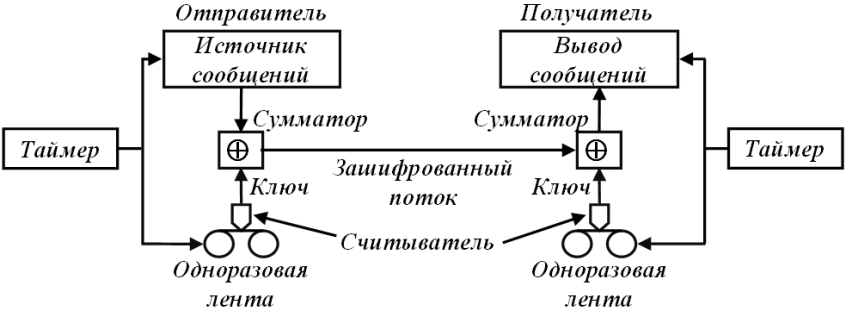


Рис. 2.17. Система шифрования *Вернама*, которая использует закольцованные ленты

На принимающей стороне зашифрованное сообщение записывается и пропускается через машину, похожую на устройство, которое используется для зашифрования. Это устройство к каждой двоичной цифре сообщения добавляет (отнимает, поскольку сложение и вычитание по модулю *2* эквивалентны) по модулю *2* цифры, определенные в ключевой ленте, получая, таким образом, открытый текст. При этом ключевая лента должна продвигаться абсолютно синхронно со своим дубликатом, что используется для зашифрования.

Главным недостатком этой системы является то, что для каждого бита переданной информации должен быть заранее подготовлен бит ключевой информации, причем эти биты должны быть случай-

ными. Зашифрование большого объема данных является серьезным ограничением. Поэтому данная система используется только для передачи сообщений наивысшей секретности.

Чтобы обойти проблему предыдущей передачи секретного ключа большого объема, инженеры и изобретатели придумали много схем генерации очень длинных потоков псевдослучайных цифр из нескольких коротких потоков согласно некоторому алгоритму. Получателя зашифрованного сообщения при этом необходимо обеспечить точно таким же генератором, как и у отправителя. Но такие алгоритмы добавляют регулярность в зашифрованный текст, выявление которых может помочь криптографическому аналитику дешифровать сообщение. Один из основных методов построения подобных генераторов заключается в использовании двух или более битных лент, в которых определеные данные побитно складываются для получения *смешанного* потока (рис. 2.18).

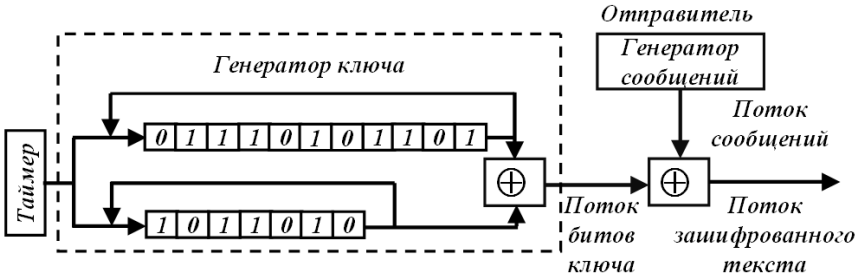


Рис. 2.18. Система шифрования *Вернама*, которая использует смешанный поток

Следует отметить, что метод *Вернама* не зависит от длины последовательности ключей и, кроме того, позволяет использовать случайную последовательность ключей. Однако при реализации метода *Вернама* возникают серьезные проблемы, связанные с необходимостью доставки получателю той самой последовательности ключей как у отправителя, или с необходимостью безопасного хранения идентичных последовательностей ключей у отправителя и у получателя. Эти недостатки системы шифрования *Вернама* устраняются при использовании шифрования методом гаммирования.

Шифрование методом гаммирования

Суть *шифрования методом гаммирования* заключается в том, что символы данных, которые шифруются, последовательно складываются с символами некоторой специальной последовательности, которая называется *гаммой*. Иногда такой метод представляют как наложение гаммы на входные данные, отсюда и название "*гаммирование*".

Процедуру наложения гаммы на входные данные можно осуществить двумя способами.

В первом способе символы входного текста и гаммы заменяются цифровыми эквивалентами, которые затем складываются по модулю n, где n – число символов в алфавите, то есть:

$$c_i = (m_i + g_i) \bmod n, \quad i = 1, 2, 3, \ldots, n, \tag{2.17}$$

где c_i, m_i, g_i – символы соответственно зашифрованного текста, входящего текста и гаммы.

Пример 2.20. Пусть открытый текст *Гамбит*, цифровой эквивалент которого в соответствии с табл. 2.8 отвечает: *03 00 12 01 08 18*. Гамма: *Модель - 12 14 04 05 11 28*. Для русского алфавита $n = 32$.

Решение: Для зашифрования открытого текста воспользуемся соотношением (2.17) и получим:

$$c_1 = (03 + 12) \bmod 32 = 15; \quad c_2 = (00 + 14) \bmod 32 = 14;$$

$$c_3 = (12 + 04) \bmod 32 = 16; \quad c_4 = (01 + 05) \bmod 32 = 06;$$

$$c_5 = (08 + 11) \bmod 32 = 19; \quad c_6 = (18 + 28) \bmod 32 = 14.$$

Итак, вышел зашифрованный текст: *15 14 16 06 19 14*, который с учетом рис. 2.2 представляется в виде: *Поржуо*.

При использовании второго способа символы входного текста и гаммы представляются в виде двоичного кода, а затем соответствующие разряды складаются по модулю *2*:

$$c_{ij} = m_{ij} \oplus g_{ij}, \quad i = 1, 2, 3, \ldots, n; \quad j = 0, 1, \ldots, l, \tag{2.18}$$

где $\oplus$ – операция суммирования по модулю *2*; c_{ij}, m_{ij} и g_{ij} – двоичные символы зашифрованного текста, входного текста и гаммы

соответственно; n – количество символов данных, которые шифруются (количество символов зашифрованных данных; количество символов гаммы); l – количество двоичных битов входных символов, гаммы и зашифрованных символов.

Пример 2.21. Пусть открытый текст и гамма как в вышеприведенном примере.

Решение: Для зашифрования воспользуемся соотношением (2.18):

	03	*00*	*12*	*01*	*08*	*18*
	00011	*00000*	*01100*	*00001*	*01000*	*10010*
⊕						
	01100	*01110*	*00100*	*00101*	*01011*	*11100*
	01111	*01110*	*01000*	*00100*	*00011*	*01110*
	15	*14*	*08*	*04*	*03*	*14*

Имеем зашифрованное сообщение: *15 14 08 03 04 14*, которое соответствует *Поидго*.

Вместо сложения по модулю *2* при гаммирования можно использовать другие логические операции, например, по правилам логической эквивалентности или логической неэквивалентности. Такая замена равнозначна введению еще одного ключа, которым является выбор правила формирования символов зашифрованного сообщения из символов входного текста и гаммы.

Стойкость шифрования методом гаммирования определяется, главным образом, свойствами гаммы – продолжительностью периода и равномерностью статистических характеристик. Последнее свойство обеспечивает отсутствие закономерностей при появлении различных символов в пределах периода.

Расшифрование осуществляется путем применения к символам зашифрованного текста и гаммы обратной операции:

$$c_i = (m_i - g_i) \bmod n, \quad i = 1, 2, 3, \ldots, l.$$

или

$$c_{ij} = m_{ij} \oplus g_{ij}, \quad i = 1, 2, 3, \ldots, n; \quad j = 0, 1, \ldots, l,$$

Пример 2.22. Расшифруем зашифрованный текст из предыдущего примера (*Пример 2.21*), зашифрованного двумя способами.

Решение: Для первого способа:

$$m_1 = (15 - 12) \bmod 32 = 03; \quad m_2 = (14 - 14) \bmod 32 = 00;$$

$m_3 = (16 - 04)\ mod\ 32 = 12$; $m_4 = (06 - 05)\ mod\ 32 = 01$;

$m_5 = (19 - 11)\ mod\ 32 = 08$; $m_6 = (14 - 28)\ mod\ 32 = 18$.

Итак, получился открытый текст: *03 00 12 01 08 18*, который соответствует сообщению *Гамбит*.

Для второго способа:

	15	*14*	*08*	*04*	*03*	*14*
	01111	*01110*	*01000*	*00100*	*00011*	*01110*
⊕	*01100*	*01110*	*00100*	*00101*	*01011*	*11100*
	00011	*00000*	*01100*	*00001*	*01000*	*10010*
	03	*00*	*12*	*01*	*08*	*18*

Как видим, получился открытый текст: *03 00 12 01 08 18*, который также отвечает открытому тексту *Гамбит*.

Стойкость систем шифрования, основанных на гаммировании, зависит от характеристик гаммы – ее длины и равномерности распределения вероятности появления знаков гаммы.

Разделяют две разновидности шифрования методои гаммирования – с конечной и бесконечной гаммой. При лучших статистических свойствах гаммы стойкость шифрования определяется только длиной периода. При этом если длина периода гаммы превышает длину зашифрованного текста, то такой шифр теоретически является *абсолютно стойким*. Однако, это не означает, что дешифрованиие текста вообще невозможно: при наличии некоторой дополнительной информации входной текст может быть частично или полностью восстановлен даже за использование бесконечной гаммы.

В качестве бесконечной гаммы может быть использована любая последовательность случайных символов, например, последовательность цифр числа π или e. При шифровании электронно-вычислительной машиной последовательность гаммы формируется с помощью датчика псевдослучайных чисел. В настоящее время разработаны алгоритмы работы таких датчиков, которые обеспечивают удовлетворительные характеристики [32].

Роторный шифр подстановки

Хотя шифры одноразового блокнота не применяются на практике, один шаг от него к более защищенному шифру – *роторный*

шифр подстановки. Он возвращается к идее моноалфавитной подстановки, но изменяет принцип отображения исходного текста в символы зашифрованного текста для каждого символа исходного текста. Рис. 2.19 показывает упрощенный пример роторного шифра.

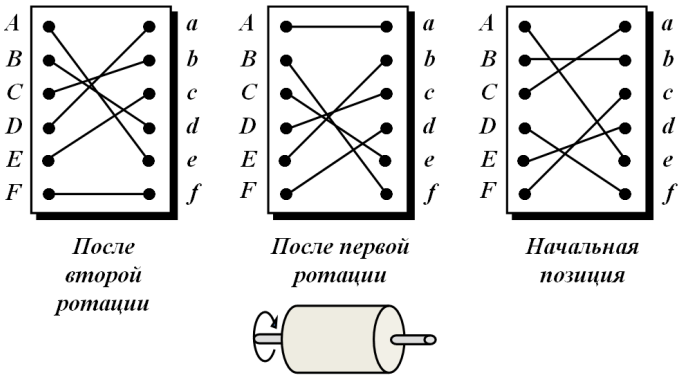


Рис. 2.19. Роторный шифр подстановки

Ротор, показанный на рис. 2.19, применен только для *6* букв, но реальные роторы используют *26* букв английского алфавита. Ротор постоянно связывает символы исходного и зашифрованного текстов, но подключение обеспечивается щетками. Обратите внимание, что соединение символов исходного и зашифрованного текстов показано так, как если бы ротор был прозрачен и можно было видеть внутреннюю часть.

Начальная установка (позиция) ротора – ключ засекречивания между отправителем и получателем – это зашифрованный первый символ исходного текста. Используя начальную установку, второй символ зашифрован после того, как проведено первое вращение (на рис. 2.19 – это поворот на *1/6* круга, на реальной установке – поворот на *1/26*), и так далее.

Слово с тремя буквами, такими как *bee*, зашифровано как *BAA,* если ротор неподвижен (моноалфавитный шифр подстановки), но оно будет зашифровано как *BCA*, если он вращается (роторный шифр подстановки). Это показывает, что роторный шифр подстановки – многоалфавитный шифр, потому что два появления того же самого символа исходного текста зашифрованы как различные символы.

Роторный шифр подстановки является стойким к атаке “грубой силы”, как моноалфавитный шифр подстановки, потому что зло-

умышленник должен найти первое множество отображений среди возможных *n!* (*n* факториал). Роторный шифр подстановки является намного более стойким к статистической атаке, чем моноалфавитный шифр подстановки, потому что в нем не сохраняется частота употребления букв алфавита.

Роторную машину подстановки "*Энигма*" первоначально была изобретена в Сербии, но специалисты немецкой армии изменили ее и интенсивно использовали в течение Второй мировой войны [2, 32]. Машина базировалась на принципе роторных шифров. Рис. 2.20 показывает упрощенную схему построения машины подстановки "*Энигма*".

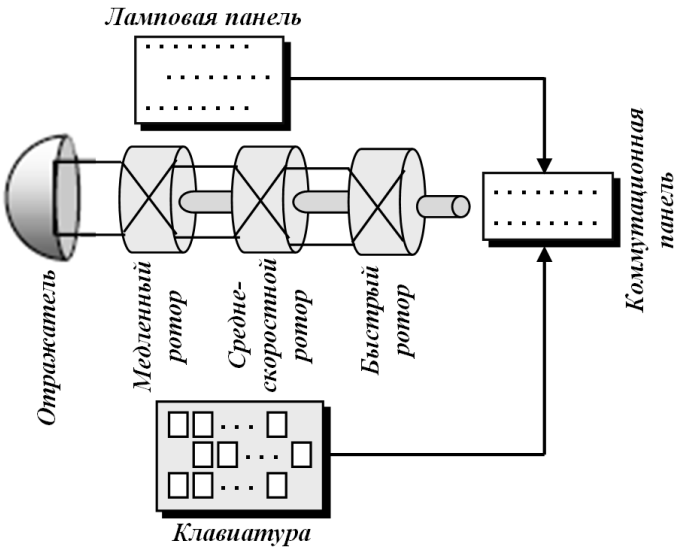


Рис. 2.20. Примерное построение машины подстановки "Энгима"

Ниже перечислены главные компоненты машины:

1. Клавиатура с *26*-ю ключами, используемыми для того, чтобы вводить исходный текст при зашифровании, и для того, чтобы вводить зашифрованный текст при расшифровании.

2. Ламповая панель с *26*-ю лампами, которая показывает символы зашифрованного текста при зашифровании и символы исходного текста при расшифровании.

3. Коммутационная панель с *26*-ю штепселями, вручную подключенными тринадцатью проводами. Конфигурация изменяется каждый день, чтобы обеспечить различное скремблирование.

4. Три замонтированных ротора, такие же, как рассмотренные в предыдущей секции. Эти три ротора выбираются ежедневно из пяти доступных роторов. Быстрый ротор вращается на *1/26* поворота при каждом символе, введенном с помощью клавиатуры. Средний ротор делает *1/26* поворота при каждом полном повороте быстрого ротора. Медленный ротор делает *1/26* поворота для каждого законченного поворота среднего ротора.

5. Отражатель, который является постоянным и предварительно замонтированным.

Чтобы использовать машину подстановки "Энгима", была издана кодовая книга, которая в течение каждого дня дает несколько параметров настройки, включая:

а) три ротора, которые должны быть выбраны из пяти доступных;

б) порядок, в котором эти роторы должны быть установлены;

в) параметры установок для коммутационной панели;

г) код с тремя буквами дня.

Чтобы зашифровать сообщение, оператор должен последовательно сделать шаги, перечисленные ниже:

1. Установить стартовую позицию роторов согласно коду дня. Например, если код был *HUA*, роторы должны быть инициализированы на *H, U* и *A* соответственно.

2. Выбрать случайный код с тремя буквами, например *ACF*. Зашифровать текст *ACFACF* (повторный код), используя начальную установку роторов шага 1. Например, предположим, что зашифрованный код – *OPNABT*.

3. Установить стартовые позиции роторов к *OPN* (половина зашифрованного кода).

4. Добавить зашифрованные шесть букв, полученных на шаге 2 (*OPNABT*), в конец к начальному сообщению.

5. Зашифровать сообщение, включая код с шестью буквами. Передать зашифрованное сообщение.

Чтобы расшифровывать сообщение, оператор должен сделать следующие шаги:

1. Получить сообщение и отделить первые шесть букв.

2. Установить стартовую позицию роторов согласно коду дня.

3. Расшифровать первые шесть букв сообщения, используя начальную установку шага 2.

4. Установить позиции роторов на первую половину расшифрованного кода.

5. Расшифровать сообщение (без первых шести букв).

Известно, что машина подстановки “Энигма” во время войны была взломана, хотя немецкая армия и остальная часть мира не знала об этом факте еще несколько десятилетий после того [6]. Хотя немецкий язык весьма сложен, союзники, так или иначе, получили некоторые копии машин. Следующим шагом был поиск параметров установки в течение каждого дня и кода, передаваемого для инициализации роторов для каждого сообщения. Изобретение первого компьютера помогло союзникам преодолеть эти трудности.

Шифрование методом решения задачи об укладании рюкзака

Первым алгоритмом для обобщенного шифрования с открытым ключом стал *алгоритм рюкзака*, разработанный *Ральфом Меркелем* и *Мартином Хеллманом* [5, 32]. Безопасность алгоритма рюкзака опирается на проблему рюкзака. Проблема рюкзака несложная. Дан набор предметов различной массы. Можно положить некоторые из этих предметов в рюкзак так, чтобы масса рюкзака стала равна заданному значению. Или более формально, дан набор значений m_1, m_2, ..., m_n и сумма C. Вычислить значение b_i, такие что:

$$C = b_1 \cdot m_1 + b_2 \cdot m_2 + \ldots + b_n \cdot m_n,$$

где b_i может быть нулем или единицей. Единица показывает, что предмет кладут в рюкзак, а ноль – не кладут.

Например, массы предметов могут иметь значения *1*, *5*, *6*, *11*, *14* и *20*. Можно упаковать рюкзак так, чтобы его масса стала равна 22. Это можно сделать только, если использовать массы *5*, *6* и *11*. Невозможно упаковать рюкзак так, чтобы его масса была равна *24*. В общем случае время, необходимое для решения этой проблемы, с ростом количества предметов в наборе растет экспоненциально.

Фокус в том, что на самом деле существуют две различные проблемы рюкзака, одна решается за линейное время, а другая, как считается, – нет. Легкую проблему можно превратить в трудную. Открытый ключ представляет собой трудную проблему, которую

легко использовать для шифрования, но невозможно для дешифрирования сообщений. Закрытый ключ является легкой проблемой, давая простой способ дешифрирования сообщения. Тому, кто не знает закрытый ключ, придется попытаться решить трудную проблему рюкзака.

В основе алгоритма рюкзака *Меркла-Хеллмана* лежит идея шифровать сообщение как решение набора проблем рюкзака. Предметы из набора выбираются с помощью блока открытого текста, по длине равного количеству предметов в куче (биты открытого текста соответствуют значениям *b*), а зашифрованный текст является полученной суммой.

Задачи об укладании рюкзака (ранца) формулируются следующим образом:

Задан вектор:

$$E = \left| e_к, e_{к-1}, \ldots, e_2, e_1 \right|,$$

который в дальнейшем будет играть роль ключа. Как правило, элементами этого вектора есть простые числа. Для однозначного зашифрования и расшифрования элементы вектора *E* избираются по правилу [13, 15]

$$e_i > \sum_{j=1}^{i-1} e_j, \quad i = 2,3,\ldots,k .$$

Каждый символ открытого сообщения m_i, что передается, представляется последовательностью из *k* битов:

$$m_i = \left| m_{i,k}, \quad m_{i,k-1}, \quad \ldots, \quad p_{i,1} \right|^T, \quad m_{i,j} \in \{0,1\},$$
$$j = 1,2,\ldots,k, i = 1,2,\ldots,n.$$

Как результат открытое сообщение представляет собой матрицу

$$M = \begin{vmatrix} m_{1,k} & m_{2,k} & \ldots & m_{n,k} \\ m_{1,k-1} & m_{2,k-1} & \ldots & m_{n,k-1} \\ \ldots & \ldots & \ldots & \ldots \\ m_{1,1} & m_{2,1} & \ldots & m_{n,1} \end{vmatrix},$$

где *n* – длина открытого сообщения.

Символы зашифрованных данных *C* выходят как произведение матрицы *E* и вектора-столбца *M*:

$$C = E \times M. \qquad (2.19)$$

Пример 2.23. Зашифровать открытое сообщение: *Обгон* (*14, 01, 03, 14, 13*) с использованием ключа в виде

$$E = \begin{vmatrix} 29 & 13 & 7 & 3 & 2 \end{vmatrix}.$$

Элементами вектора-строки есть простые числа.

Решение: Запишем символы открытого сообщения пятиразрядным двоичным кодом:

Запишемо символи відкритого повідомлення п'ятирозрядним двійковим кодом:

О	*б*	*г*	*о*	*н*
14	*01*	*03*	*14*	*13*
01110	*00001*	*00011*	*01110*	*01101*

Составим матрицу сообщения *M*:

$$M = \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{vmatrix}.$$

Сделаем соответствующие (2.19) операции:

$$C = \begin{vmatrix} 29 & 13 & 7 & 3 & 2 \end{vmatrix} \cdot \begin{vmatrix} 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \end{vmatrix} = \begin{vmatrix} 23 & 02 & 05 & 23 & 22 \end{vmatrix}.$$

Итак, зашифрованное сообщение будет иметь вид: *23, 02, 05, 23, 22* (*ЧВЕЧЦ*).

Пример 2.24. Расшифруем это зашифрованное сообщение.

Решение: Пятиразрядные двоичные числа открытого сообщения выходят из матричного уравнения (2.19).

Открытое сообщение можно получить, решив шесть уравнений:

$$C = \left| 29 \quad 13 \quad 07 \quad 03 \quad 02 \right| \cdot \begin{vmatrix} m_{1,5} & m_{2,5} & m_{3,5} & m_{4,5} & m_{5,5} \\ m_{1,4} & m_{2,4} & m_{3,4} & m_{4,4} & m_{5,4} \\ m_{1,3} & m_{2,3} & m_{3,3} & m_{4,3} & m_{5,3} \\ m_{1,2} & m_{2,2} & m_{3,2} & m_{4,2} & m_{5,2} \\ m_{1,1} & m_{2,1} & m_{3,1} & m_{4,1} & m_{5,1} \end{vmatrix} =$$

$$= \begin{vmatrix} 29 \cdot m_{1,5} + 13 \cdot m_{1,4} + 7 \cdot m_{1,3} + 3 \cdot m_{1,2} + 2 \cdot m_{1,1} \\ 29 \cdot m_{2,5} + 13 \cdot m_{2,4} + 7 \cdot m_{2,3} + 3 \cdot m_{2,2} + 2 \cdot m_{2,1} \\ 29 \cdot m_{3,5} + 13 \cdot m_{3,4} + 7 \cdot m_{3,3} + 3 \cdot m_{3,2} + 2 \cdot m_{3,1} \\ 29 \cdot m_{4,5} + 13 \cdot m_{4,4} + 7 \cdot m_{4,3} + 3 \cdot m_{4,2} + 2 \cdot m_{4,1} \\ 29 \cdot m_{5,5} + 13 \cdot m_{5,4} + 7 \cdot m_{5,3} + 3 \cdot m_{5,2} + 2 \cdot m_{5,1} \end{vmatrix} = \begin{vmatrix} 23 \\ 02 \\ 05 \\ 23 \\ 22 \end{vmatrix}.$$

Первый элемент сообщения (пятиразрядное двоичное число) получим, решив первое скалярное уравнение.

Первое скалярное уравнение

$$29 \cdot m_{1,5} + 13 \cdot m_{1,4} + 7 \cdot m_{1,3} + 3 \cdot m_{1,2} + 2 \cdot m_{1,1} = 23.$$

Это скалярное уравнение будет справедливым только в том случае, если $m_{1,5} = 0$, $m_{1,4} = 1$, $m_{1,3} = 1$, $m_{1,2} = 1$ і $m_{1,1} = 0$.

Это двоичный код – $14_{10} = 01110_2$.

Второе скалярное уравнение:

$$29 \cdot m_{2,5} + 13 \cdot m_{2,4} + 7 \cdot m_{2,3} + 3 \cdot m_{2,2} + 2 \cdot m_{2,1} = 02.$$

Данное скалярное уравнение будет справедливым только в том случае, если $m_{2,5} = 0$, $m_{2,4} = 0$, $m_{2,3} = 0$, $m_{2,2} = 0$ і $m_{2,1} = 1$.

Это двоичный код – $01_{10} = 00001_2$.

Третье скалярное уравнение:

$$29 \cdot m_{3,5} + 13 \cdot m_{3,4} + 7 \cdot m_{3,3} + 3 \cdot m_{3,2} + 2 \cdot m_{3,1} = 05.$$

Это скалярное уравнение будет справедливым только в том случае, если $m_{3,5} = 0$, $m_{3,4} = 0$, $m_{3,3} = 0$, $m_{3,2} = 1$ і $m_{3,1} = 1$.

Это двоичный код – $03_{10} = 00011_2$.

Четвертое скалярное уравнение:

$$29 \cdot m_{4,5} + 13 \cdot m_{4,4} + 7 \cdot m_{4,3} + 3 \cdot m_{4,2} + 2 \cdot m_{4,1} = 23.$$

Такое скалярное уравнение будет справедливым только в том случае, если $m_{4,5} = 0$, $m_{4,4} = 1$, $m_{4,3} = 1$, $m_{4,2} = 1$ і $m_{4,1} = 0$.

Это двоичный код – $14_{10} = 01110_2$.

Пятое скалярное уравнение:

$$29 \cdot m_{5,5} + 13 \cdot m_{5,4} + 7 \cdot m_{5,3} + 3 \cdot m_{5,2} + 2 \cdot m_{5,1} = 22.$$

Это скалярное уравнение будет справедливым только в том случае, если $m_{5,5} = 0$, $m_{5,4} = 1$, $m_{5,3} = 1$, $m_{5,2} = 0$ і $m_{5,1} = 1$.

Это двоичный код – $13_{10} = 01101_2$.

Итак, имеем открытое сообщение

01110	*00001*	*00101*	*01110*	*01101*
14	*01*	*05*	*14*	*13*
О	*б*	*г*	*о*	*н*

Из полученного выходит, что открытое сообщение: *Обгон*.

Было предложено большое количество вариантов реализации этого алгоритма, но почти все было взломано. Да и те, что остались, требуют для взлома гораздо меньших усилий, чем некоторые другие алгоритмы асимметричной криптографии.

2.2. ШИФРЫ ПЕРЕСТАНОВКИ

Шифр перестановки не заменяет один символ другим, вместо этого он изменяет местоположение символов в зашифрованном тексте. Символ в первой позиции исходного текста может появиться в десятой позиции зашифрованного текста. Символ, который находится в восьмой позиции исходного текста, может появиться в первой позиции зашифрованного текста. Другими словами, шифр перестановки ставит в другом порядке (перемещает) символы.

2.2.1. Шифры перестановки без использования ключа

Простые шифры перестановки, которые применялись в прошлом, не использовали ключ. Есть два метода для перестановки символов. В первом методе текст записывается в таблице столбец за

столбцом и затем передаётся строка за строкой. Во втором методе текст записан в таблицы строка за строкой и затем передаётся столбец за столбцом.

Пример 2.25. Хороший пример шифра без использования ключа – *шифр изгороди* (*rail fence cipher*). В этом шифре исходный текст размещен на двух линиях как зигзагообразный шаблон (что может рассматриваться как столбец за столбцом таблицы, которая имеет две строки); зашифрованный текст составляется при чтении шаблона строка за строкой. Например, чтобы передать сообщение *Meet me at the park* (*Встречай меня в парке*), отправитель пишет получателю (рис. 2.21).

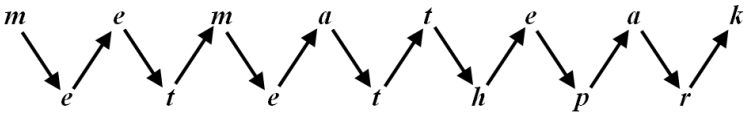


Рис. 2.21. Пример шифрования сообщения *Meet me at the park* с помощью шифра изгороди

Отправитель создает зашифрованный текст *MEMATEAKETET HPR*, посылая первую строку, сопровождаемую второй строкой. Получатель получает зашифрованный текст и разделяет его пополам (в этом случае вторая половина имеет на один символ меньше). Первая половина формы – первая строка; вторая половина – вторая строка. Получатель читает результат по зигзагу. Поскольку нет никакого ключа, и номер строк установлен (*2*), криптографический анализ зашифрованного текста был бы очень прост для злоумышленника. Все, что он должен знать, – это то, что используется шифр изгороди.

m	*e*	*e*	*t*
m	*e*	*a*	*t*
t	*h*	*e*	*p*
a	*r*	*k*	

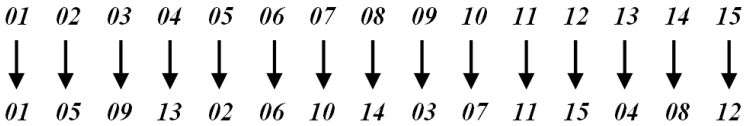
Рис. 2.22. Пример зашифролвания повідомлення *Meet me at the park* с помощью таблицы

Пример 2.26. Отправитель и получатель могут договориться о числе столбцов и использовать второй метод. Отправитель пишет тот же самый исходный текст, строка за строкой, в таблице из четырех столбцов (рис. 2.22).

Отправитель создает зашифрованный текст *MMTAEEHREAEKTTP*, передавая символы столбец за столб-

цом. Получатель получает зашифрованный текст и применяет обратный процесс. Он пишет полученное зашифрованное сообщение столбец за столбцом и читает его строка за строкой как исходный текст. Злоумышленник может легко расшифровать сообщение, если он знает число столбцов.

Шифр в примере 2.23 – реальный шифр перестановки. Далее покажем перестановку каждой буквы исходного текста и зашифрованный текст, базируясь на номерах их позиций:



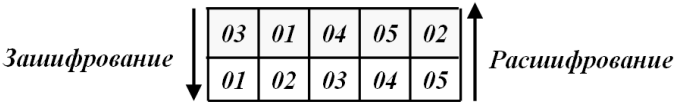
Второй символ в исходном тексте передвинулся на пятую позицию в зашифрованном тексте; третий символ передвинулся на девятую позицию; и так далее. Хотя символы переставлены, они сами являются шаблонами: (*01*, *05*, *09*, *13*), (*02*, *06*, *10*, *14*), (*03*, *07*, *11*, *15*) и (*04*, *08*, *12*). В каждой секции разность между двумя смежными номерами – *4*.

2.2.2. Шифры перестановки с использованием ключа

Бесключевые шифры перестановки переставляют символы, используя запись исходного текста одним способом (например, строка за строкой) и передачу этого текста в другом порядке (например, столбец за столбцом). Перестановка делается во всём исходном тексте, чтобы создать весь зашифрованный текст. Другой метод состоит в том, чтобы разделить исходный текст на группы заранее определенного размера, называемые блоками, а затем использовать ключ, чтобы переставить символы в каждом блоке отдельно.

Пример 2.27. Отправитель должен передать получателю сообщение *Enemy attacks tonight* (*Вражеские атаки сегодня вечером*). Отправитель и получатель согласились разделить текст на группы по пять символов и затем переставить символы в каждой группе. Ниже показана группировка после добавления фиктивного символа в конце, чтобы сделать последнюю группу одинаковой по размеру с другими: *Enemy attacks tonightz*.

Ключ, используемый для зашифрования и расшифрования, – ключ перестановки, который показывает, как переставлять символы. Для этого сообщения примем, что отправитель и получатель использовали следующий ключ:



Третий символ в блоке исходного текста становится первым символом в блоке зашифрованного текста, первый символ в блоке исходного текста становится вторым символом в блоке зашифрованного текста и так далее. Результаты перестановки: *EEMYN TAACT TKONS HITZG*

Отправитель передает зашифрованный текст *EEMYN TAACT TKONS HITZG* получателю. получатель делит зашифрованный текст на группы по 5 символов и, используя ключ в обратном порядке, находит исходный текст.

Шифры перестановки столбцов с использованием ключа

Современные шифры перестановки, чтобы достигнуть лучшего скремблирования, объединяют два подхода. Зашифрование и дешифрование делается в три шага.

Первый шаг: исходный текст записывается в таблицу строка за строкой.

Второй шаг: делается перестановка, изменяя порядок следования столбцов (переставляются столбцы в таблице по заданному ключу).

Третий шаг: зашифрованный текс считывается из новой таблицы столбец за столбцом.

Первые и третьи шаги обеспечивают бесключевое глобальное изменение порядка следования; второй шаг обеспечивает блоковую ключевую перестановку. Эти типы шифров упоминаются часто как *шифры перестановки столбцов по ключу*.

Пример 2.28. Предположим, что отправитель снова зашифровывает сообщение *Enemy attacks tonight*, на сей раз используя объединенный подход. Зашифрование и расшифрование показано на рис. 2.23.

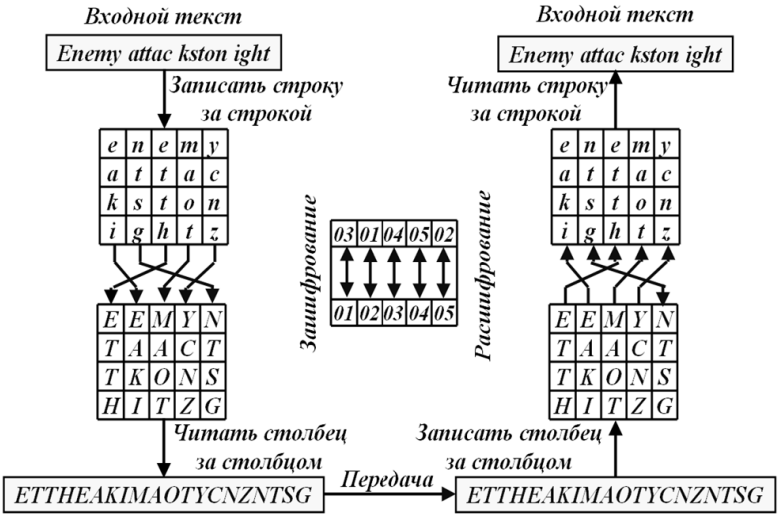


Рис. 2.23. Пример зашифрования и расшифрования путем перестановки с использованием ключа

Первая таблица, созданная отправителем, содержит исходный текст, записанный строка за строкой. Столбцы переставлены с использованием того же самого ключа, что и в предыдущем примере. Зашифрованный текст создан с помощью чтения символов из второй таблицы столбец за столбцом. Получатель делает те же самые три шага, но в обратном порядке. Он записывает зашифрованный текста в первую таблицу столбец за столбцом, переставляет столбцы данной таблицы, а затем считает символы из второй таблицы строку за строкой.

В примере 2.28 единственный ключ использовался в двух направлениях для изменения порядка следования столбцов – вниз для зашифрования, вверх для расшифрования. Обычно принято создавать два ключа для этого графического представления: один для зашифрования и один для расшифрования. Ключи накапливаются в таблицах, имеющих один адрес (вход) для каждого столбца. Вход содержит исходный номер столбца – номер столбца пункта назначения, указывающий его положение от номера входа. Рис. 2.24 показывает, как эти две таблицы могут быть созданы с помощью графического представления ключа.

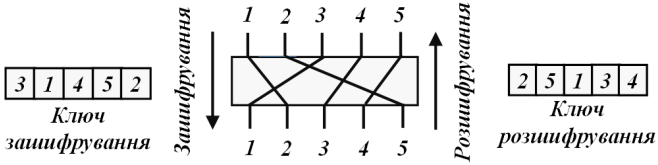


Рис. 2.24. Зашифрование/расшифрование в шифре перестановки с помощью двух ключей

Ключ зашифрования – (*3 1 4 5 2*). Первый вход показывает, что столбец *3* (содержание) в источнике становится столбцом *1* (положение или индекс входа) в пункте назначения. Ключ расшифрования – (*2 5 1 3 4*). Первый вход показывает, что столбец *2* в источнике становится столбцом *1* в пункте назначения.

Как найти ключ расшифрования, если дан ключ зашифрования или, наоборот, дан ключ расшифрования, а нужно найти ключ зашифрования? Процесс может быть выполнен вручную за несколько шагов, как это показано на рис. 2.25.

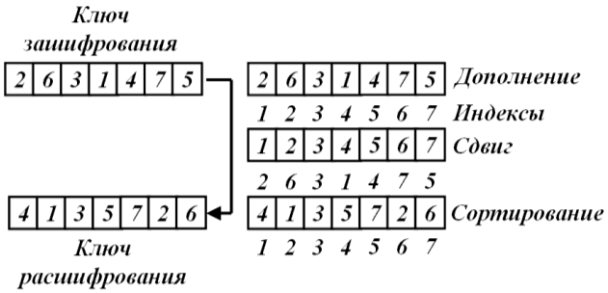


Рис. 2.25. Пояснення процесса инверсии ключа в шифре перестановки

Сначала добавим индексы к таблице ключей, потом сделаем сдвиг в соответствии с полученным ключом, и, наконец, сортируем пару согласно индексу.

Можно использовать матрицы, чтобы показать процесс зашифрования/расшифрования для шифра перестановки. Исходный текст $M(l \times g)$ и зашифрованный текст $C(l \times g)$ – матрицы размером $l \times g$, представляющие числовые значения символов; ключи $K(g \times g)$ – квадратные матрицы перестановки размером $g \times g$.

В таком случае алгоритм зашифрования данных можно представить в виде:

$$C(l \times g) = M(l \times g) \times K_{Ш}(g \times g), \qquad (2.20)$$

а расшифрование

$$M(l \times g) = C(l \times g) \times K_{P}(g \times g), \qquad (2.21)$$

где $K_{Ш}(g \times g)$ и $K_{P}(g \times g)$ – ключевые матрицы, которые используются для зашифрования и расшифрования соответственно. Ключевые матрицы являются обратными, то есть:

$$K_{P}(g \times g) = K_{Ш}^{-1}(g \times g) \quad \text{и} \quad K_{Ш}(g \times g) = K_{P}^{-1}(g \times g).$$

В данном алгоритме (2.20) и (2.21) ключевые матрицы описывают процесс перестановки столбцов, поэтому их чаще называют *матрицами перестановки*.

В матрицах перестановки каждая строка или столбец имеют строго одну единицу (*1*), и остальная часть значений – нули (*0*). Зашифрование выполняется умножением матрицы исходного текста на ключевую матрицу зашифрования (матрицу прямой перестановки), чтобы получить матрицу зашифрованного текста; расшифрование выполняется умножением зашифрованного текста на ключевую матрицу расшифрования (матрицу инверсной перестановки), после чего получаем исходный текст. Очень интересно, что матрица расшифрования в этом случае, как и всегда, – инверсия матрицы зашифрования. Однако нет никакой необходимости инвертировать матрицу расшифрования – ключевая матрица зашифрования может просто быть переставлена сдвигом строк и столбцов, чтобы получить ключевую матрицу расшифрования.

Пример 2.29. Передаваемое сообщение *Enemy attacks tonight*. Ключ зашифрования: *3, 1, 4, 5, 3*. Зашифровать сообщение с использованием перестановки столбцов матрицы исходных данных с ключем зашифрования: *3, 1, 4, 5, 3*.

Решение: Сообщения *Enemy attacks tonight* в цифровом эквиваленте имеет вид *04, 13, 04, 12, 24, 00, 19, 19, 00, 02, 10, 18, 19, 14, 13, 08, 06, 07, 19, 25*. Рис. 2.26 показывает процесс зашифрования сообщения.

Умножение матрицы исходного текста $M(4 \times 5)$ на ключевую матрицу зашифрования $K_{Ш}(5 \times 5)$ дает матрицу зашифрованного текста $C(4 \times 5)$. Матричная манипуляция требует изменения символов в примере 2.29 к их числовым значениям (от *00* до *25*). Обратите

внимание, что матричное умножение обеспечивает только перестановку столбцов; чтение и запись в матрицу должны быть обеспечены остальной частью алгоритма.

Рис. 2.26. Пояснение процесса зашифрования данных для примера 2.29 с использованием шифра перестановки

Итак, зашифрованное сообщение примет вид *04, 04, 12, 24, 13, 19, 00, 00, 02, 19, 19, 10, 14, 13, 18, 07, 08, 19, 25, 06*, что отвечает: *EEMYNTAACTTKONSHITZG*.

Пример 2.30. Расшифровать зашифрованное сообщение полученное в примере 2.29 с использованием перестановки столбцов матрицы исходных данных по ключу: *2, 5, 4, 1, 3*, который получается из инверсной матрицы зашифрования (см. пример 2.29).

Решение: Умножение матрицы зашифрованного текста $C(4 \times 5)$ на ключевую матрицу расшифрования $K_P(5 \times 5)$, дает матрицу исходного текста $M(4 \times 5)$.

Рис. 2.27 показывает процесс расшифрования сообщения.

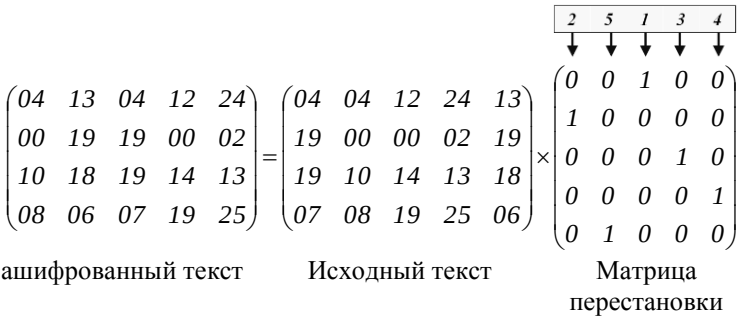


Рис. 2.27. Пояснение процесса расшифрования данных для примера 2.30 с использованием шифра перестановки

Итак, исходный текст при расшифровании примет вид: *04, 13, 04, 12, 24, 00, 19, 19, 00, 02, 10, 18, 19, 14, 13, 08, 06, 07, 19, 25*, что соответствует: *Enemy attacks tonightz*.

Криптографический анализ шифров перестановки

Шифры перестановки уязвимы к нескольким видам атак только на зашифрованный текст.

Статистическая атака

Шифр перестановки не изменяет частоту букв в зашифрованном тексте; он только переставляет буквы. Так что первая атака, которая может быть применена, – анализ частоты появления в зашифрованном тексте отдельных букв. Этот метод может быть полезен, если длина зашифрованного текста достаточно большая. Пример такой атаки уже рассматривался ранее. Однако шифры перестановки не сохраняют частоту появления пар и триграмм. Это означает, что злоумышленник не может использовать такие инструментальные средства. Фактически, если шифр не сохраняет частоту появления пар и триграмм, но сохраняет частоту появления отдельных букв, то вероятнее всего, что это шифр перестановки.

Злоумышленник, чтобы расшифровать сообщение, может попробовать все возможные ключи. Однако число ключей может быть огромно $1! + 2! + 3! + \ldots + L!$, где L – длина зашифрованного текста. Лучший подход состоит в том, чтобы попробовать отгадать число столбцов. Злоумышленник знает, что число столбцов делится на g. Например, если длина шифра – 20 символов, то $20 = 1 \times 2 \times 2 \times 5$. Это означает, что номерами столбцов может быть комбинация этих коэффициентов (*1, 2, 4, 5, 10, 20*). Однако только один столбец и только одна строка – маловероятные варианты [8, 32].

Пример 2.31. Предположим, что злоумышленник перехватил сообщение зашифрованного текста *EEMYNTAACTTKONSHITZG,* которое получено при выполнении примера 2.29.

Решение: Длина сообщения $L = 20$, число столбцов может быть *1, 2, 4, 5, 10* или *20*. Злоумышленник игнорирует первое значение, потому что это означает только один столбец и оно маловероятно.

1. Если число столбцов – *2*, единственные две перестановки – (*1, 2*) и (*2, 1*). Первое означает, что перестановки не было. Злоумышленник пробует вторую комбинацию. Он делит зашифрован-

ный текст на модули по два символа *EE MY NT AA CT TK ON SH IT ZG*. Затем он пробует переставлять каждый модуль из них, получая текст *ee ym nt aa tc kt no hs ti gz*, который не имеет смысла.

2. Если номер столбцов – *4*, тогда имеется *4! = 24* возможных перестановок. Первая перестановка (*1 2 3 4*) означает, что не было никакой перестановки. Злоумышленник должен попробовать остальные. После испытания всех *23* возможностей злоумышленник находит, что никакой исходный текст, при таких перестановках не имеет смысла.

3. Если число столбцов – *5*, тогда есть *5! = 120* перестановок. Первая (*1 2 3 4 5*) означает отсутствие перестановки. Злоумышленник должен попробовать остальные. Перестановка (*2 5 1 3 4*) приносит плоды – исходный текст *enemyattackstonightz*, который имеет смысл после удаления фиктивной буквы *z* и добавления пробелов (*Enemy attacks tonight*).

Атака по образцу

Другая атака шифра перестановки может быть названа атакой по образцу. Зашифрованный текст, созданный с помощью ключевого шифра перестановки, имеет некоторые повторяющиеся образцы. Следующий пример показывает зашифрованный текст, относительно которого известно, что каждый символ в зашифрованном тексте в примере 2.31 получается из исходного текста по следующему правилу:

01	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*
03	08	13	18	*01*	*06*	*11*	*16*	04	09	14	19	*05*	*10*	*15*	*20*	02	07	12	17

1-й символ в зашифрованном тексте получается из *3*-го символа исходного текста. *2*-й символ в зашифрованном тексте получается из *8*-го символа исходного текста. *20*-й символ в зашифрованном тексте получается из *17*-го символа исходного текста, и так далее. Исходя из вышеупомянутого списка, имеем пять групп образцов: (*3, 8, 13, 18*), (*1, 6, 11, 16*), (*4, 9, 14, 19*), (*5, 10, 15, 20*) и (*2, 7, 12, 17*). Во всех группах разность между двумя смежными номерами – *5*. Эта регулярность может использоваться криптографическим аналитиком, чтобы взломать шифр. Если злоумышленник знает или может предположить количество столбцов (в этом случае оно равняется *5*), тогда он может преобразовать зашифрованный текст, в группы по

четыре символа. Перестановка групп может обеспечить ключ к нахождению исходного текста.

Шифры с двойной перестановкой

Шифры с двойной перестановкой могут затруднить работу криптографичного аналитика. Примером такого шифра было бы повторение дважды алгоритма, используемого для зашифрования и расшифрования в примере 2.28. На каждом шаге может применяться различный ключ, но обычно ключ используется один и тот же.

Пример 2.32. Повторим пример 2.28, где использована двойная перестановка. Рис. 2.28 показывает процесс.

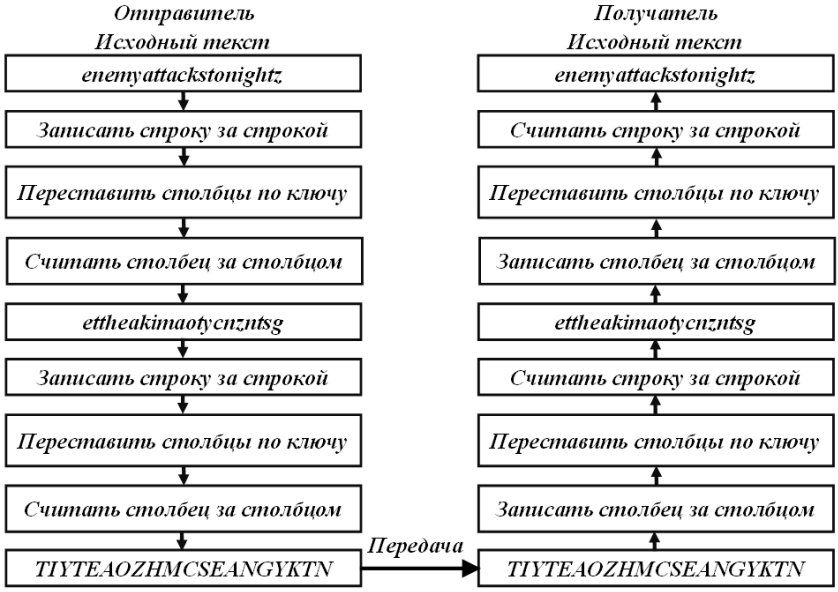


Рис. 2.28. Пример зашифрования и расшифрования путем двойной ключевой перестановки

Хотя криптографический аналитик может еще использовать частоту появления отдельных символов для статистической атаки на зашифрованный текст, атака по образцу теперь усложнена:

13 16 05 07 03 06 10 20 18 04

10 12 01 09 15 17 08 11 19 02.

Сравнив приведенный результат и результат примера 2.29, видим, что теперь нет повторяющихся образцов. Двойная перестановка удалила ту регулярность, которая была раньше.

Контрольные вопросы и задания

1. Поясните общую схему симметричного шифрования. Что общего имеют все методы шифрования с закрытым ключом?
2. Дайте определение шифров подстановки. Сформулируйте общие принципы для методов шифрования подстановкой.
3. Поясните сущность шифров моноалфавитной (одноалфавитной) подстановки.
4. Поясните сущность шифров многоалфавитной подстановки.
5. Поясните сущность аддитивного шифра подстановки.
6. Поясните сущность мультипликативного шифра подстановки.
7. Поясните сущность аффинного шифра подстановки.
8. Поясните сущность автоключевого шифра подстановки.
9. Поясните сущность шифра подстановки *Плейфера*.
10. Поясните сущность шифра подстановки *Виженера*.
11. К какой группе методов шифрования с закрытым ключом принадлежит метод с использованием таблицы *Виженера*? Какие алгоритмы зашифрования и расшифрования в этом методе?
12. Поясните сущность шифра *Хилла*.
13. Поясните сущность одноразовой системы шифрования.
14. Поясните сущность шифрования методом гаммирования.
15. Поясните сущность шифрования методом решения задачи об укладании рюкзака.
16. Дайте определение шифра перестановки.
17. Поясните сущность шифров перестановки без использования и с использованием ключа.
18. Зашифровать с использованием аддитивного шифра подстановки ($K = 3$) сообщение на русском языке “*комната”* (см. рис. 2.2).
19. Расшифровать с использованием аддитивного шифра подстановки ($K = 5$) зашифрованное сообщение “*УГФХИРЛИ”* при зашифровании которого использовали русский язык (см. рис. 2.2).
20. Определить ключ аддитивного шифра подстановки с использованием русского языка (см. рис. 2.2), если известна пара от-

крытый текст – зашифрованный текст: “*апельсин”* – “*ЗЦМТГШ ПФ”*.

21. Используя мультипликативный шифр подстановки, зашифровать сообщения на русском языке *отечество* с ключом $K = 5$ (см. рис. 2.2).

22. Расшифровать зашифрованный текст “*ЖРШЙЮХРАМШЩ*” при зашифровании которого использовали мультипликативный шифр ($K = 7$) и русский язык (см. рис. 2.2).

23. Используя аффинный шифр ($k_1 = 5$ и $k_2 = 7$) зашифровать сообщения на русском “*аргумент*” (см. рис. 2.2).

24. Расшифровать шифротекст “*ЛЗЬЩЗАЕУГ*” при зашифровании которого использовали аффинный шифр ($k_1 = 7$ и $k_2 = 5$) и русский язык (см. рис. 2.2).

25. Используя автоключевой шифр подстановки ($k_1 = 13$), зашифровать сообщения на русском языке “*гордость*” (см. рис. 2.2).

26. Расшифровать шифротекст “*ЩЯГАЮЮФУЫЯГО*” при зашифровании которого использовали автоключевой шифр подстановки ($k_1 = 11$) и русский язык (см. рис. 2.2).

27. Используя шифр Плейфера (см. рис. 2.9), зашифровать сообщение на русском языке “*криптография*”.

28. Расшифровать шифротекст “*АПЙШФЬШЧЬЛКЯ*” при зашифровании которого использовали шифр *Плейфера* и русский язык (см. рис. 2.9).

29. Используя шифр Виженера и ключевое слово “*километр*”, зашифровать сообщения на русском языке “*комната*” (см. табл. 2.8).

30. Расшифровать шифротекст “*АОЯИДЮХВРЫПЕ*” при зашифровании которого использовали шифр Виженера (см. табл. 2.8), ключевое слово “*роза*” и русский язык.

31. Используя шифр *Хилла* и ключевую матрицу K из примера 2.18, зашифровать сообщения на английском языке “*perpendicular*” (см. рис. 2.3).

32. Расшифровать шифротекст “*CVJRKDXAHIBS*” при зашифровании которого использовали шифр *Хилла*, ключевую матрицу K^{-1} из примера 2.18 и английский язык (см. рис. 2.1).

Раздел 3

ПРИНЦИПЫ ПОСТРОЕНИЯ СОВРЕМЕННЫХ СИММЕТРИЧНЫХ КРИПТОГРАФИЧЕСКИХ СИСТЕМ

Рассмотренные выше традиционные шифры с симметричным ключом ориентируются на символы. С появлением компьютерных систем необходимыми стали шифры ориентированы на биты, потому что информация, которую нужно зашифровывать, не всегда может состоять только из текста, но и также из чисел, графики, аудио - и видеоданных. Для зашифрования удобно превратить эти типы данных в поток битов и потом уже передавать зашифрованный текст. Кроме того, когда текст обработан на разрядном уровне, каждый символ заменен на *8* (или *16*) бит, а это означает, что количество символов становится в *8* (или *16*) раз больше. Именно смешивания большего количества символов увеличивает безопасность [6, 18].

3.1. СОВРЕМЕННЫЕ БЛОЧНЫЕ ШИФРЫ

Современные блочные шифры с симметричными ключами зашифровывают *n*-битовый блок исходного текста или расшифровывают *n*-битовый блок зашифрованного текста. Алгоритм зашифрования или расшифрования используют *k*-битовый ключ. Алгоритм расшифрования должен быть инверсией алгоритма зашифрования, и оба в работе используют один и тот же ключ шифрования так, чтобы получатель мог восстановить сообщение, передаваемое отправителем. Рис. 3.1 показывает общую идею зашифрования и расшифрования в современном блочном шифре.

Если сообщение имеет размер меньше, чем *n* бит, нужно добавить заполнение, чтобы создать этот *n*-разрядный блок; если сообщение имеет больше, чем *n* бит, оно должно быть разделено на *n*-разрядные блоки, и в случае необходимости нужно добавить к по-

следнему блоку соответствующее заполнение. Общие значения для *n* обычно *64, 128, 256* или *512* битов [12].

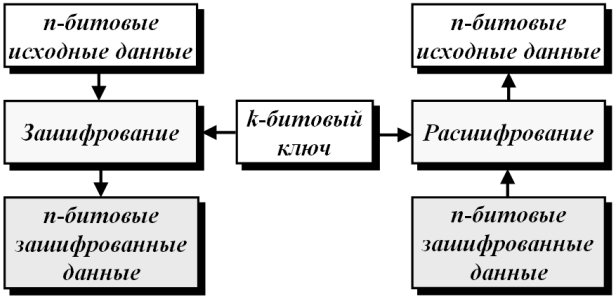


Рис. 3.1. Современный блочный шифр

Пример 3.1. Определить количество дополнительных бит, которые нужно добавить к сообщению, состоящему из *100* символов, если для кодирования используется *ASCII* коды по *8* битов и блоковый шифр принимает блоки *64* бита.

Решение. Для кодирования *100* символов, используя *ASCII* коды по *8* битов, тогда длина сообщения будет содержать *800* бит. Исходные данные должны делиться без остатка на *64*. Если $|M|$ и $|Pad|$ – длина сообщения и длина заполнения, то

$$(|M| + |Pad|) \bmod 64 = 0.$$

Отсюда следует, что

$$|Pad| = -|M| \bmod 64 = -800 \bmod 128 = -32 \bmod 64 = 32.$$

Это означает, что к сообщению нужно добавить *32* бита заполнения (например, нулей). Исходные данные тогда будут состоять из *832* битов или тринадцати *64*-разрядных блоков. Заметим, что только последний блок содержит заполнение. Шифратор использует алгоритм зашифрования тринадцать раз, чтобы создать тринадцать блоков зашифрованных данных.

3.1.1. Шифры подстановки и транспозиции

Современный блочный шифр может быть спроектирован так, чтобы действовать как шифр подстановки (замены) или как шифр транспозиции (перестановки). Такая идея используется и в тради-

ционных шифрах, за исключением того, что символы, которые будут заменены или перемещены, содержат биты вместо символов.

Если шифр спроектирован как шифр подстановки, значения бита *1* или *0* в исходных данных могут быть заменены либо на *0*, либо на *1*. Это означает, что исходные и зашифрованные данные могут иметь различное число единиц. Например, блок исходных данных на *64* бита, который содержит *12* нулей и *52* единицы, может быть представлен в зашифрованных данных *34* нулями и *30* единицами. Если шифр спроектирован как шифр перестановки (транспозиции), биты только меняют порядок следования (перемещаются), сохраняя то же самое число символов в исходных и зашифрованных данных. В любом случае, число возможных *n*-битовых исходных или зашифрованных данных равно 2^n, потому что каждый из *n* битов, использованных в блоке, может иметь одно из двух значений – *0* или *1*.

Современные блочные шифры спроектированы как шифры подстановки, потому что свойства транспозиции (сохранение числа единиц или нулей) делают шифр уязвимым к атакам исчерпывающего поиска, как это показывают нижеследующие примеры.

Пример 3.2. Предположим, имеется блочный шифр, где $n = 64$. Пусть в зашифрованных данных содержится *10* единиц, а остальные данные – нули (*54* нуля). Сколько испытаний типа "*проб и ошибок*" должен сделать злоумышленник, чтобы получить исходные данные путем перехвата зашифрованных данных в каждом из следующих случаев:

а) шифр спроектирован как шифр подстановки;

б) шифр спроектирован как шифр транспозиции.

Решение. В первом случае (подстановка) злоумышленник понятия не имеет, сколько единиц находится в исходных данных. Тогда он должен попробовать все возможные 2^{64} блока по *64* бита, чтобы найти один, который имеет смысл. Если бы злоумышленник мог пробовать *1* миллиард блоков в секунду, и тогда ему потребовалось бы сотни лет, прежде чем эта работа могла бы принести успех.

Во втором случае (перестановка) злоумышленник знает, что в исходных данных есть точно *10* единиц, потому что транспозиция не изменяет числа единиц (или нулей) в зашифрованных данных. Злоумышленник может начать атаку исчерпывающего поиска, используя только те *64*-битовые блоки, которые имеют точно *10* единиц. Есть только

$$\frac{n!}{(m_1)!(n-m_1)!} = \frac{64!}{10!\cdot 54!} = 151473214816 \qquad (3.1)$$

из 2^{64} слов по *64* бита, которые имеют точно *10* единиц.

В выражении (3.1) m_1 – число единиц в зашифрованных данных.

Злоумышленник может проверить их все меньше чем за *3* минуты, если он может проводить *1* миллиард испытаний в секунду.

Поэтому, стойкий к атаке исчерпывающего поиска, современный блочный шифр должен быть спроектирован как шифр подстановки.

3.1.2. Блочные шифры как групповые математические перестановки

Необходимо определить, является ли современный блочный шифр математической группой. Предположим сначала, что ключ у блочного шифра достаточно длинный, чтобы создать отображение любых возможных исходных данных в зашифрованные. Такие блочные шифры называются *полноразмерными ключевыми шифрами.* Практически, ключ меньше; длинный ключ можно применять только для некоторых отображений входной информации в выходную. Хотя блочный шифр должен иметь ключ, который является секретным при обмене между отправителем и получателем, в шифре используются также компоненты, которые не зависят от ключа [15, 16].

Полноразмерные ключевые шифры

Хотя полноразмерные ключевые шифры практически не используются, сначала обсудим их, чтобы сделать более понятным обсуждение шифров с ключом частичного размера.

Полноразмерный ключевой блочный шифр транспозиции перемещает биты, не изменяя их значения, так что может быть смоделирован как шифр перестановки *n*-мерного объекта с множеством *n!* таблиц перестановки, в которых ключ определяет, какая таблица используется отправителем и получателем. Для этого нужно иметь *n!* возможных ключей, и такой ключ должен иметь длину $[\log_2 n!]$ бит.

Пример 3.3. Построить модель и множество таблиц перестановки для блочного шифра транспозиции на *3* бита, где размер блока – *n = 3* бита.

Решение. Множество таблиц перестановки имеет *n! = 3! = 6* элементов, как показано на рис. 3.2.

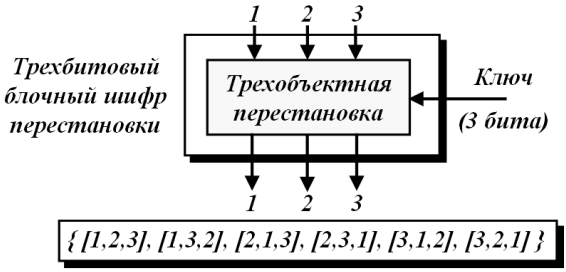


Рис. 3.2. Блочный шифр транспозиции в виде перестановки

Ключ должен быть длиной $[\log_2 n!] = 3$ бита. Заметим, что хотя ключ на *3* бита, он может выбрать $2^3 = 8$ различных отображений, используем только *6* из них.

Полноразмерные ключевые блочные шифры подстановки на первый взгляд кажутся, что не могут быть смоделированы как перестановка. Однако можно применить модель перестановки и для шифра подстановки, если декодировать входную информацию и кодировать выходную. Декодирование здесь означает преобразование *n*-разрядного целого числа в строку из 2^n-бит с единственной единицей и 2^n-1 нулями [4, 32]. Позиция единственной единицы указывает значение целого числа в упорядоченной последовательности позиций строки от *0* до 2^n-1. Поскольку новая входная информация имеет всегда единственную единицу, шифр может быть смоделирован как перестановка $2^n!$ объектов.

Пример 3.4. Построить модель и множество таблиц перестановки для блочного шифра подстановки блока на *3* бита.

Решение. Три входных исходных блока данных могут быть обозначены целыми числами от *0* до *7*. Их можно закодировать как строку, содержащую *8* битов с единственной единицей. Например, комбинация *000* может быть закодирована как *00000001* (первая единица справа); комбинация *101* может быть закодирована как

00100000 (шестая единица справа). Рис. 3.3 показывает модель и множество таблиц перестановки.

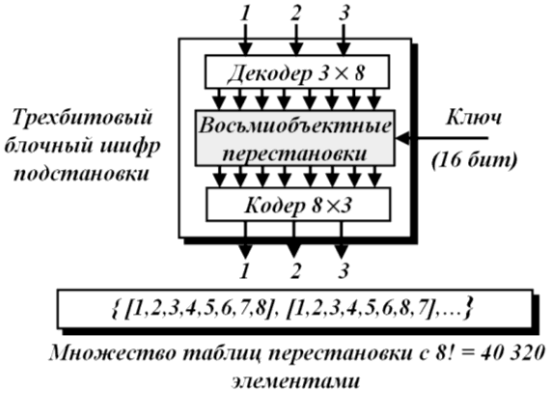


Рис. 3.3. Моделирование блочного шифра подстановки как шифра перестановки

Заметим, что число элементов в закодированном множестве намного больше, чем число элементов в шифре транспозиции (*8! = 40 320*). Ключ – также намного более длинный $[log_2 40\,320] = 16$ бит. Хотя ключ на *16* битов может определить *65 536* различных отображений, используются только *40 320*.

Полноразмерный ключ – это *n*-разрядный шифр транспозиции или блочный шифр подстановки. Он может быть смоделирован как шифр перестановки, но размеры их ключа различны: для шифра транспозиции ключ длиной - $[log_2 n!]$ для шифра подстановки ключ длиной $[log_2 (2n)!]$.

Полноразмерная ключевая транспозиция или шифр подстановки/перестановки показывает, что если зашифрование или расшифрование использует больше чем одну любую комбинацию из этих шифров, результат эквивалентен операции групповой перестановки. Известно [43], что две или больше каскадных перестановки могут всегда быть заменены единственной перестановкой. Это означает, что бесполезно иметь больше чем один каскад полноразмерных ключевых шифров, потому что эффект тот же самый, как и при наличии единственного шага.

Шифры ключа частичного размера

Фактические шифры не могут использовать полноразмерные ключи, потому что размер ключа становится несуразно большим, особенно для блочного шифра подстановки. Например, общий шифр подстановки *DES* применяет *64*-разрядный блочный шифр. Если бы проектировщики *DES* использовали полноразмерный ключ, он был бы $log_2(2^{64}!) = 2^{2^{70}}$ битов. На практике ключ для *DES* – только *56* битов, что является очень маленьким фрагментом полноразмерного ключа. Это означает, что *DES* использует только 2^{56} отображений из приблизительно $2^{2^{70}}$ возможных.

Зададим себе вопрос: можно ли установить, что многоступенчатая транспозиция с частичным ключом или подстановка – это группа перестановки с композицией операций? Ответ на этот вопрос чрезвычайно важен, потому что он говорит о том, является ли многоступенчатая версия с частичным шифром таким же средством шифрования, как и сам шифр. Этот факт позволяет достигнуть большей степени безопасности [19].

Частичный ключевой шифр – это подгруппа соответствующего размера ключа шифра (полноразмерного). Другими словами, если полноразмерный ключевой шифр – это группа $G = <Q, O>$, где Q – множество отображений и O – композиция операций, то шифр с ключом частичного размера должен представлять подгруппу $H = <U, O>$, где U – подмножество Q с теми же самыми операциями.

Например, было доказано, что многоступенчатый *DES* с *56*-битовым ключом не является группой, потому что подгруппа с 2^{56} отображениями не может быть создана из группы с $2^{64}!$ отображениями.

Частичный ключевой шифр есть группа с набором операций, если он является подгруппой соответствующего полноразмерного ключевого шифра.

Шифры без ключа

Использование отдельно шифра без ключа фактически бесполезно, возможно только их применение в качестве компонентов ключевых шифров.

Шифры транспозиции без ключа (или с фиксированным ключом) можно рассматривать как шифр транспозиции, реализованный

в аппаратных средствах. Фиксированный ключ (единственное правило перестановки) может быть представлен как таблица в случае реализации шифра в программном обеспечении. Далее будут рассмотрены шифры транспозиции без ключа, названные *P-блоками перестановки*, которые используются как стандартные блоки современных блочных шифров.

Шифр подстановки без ключа (или с фиксированным ключом) можно представить себе как заранее определенное отображение входной информации в выходной. Отображение может быть представлено как таблица, как математическая функция, а также другими способами. Далее будут рассмотрены шифры подстановки без ключей, названные *S*-блоками замены, которые применяются как стандартные блоки современных блочных шифров.

3.1.3. Компоненты современного блочного шифра

Современные блочные шифры обычно являются ключевыми шифрами подстановки, в которых ключ позволяет только частичные отображения возможных входов информации в возможные выходы. Однако эти шифры обычно не проектируются как единый модуль. Чтобы обеспечивать требуемые свойства современного блочного шифра, такие как рассеяние и перемешивание информации, этот шифр формируется как комбинация модулей транспозиции (называемых *P*-блоками перестановки), модулей подстановки (называемых *S*-блоками замены) и некоторыми другими модулями.

P-блок перестановки подобен традиционному шифру транспозиции символов. Он перемещает биты. В современных блочных шифрах можно найти три типа *P*-блоков перестановки: прямые *P*-блоки; *P*-блоки расширения и *P*-блоки сжатия, что и показано на рис. 3.4.

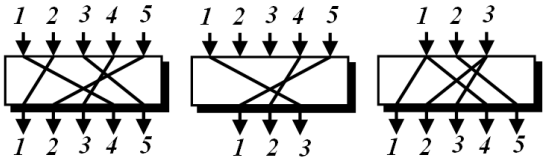


Рис. 3.4. Три типа *P*-блоков: а) прямой *P*-блок;
б) *P*-блок сжатия; в) *P*-блок расширения

Рис. 3.4 показывает прямой *P*-блок *5×5* (рис. 3.4, а), *P*-блок сжатия *5×3* (рис. 3.4, б) и *P*-блок расширения *3×5* (рис. 3.4, в). Рассмотрим каждый из них более подробно.

Прямые P-блоки

Прямой P-блок с *n* входами и *n* выходами – это перестановка с *n!* возможными отображениями.

Пример 3.5. Рис. 3.5 показывает все шесть возможных отображений прямого *P*-блока *3×3*.

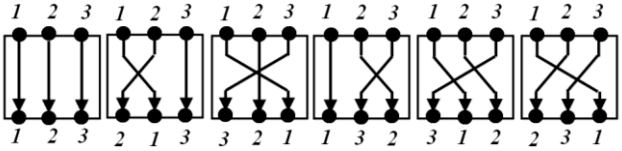


Рис. 3.5. Возможные отображения прямого *P*-блока *3×3*

Хотя прямой *P*-блок может использовать ключ, чтобы определить одно из *n!* отображений, обычно прямые *P*-блоки – без применения ключа, то есть отображение задано заранее. Если прямой *P*-блок задан заранее и реализован в аппаратных средствах, или если он реализован в программном обеспечении, таблицы перестановок задают правило отображения. Во втором случае входы в таблице указывают в позиции, в которых указаны позиции выходов. Табл. 3.1 дает пример таблицы перестановок, когда *n* равно *32*.

Таблица. 3.1 имеет *32* табличных входа, которые фиксируют соответствие *32* информационным входам. Позиция (индекс) входа соответствует выходу. Например, первый табличный вход содержит номер *26*. Это означает, что первый выход будет соответствовать *26*-му входу. Поскольку последний табличный вход – *7*, это означает, что, *32*-й выход будет соответствовать седьмому информационному входу, и так далее.

Таблица 3.1

Пример таблицы перестановки для прямого *P*-блока

	Номера битов															
Вход	*26*	*18*	*10*	*02*	*28*	*20*	*12*	*04*	*30*	*22*	*14*	*06*	*32*	*24*	*16*	*08*
Выход	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Вход	*25*	*17*	*09*	*01*	*27*	*19*	*11*	*03*	*29*	*21*	*13*	*05*	*31*	*23*	*15*	*07*
Выход	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*

Прямые *P*-блоки являются обратимыми. Это означает, что можно использовать прямой *P*-битовый блок для зашифрования, а потом использовать его для расшифрования. Таблицы перестановки, однако, должны быть обратимыми по отношению друг к другу.

Пример 3.6. Необходимо составить таблицу перестановки для прямого *P*-блока 8×8, которая перемещает два средних бита (биты *4* и *5*) во входном слове к двум крайним битам (биты *1* и *8*) выходного слова. Относительные позиции других битов не изменяются.

Решение. Нужно создать прямой *P*-блок с таблицей *[4, 1, 2, 3, 6, 7, 8, 5]*. Относительные позиции бит *1, 2, 3, 6, 7* и *8* не изменяются, но первый информационный выход связан с четвертым информационным входом, восьмой информационный выход – с пятым информационным входом. Перестановки для такого прямого *P*-блока приведены в табл. 3.2.

Таблица 3.2

Таблица перестановки для прямого *P*-блока для примера 3.6

	Номера битов							
Вход	*04*	*01*	*02*	*03*	*06*	*07*	*08*	*05*
Выход	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*

P-блоки сжатия

P-блок сжатия – это *P*-блок с *n* входами и *m* выходами, где $m < n$. Некоторые из информационных входов блокированы и не связаны с выходом (см. рис. 3.4, б). *P*-блоки сжатия, используемые в современных блочных шифрах, обычно являются безключевыми с таблицей перестановки, которая указывает правила перестановки бит. Необходимо учитывать, что таблица перестановок для *P*-блока сжатия имеет *n* табличных входов – от *1* до *n*, но в содержании каждого табличного выхода некоторые из них могут отсутствовать (те информационные входы, которые блокированы). Таблица 3.3 показывает пример таблицы перестановки для *P*-блока сжатия 32×24, у которых входы *7, 8, 9, 16, 23, 24* и *25* блокированы.

P-блоки сжатия используются, когда нужно переставить биты и в то же время уменьшить число битов для следующей ступени.

P-блоки сжатия необратимы. В *P*-блоках сжатия отдельные входы могут быть отброшены в процессе зашифрования, но алгоритм расшифрования не имеет ключа, чтобы восстановить отброшенный бит.

Таблица 3.3

Пример таблицы перестановки для *P*-блока сжатия *32×24*

	Номера битов											
Вход	*01*	*02*	*03*	*21*	*22*	*26*	*27*	*28*	*29*	*13*	*14*	*17*
Выход	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*
Вход	*18*	*19*	*20*	*04*	*05*	*06*	*10*	*11*	*12*	*30*	*31*	*32*
Выход	*13*	*14*	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*

Рис. 3.6 демонстрирует случай использования *P*-блока сжатия для зашифрования и расшифрования.

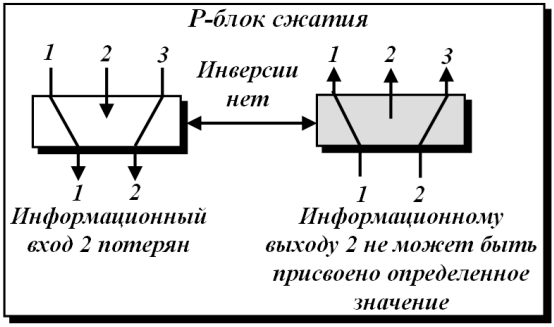


Рис. 3.6. *P*-блок сжатия как необратимый компонент

P-блоки расширения

P-блок расширения – *P*–блок с *n* входами и *m* выходами, где $m > n$. Некоторые из входов связаны больше чем с одним выходом (см. рис. 3.4, в). *P*-блоки расширения, используемые в современных блочных шифрах, обычно без ключа. Правила перестановки бит указываются в таблице. Таблица перестановки для *P*–блока расширения имеет *m* табличных выходов, но *m–n* входов (те входы, которые связаны больше чем с одним информационным выходом). Табл. 3.4 показывает пример таблицы перестановки для *P*-блока расширения *12×16*. Обратите внимание, что каждый из *1, 3, 9* и *12* входов соединен с двумя выходами.

P–блоки расширения используются, когда нужно переставить биты и то же время увеличить число битов для следующего каскада шифрования.

Таблица 3.4

Пример таблицы перестановки для *P*-блока расширения *12*× *16*

	Номера битов															
Вход	*01*	*09*	*10*	*11*	*12*	*01*	*02*	*03*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*12*
Выход	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*

Пример 3.7. Рис. 3.7 показывает, как нужно изменить таблицу перестановки в случае одномерной таблицы.

Рис. 3.7. Изменение таблицы перестановки

P-блоки расширения также необратимы. В *P*-блоке расширения некоторые входы в процессе зашифрования могут быть отображены более чем в один выход, но алгоритм расшифрования не имеет ключа и не может определить тем самым, какие из нескольких входов отображены в данном выходе. Рис. 3.8 демонстрирует случай использования *P*-блока расширения для зашифрования и расшифрования.

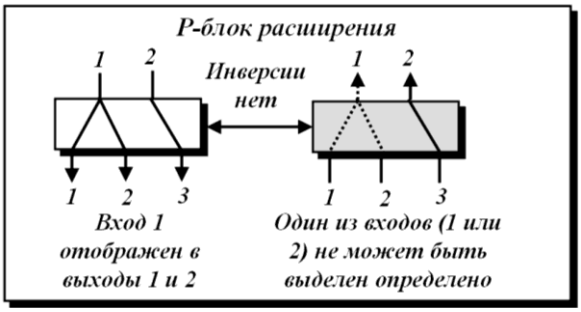


Рис. 3.8. *P*-блок расширения как необратимый компонент

Рис. 3.6 и рис. 3.8 также показывают, что *P*-блок сжатия не является обратным шифром *P*-блока расширения и наоборот. Это означает, что если использовать *P*-блок сжатия для зашифрования, то нельзя использовать *P*-блок расширения для расшифрования и наоборот. Однако, как будет показано ниже, есть шифры, которые применяют *P*-блоки сжатия или расширения для шифрования; но их эффективность хуже, чем у некоторых других способов.

S-блоки подстановки

S-блок подстановки (замены) можно представить себе как миниатюрный шифр подстановки. Этот блок может иметь различное число входов и выходов. Другими словами, вход к *S*-блоку подстановки может быть *n*-битовым словом, а выход может быть *m* разрядным словом, где *m* и *n* – не обязательно одинаковые числа. Хотя *S*-блок подстановки может быть ключевым или без ключа, современные блочные шифры обычно используют *S*-блоки подстановки без ключей, где отображение от информационных входов к информационным выходам заранее определено.

S-блок подстановки – $m \times n$ модуль подстановки, где m и n не обязательно равны.

В *S*-блоке подстановки с n входами и m выходами обозначим входы $x_1, x_2, \ldots, x_n$ и выходы $y_1, y_2, \ldots, y_m$. Соотношения между входами и выходами могут быть представлены как система уравнений

$$y_1 = f_1(x_1, x_2, \ldots, x_n);$$
$$y_2 = f_2(x_1, x_2, \ldots, x_n);$$
$$\ldots$$
$$y_m = f_m(x_1, x_2, \ldots, x_n).$$

В линейном *S*-блоке подстановки вышеупомянутые соотношения могут быть выражены как:

$$y_1 = a_{1,1} \cdot x_1 \oplus a_{1,2} \cdot x_2 \oplus \ldots \oplus a_{1,n} \cdot x_n;$$
$$y_2 = a2,1 \cdot x_1 \oplus a_{2,2} \cdot x_2 \oplus \ldots \oplus a_{2,n} \cdot x_n;$$
$$\ldots$$
$$y_m = a_{m,1} \cdot x_1 \oplus a_{m,2} \cdot x_2 \oplus \ldots \oplus a_{m,n} \cdot x_n.$$

В нелинейном *S*-блоке подстановки тоже можно всегда задать для каждого выхода указанные выше соотношения.

Пример 3.8. Пусть в *S*-блоке подстановки с тремя входами и двумя выходами имеем

$$y_1 = x_1 \oplus x_2 \oplus x_2 \oplus x_3; \quad y_2 = x_1.$$

S-блок подстановки линеен, потому что

$$a_{1,1} = a_{1,2} = a_{1,3} = a_{2,1} = 1 \quad \text{i} \quad a_{2,2} = a_{2,3} = 0.$$

Эти соотношения могут быть представлены матрицами, как показано ниже:

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = \begin{pmatrix} 111 \\ 100 \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}.$$

Пример 3.9. В *S*-блоке подстановки с тремя входами и двумя выходами имеем

$$y_1 = (x_1)^3 + x_2;$$

$$y_2 = (x_1)^2 + x_1 \cdot x_2 + x_3,$$

где умножение и сложение проводится в поле *Галуа GF(2)*. *S*-блок подстановки нелинеен, потому что нет линейных соотношений между входами и выходами.

Пример 3.10. Следующая таблица (см. рис. 3.9) определяет отношения между входами/выходами для *S*-блока подстановки размера 3×2. Крайний левый бит входа определяет строку; два самых правых бита входа определяют столбец. Два бита выхода – это значение на пересечении секции выбранной строки и столбца.

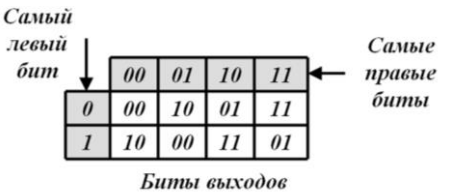


Рис. 3.9. Таблица *S*-блока подстановки для примера 3.10

S-блоки подстановки – это шифры подстановки, в которых отношения между входом и выходом определены таблицей или математическим соотношением. *S*-блок может или не может быть обратимым. В обратимом *S*-блоке подстановки число входных битов должно быть равным числу бит выхода.

Пример 3.11. Рис. 3.10 показывает пример обратимого *S*-блока подстановки. Одна из таблиц используется в алгоритме зашифрования; другая таблица – в алгоритме расшифрования.

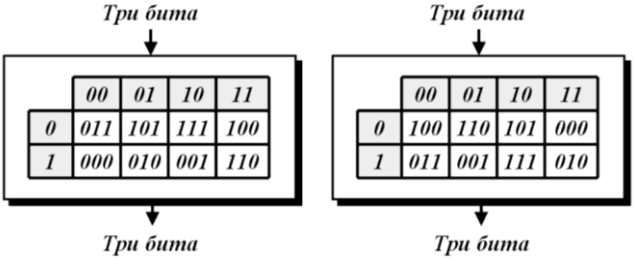


Рис. 3.10. Таблицы *S*-блока подстановки для примера 3.11

В каждой таблице крайний левый бит входа определяет строку; следующие два бита определяют столбец. Выход – это значение на пересечении строки и столбца таблицы.

Например, если вход к левому блоку – *001*, выход – *101*. Вход *101* в правой таблице дает выход *001*. Это показывает, что эти две таблицы позволяют получить обратный результат по отношению друг к другу.

Исключающее ИЛИ

Важный компонент в большинстве блоков шифрования – *операция исключающее или (xor)*. Операции сложения и вычитания в поле *Галуа GF(2^n)* выполняются с помощью одной и той же операции, называемой *исключающее или (xor)*.

Пять свойств операции *xor* в поле *Галуа GF(2^n)* делают эту операцию очень удобной для использования в блочном шифре:

1. *Замкнутость.* Это свойство гарантирует, что в результате этой операции два *n*-битовых слова дают другое *n*-битовое слово.

2. *Ассоциативность.* Это свойство позволяет использовать больше чем одно *xor*, которое можно вычислять в любом порядке.

$$x \oplus (y \oplus z) \leftrightarrow (x \oplus y) \oplus z.$$

3. *Коммутативность.* Это свойство позволяет менять местами операторы (входную информацию), не изменяя результат (выходную информацию).

$$x \oplus y \leftrightarrow y \oplus x.$$

4. *Существование нулевого (тождественного) элемента.* Нулевой элемент для операции *xor* – слово, которое состоит из всех нулей, или (*00...0*). Это подразумевает, что существует слово с нейтральными элементами, которое при проведении операции не изменяет слово.

$$x \oplus (00\ldots00) = x.$$

Это свойство используется в шифре *Фейстеля*, который рассмотрим далее.

5. *Существование инверсии.* В поле *Галуа* $GF(2^n)$ каждое слово есть аддитивная инверсия самого себя. Это подразумевает, что проведение операции *xor* слова с самим собой приводит к нулевому элементу:

$$x \oplus x = (00...00).$$

Используем это свойство в шифре *Фейстеля*, который рассмотрим тоже далее.

6. *Дополнение.* Операция дополнения – одноместная операция (один информационный вход и один информационный выход), которая инвертирует каждый бит в слове. Нулевой бит изменяется на единичный бит, а единичный бит – на нулевой бит.

При шифровании вызывают интерес операции с дополнением относительно операции *xor*. Если $\bar{x}$ – дополнение *x*, тогда верны следующие два соотношения:

$$x \oplus \bar{x} = (11\ldots11) \quad \text{и} \quad x \oplus (11\ldots11) = \bar{x}.$$

Воспользуемся этим свойством далее, когда будет обсуждаться безопасность некоторых шифров.

7. *Инверсия.* Инверсия компонента в шифре имеет смысл, если компонент представляет одноместную операцию (один вход и один выход). Например, *P*-блок без ключа или *S*-блок без ключа могут быть обратимыми, потому что они имеют один вход и один выход (рис. 3.11).

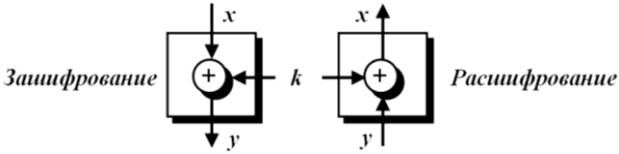


Рис. 3.11. Обратимость операции *xor*

Операция *исключающее или* (*xor*) – бинарная операция. Инверсия операции *xor* может иметь смысл, только если один из входов зафиксирован (один и тот же при зашифровании и расшифровании). Например, если один из входов – ключ, который обычно является одним и тем же при зашифровании и расшифровании, тогда операция *xor* является обратимой, как показано на рис. 3.11.

На рис. 3.11 свойство аддитивной инверсии подразумевает, что

$$y = x \oplus k \quad \text{и} \quad x = k \oplus y.$$

Воспользуемся этим это свойством, когда будет обсуждаться структура блоковых шифров.

Циклический сдвиг

Следующий компонент, применяемый в некоторых современных блочных шифрах, – *операция циклического сдвига*. Сдвиг может быть влево или вправо. Круговая операция левого сдвига сдвигает каждый бит в *n*-битовом слове на *k* позиции влево; крайние левые *k*-биты удаляются слева и становятся самыми правыми битами. Круговая операция правого сдвига сдвигает каждый бит в *n*-битовом слове на *k* позиций вправо; самые правые *k*-биты справа удаляются и становятся крайними левыми битами. Рис. 3.12 показывает и левые и правые операции в случае, где $n = 8$ и $k = 3$.

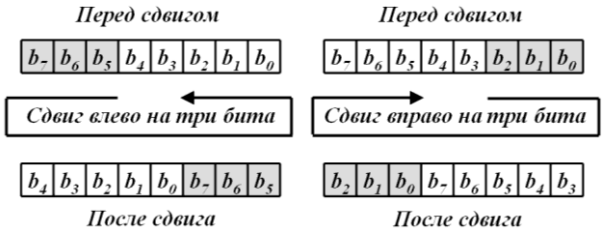


Рис. 3.12. Пример циклического сдвига *8* битового слова влево или вправо

Циклическая операция сдвига смешивает биты в слове и помогает скрыть образцы в первоначальном слове. Хотя число позиций, на которые будут сдвинуты биты, может использоваться как ключ, циклическая операция сдвига обычно без ключа; значение k устанавливается и задается заранее.

Циклическая операция левого сдвига – инверсия операции правого сдвига, т.е. является обратимой. Если одна из них используется для зашифрования, другая может применяться для расшифрования.

Операция циклического сдвига имеет два свойства.

Первое – это смещение по модулю n. Другими словами, если $k = 0$ или $k = n$, никакого смещения не происходит. Если k является большим, чем n, тогда входная информация сдвигается на $k \bmod n$ бит.

Второе свойство, операция циклического сдвига над соединением операций – есть группа. Это означает, что если смещение делается неоднократно, то одно и то же значение может появиться несколько раз.

Замена

Операция замены – специальный случай операции циклического сдвига, где $k = n/2$ означает, что эта операция возможна, только если n – четный номер. Поскольку сдвиг влево $n/2$ – то же самое, что сдвиг $n/2$ вправо, эта операция является обратимой. Операция замены для зашифрования может быть полностью раскрыта операцией замены при расшифровании. Рис. 3.13 иллюстрирует операцию замены для слова из *8* битов.

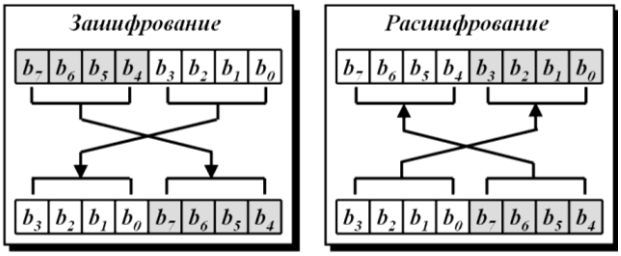


Рис. 3.13. Операция замены в *8* битовом слове

Разбиение и объединение

Следующие операции, применяемые в некоторых блочных шифрах, – *разбиение* и *объединение*. Разбиение обычно разделяет *n*-битовое слово в середине, создавая два слова равной длины. Объединение связывает два слова равной длины, чтобы создать *n*-битовое слово. Эти две операции инверсны друг другу и могут использоваться как пара, чтобы уравновесить друг друга. Если одна используется для зашифрования, то другая – для расшифрования. Рис. 3.14 показывает эти две операции для случая *n = 8*.

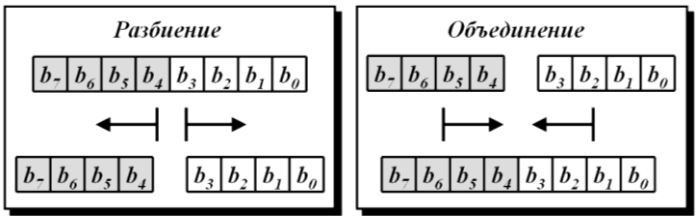


Рис. 3.14. Операции разбиение и объединения *8*-битового слова

3.2. СОСТАВНЫЕ ШИФРЫ

Шеннон ввел понятие составные шифры. *Составной шифр* – комплекс, который объединяет подстановку, перестановку и другие компоненты, рассмотренные выше.

3.2.1. Рассеивание и перемешивание

Идея *Шеннона* в представлении составного шифра должна была дать возможность блочным шифрам иметь две важных свойства: *рассеивание* и *перемешивание*.

Рассеивание должно скрыть отношения между зашифрованными и исходными данными. Это "*собьет с толку*" злоумышленника, который использует статистику зашифрованных данных, чтобы найти исходные данные. Рассеивание подразумевает, что каждый бит (символ) в зашифрованных данных будет зависеть от одного или всех битов (символов) в исходных данных. Другими словами, если один единственный бит в исходных данных изменен, то несколько или все биты в зашифрованных данных будут также изменены.

Идея относительно *перемешивания* состоит в том, что оно должно скрыть зависимость между зашифрованными данными и ключом. Это также "*собъет с толку*" злоумышленника, который стремится использовать зашифрованные данные, чтобы найти ключ. Другими словами, если один единственный бит в ключе будет изменен, все биты в зашифрованных данных будут также изменены.

3.2.2. Раунды

Рассеяние и перемешивание могут быть достигнуты использованием повторения составных шифров, где каждая итерация – комбинация *S*-блоков, *P*-блоков и других компонентов. Каждая итерация называется *раундом*. Блочный шифр использует ключевой список, или генератор ключей, который создает различные ключи для каждого раунда из ключа шифра. В *N*-раундном шифре, чтобы создать зашифрованные данные, исходные данные зашифровуются *N* раз; соответственно, зашифрованные данные расшифровываются *N* раз. Данные, созданные на промежуточных уровнях (между двумя раундами), называются *средними данными*. Рис. 3.15. показывает простой составной шифр с двумя раундами.

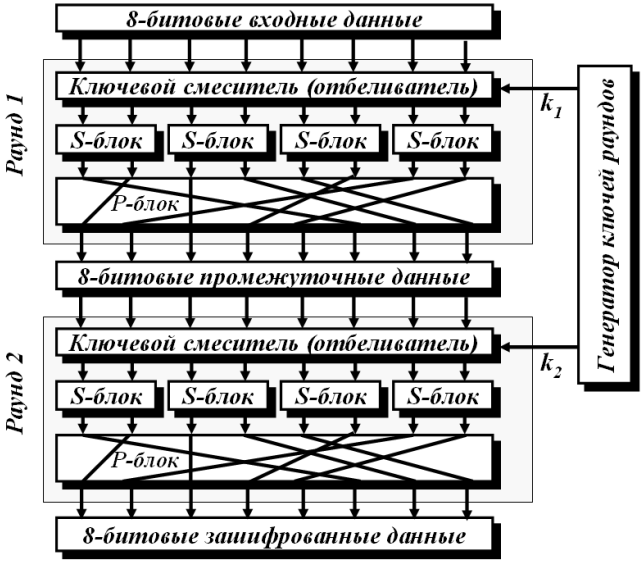


Рис. 3.15. Составной шифр, имеющий два раунда

На практике составные шифры имеют больше чем два раунда. На рис. 3.15 в каждом раунде проводятся три преобразования:

1. *8*-битовые данные смешиваются с раундовым ключом, чтобы сделать биты данных равновероятными (скрыть биты, используя ключ) – "*отбелить*" данные (*whiting*). Это обычно делается с помощью операции *xor* слова из *8* бит с раундовым ключом также из *8* бит.

2. Данные с выходов "*отбеливателя*" разбиваются на четыре группы по *2* бита и подаются в четыре *S*-блока замены. Значения битов изменяются в соответствии с построением *S*-блоков замены в этом преобразовании.

3. С выходов *S*-блоков замены данные поступают в *P*-блок перестановки, при этом биты должны быть переставлены так, чтобы в следующем раунде результат с выхода каждого блока поступил на различные входы.

Упрощенный составной шифр, схема которого показана на рис. 3.15, используя комбинацию *S*-блоков замены и *P*-блоков перестановки, может гарантировать рассеивание.

В первом раунде, после проведения операции *xor* с соответствующими битами раундового ключа k_1, изменяются два бита (биты *7* и *8*) через *S*-блок *4*. Бит *7* переставлен и становится битом *2*; бит *8* переставлен и становится битом *4*. После первого раунда бит *8* изменяет биты *2* и *4*. Во втором раунде бит *2* после проведения операции *xor* с соответствующими битами ключа k_2 изменяются два бита (биты *1* и *2*) через *S*-блок *1*. Бит *1* переставлен и становится битом *6*; бит *2* переставлен и становится битом *1*. Бит *4* после проведения операции *xor* с соответствующим битом раундового ключа k_2 изменяет биты *3* и *4*. Бит *3* остается, бит *4* переставлен и становится битом *7*. После второго раунда из *8* бит, изменены биты *1, 3, 6* и *7*.

Прохождение этих шагов в другом направлении (от зашифрованных до исходных данных) показывает, что каждый бит в зашифрованных данных изменяет исходные данные на несколько битов.

На рис. 3.16 показано, как изменение одного единственного бита в исходных данных вызывает изменение многих битов в зашифрованных данных.

Рис. 3.16 также доказывает, что свойство перемешивания может быть получено с помощью составного шифра. Четыре бита зашифрованных данных (биты *1, 3, 6* и *7*) преобразованы с помощью трех

битов в раундовых ключах (бит *8* в k_1 и биты *2* и *4* в k_2). Прохождение в обратном направлении показывает, что каждый бит раундового ключа в каждом раунде затрагивает несколько битов в зашифрованных данных. Отношения между битами зашифрованных данных и битами раундовых ключей показаны в затененных прямоугольниках.

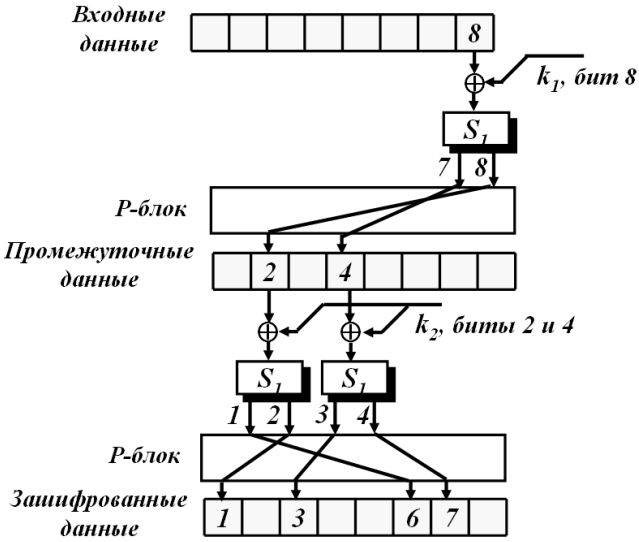


Рис. 3.16. Рассеивание и перемешивание в блоковом шифре

Чтобы улучшить рассеивание и перемешивание, практические шифры используют крупные блоки данных, больше *S*-блоков замены и больше раундов. Очевидно, что некоторое увеличение числа раундов при использовании большого числа *S*-блоков замены может создать лучший шифр, в котором зашифрованные данные выглядят все более как случайное *n*-битовое слово. Таким образом, отношения между зашифрованными и исходными данными будут полностью скрыты (рассеяны). Увеличение числа раундов увеличивает число ключей раундов, что лучше скрывает отношения между зашифрованными данными и ключом.

3.2.3. Два класса составных шифров

Современные блочные шифры – все составные, но они разделены на два класса. *Шифры в первом классе* используют и обратимые, и необратимые компоненты. Эти шифры упоминаются обычно как

шифры Фейстеля. *Шифры во втором классе* применяют только обратимые компоненты. Следует обратить внимание на шифры в этом классе как *шифры не-Фейстеля* (из-за отсутствия другого названия).

Фейстель разработал очень интеллектуальный и интересный шифр, который использовался в течение многих десятилетий. Шифр *Фейстеля* может иметь три типа компонентов: *самообратимый*, *обратимый* и *необратимый*.

Шифр *Фейстеля* содержит в блоках все необратимые элементы и использует один и тот же модуль в алгоритмах зашифрования и расшифрования. Вопрос в том, как алгоритмы зашифрования и расшифрования позволяют инвертировать открытые и закрытые данные друг в друга, если каждый содержит необратимый модуль. *Фейстель* показал, что они могут быть сбалансированы [8, 27].

Чтобы лучше понять шифр *Фейстеля*, посмотрим, как можно использовать один и тот же необратимый компонент в алгоритмах зашифрования и расшифрования. Эффекты необратимого компонента в алгоритме зашифрования могут быть отменены в алгоритме расшифрования, если использовать операцию *xor*, как показано на рис. 3.17.

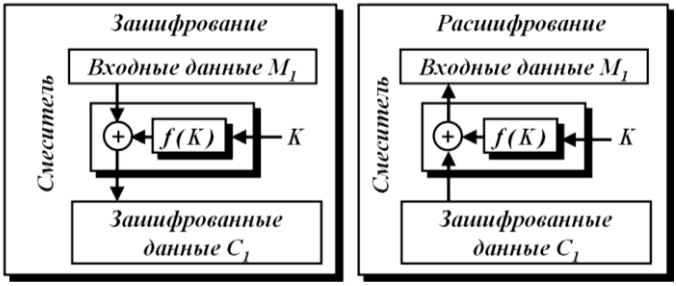


Рис. 3.17. Использование необратимого компонента в алгоритмах зашифрования и расшифрования Фейстеля

При зашифровании ключ поступает на вход необратимой функции *f(K)*, которая является одним из слагаемых оператора *xor* с исходными данными. В результате получается зашифрованные данные. Будем называть комбинацию функции *f(K)* и операции *xor* – *смесителем* (из-за отсутствия другого названия). Смеситель играет важную роль в более поздних вариантах шифра *Фейстеля*.

Поскольку ключ при зашифровании и расшифровании один и тот же, можно доказать, что два алгоритма инверсны друг другу.

Другими словами, если $C_2 = C_1$ (любое изменение в зашифрованных данных в течение передачи), то $M_2 = M_1$.

Зашифрование:

$$C_1 = M_1 \oplus f(K).$$

Расшифрование:

$$M_2 = C_2 \oplus f(K) = C_1 \oplus f(K) = M_1 \oplus f(K) \oplus f(K) =$$
$$= M_1 \oplus (00...0) = M_1.$$

Обратите внимание, что для вычисления использовались два свойства операции *xor* (существование инверсии и существование нулевого кода).

Показанные выше уравнения доказывают, что хотя смеситель имеет неконвертируемый элемент, сам смеситель является самоконвертируемым.

Пример 3.12. Пусть имеются исходные и зашифрованные данные, каждый длиной *4* бита, и ключ длиной *3* бита. Предположим, что функция *f(K)* извлекает первый и третий биты ключа, интерпретирует биты как десятичный номер, находит квадрат этого числа и интерпретирует результат как *4*-битовую двоичную последовательность. Необходимо показать результаты зашифрования и расшифрования данных, если первоначальные исходные данные – *0111*, и ключ – *101*.

Решение. Функция *f(K)* извлекает первый и третий бит ключа. В результате получается *11* в двоичном виде или *3* в десятичном отображении. Результат возведения во вторую степень (квадрат) – *9*, в двоичном отображении *1001*.

Зашифрование: $C = M \oplus f(K) = 0111 \oplus 1001 = 1110$.

Расшифрование: $M = C \oplus f(K) = 1110 \oplus 1001 = 0111$.

Как видно, результат расшифрования совпадет с исходными данными *M*.

Функция $f(101) = 1001$ является неконвертируемой, но операция *xor* позволяет использовать функцию *f(K)* и в алгоритмах зашифрования и расшифрования. Другими словами, функция *f(K)* является неконвертируемой, но смеситель будет самоконвертируемым.

Чтобы приблизиться к шифру *Фейстеля*, необходимо усовершенствовать шифр, представленный на рис. 3.17. Известно, что для этого надо использовать вход к неконвертируемому элементу (функции), но при этом необходимо использовать не только ключ.

Задействуем также вход к функции *f(K),* чтобы применить ее для зашифрования части исходных данных и расшифрования части зашифрованных данных. Ключ может использоваться как второй вход к функции. Благодаря этому способу функция *f(K)* становится сложным элементом с некоторыми неключевыми элементами и некоторыми ключевыми элементами. Чтобы достичь цели, разделим исходные и зашифрованные данные на два блока равной длины – левый (*L*) и правый (*R*). Правый блок вводится в функцию *f(K)* и будет $f(R_1,K)$, а левый блок складывается с помощью операции *xor* с выходом функции $f(R_1,K)$. Необходимо запомнить, что входы к функции $f(R_1,K)$ должны точно совпадать при зашифровании и расшифровании. Это означает, что правая секция R_1 исходных данных до зашифрования и правая секция R_1 зашифрованных данных после расшифрования будут совпадать. Другими словами, секция R_1 должна войти в зашифрование и выйти из расшифрования неизмененной. Рис. 3.18 иллюстрирует эту идею.

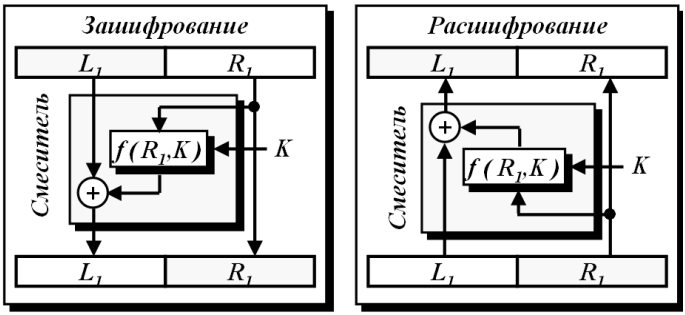


Рис. 3.18. Усовершенствование предыдущей схемы *Фейстеля*

Алгоритмы зашифрования и расшифрования инверсны друг другу. Предположим, что $L_3 = L_2$ и $R_3 = R_2$ (в зашифрованных данных в течение передачи не произошло изменений). Тогда

$$R_4 = R_3 = R_2 = R_1,$$

$$L_4 = L_3 \oplus f(R_3,K) = L_2 \oplus f(R_2,K) = L_1 \oplus f(R_1,K) \oplus f(R_1,K) = L_1.$$

Исходные данные, используемые в алгоритме зашифрования, – это данные, правильно восстановленные алгоритмом расшифрования.

Предыдущее усовершенствование имеет один недостаток: правая половина исходных данных никогда не изменяется. Злоумыш-

ленник может немедленно найти правую половину исходных данных, разбивая на части зашифрованные данные и распаковывая его правую половину. Проект нуждается в дальнейших шагах усовершенствования: *во-первых*, необходимо увеличить число раундов; *во–вторых*, необходимо добавить новый элемент в каждый раунд – *устройство замены*. Эффект устройства замены в раунде зашифрования компенсируется эффектом устройства замены в раунде расшифрования. Однако это позволяет менять левые и правые половины в каждом раунде. Рис. 3.19 иллюстрирует новый вариант шифра *Фейстеля* с двумя раундами.

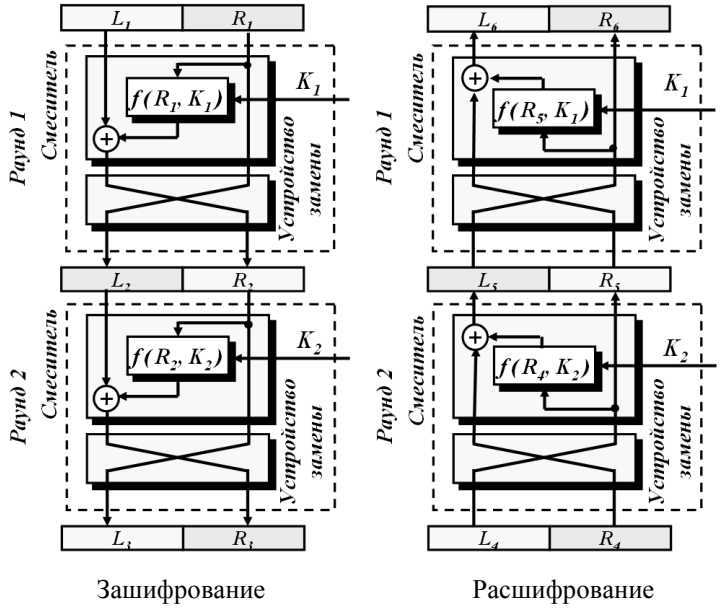


Рис. 3.19. Окончательный вариант шифра *Фейстеля* с двумя раундами

Обратите внимание, что есть два ключа раундов: k_1 и k_2. Ключи используются в обратном порядке при зашифровании и расшифровании.

Поскольку два смесителя и устройства замены инверсны друг другу, очевидно, что зашифрование и расшифрование также инверсны друг другу. Можно доказать этот факт, используя отношения между левыми и правыми секциями в каждом шифре. Другими словами, если $L_6 = L_1$ и $R_6 = R_1$, предположим, что $L_4 = L_3$ и $R_4 = R_3$

(зашифрованные данные не изменились при передаче). Вначале докажем это для промежуточных данных:

$$L_5 = R_4 \oplus f(L_4, K_2) = R_3 \oplus f(R_2, K_2) =$$

$$= L_2 \oplus f(R_2, K_2) \oplus f(R_2, K_2) = L_2,$$

$$R_5 = L_4 = L_3 = R_2.$$

Тогда просто доказать равенство для двух блоков исходных данных:

$$L_6 = R_5 \oplus f(L_5,K_1) = R_2 \oplus f(L_2,K_1) =$$

$$= L_1 \oplus f(R_1,K_1) \oplus f(R_1,K_1) = L_1,$$

$$R_6 = L_5 = L_2 = R_1.$$

Шифры не-*Фейстеля* используют только обратимые компоненты. Компонент в исходных данных имеет соответствующий компонент в шифре. Например, *S*-блоки замены должны иметь равное число входов и выходов, чтобы быть совместимыми (обратимыми). Не позволяется в шифрах не-*Фейстеля* никакое сжатие или расширение *P*-блоков, потому что они станут необратимыми. В шифрах не-*Фейстеля* нет потребности, делить исходный текст на две половины.

Рис. 3.19 можно рассматривать как графическую иллюстрацию принципа шифра не-*Фейстеля*, потому что единственные компоненты в каждом раунде – самообратимые операции *xor*, *S*-блоки 2×2, которые могут построены так, чтобы быть обратимыми, и прямые *P*-блоки, которые обратимы, если использована соответствующая таблица перестановки. Поскольку каждый компонент является обратимым, то можно показать, что и каждый раунд является обратимым. Необходимо только применять ключи раундов в обратном порядке. Зашифрование использует ключи раундов k_1 и k_2. Алгоритм расшифрования должен пользоваться ключами раундов k_2 и k_1.

3.3. АТАКИ НА БЛОЧНЫЕ ШИФРЫ

Атаки традиционных шифров могут также использоваться для современных блочных шифров, но сегодняшние блочные шифры успешно противостоят большинству атак. Например, атака “грубой силы” ключа, как правило, неосуществима, потому что ключи обычно имеют очень большую длину. Однако недавно были изоб-

ретены некоторые новые виды атак блочных шифров, которые основаны на структуре современных блочных шифров. Эти атаки используют методы и дифференциального и линейного криптографического анализа.

3.3.1. Дифференциальный криптографический анализ

Идею относительно *дифференциального криптографического анализа* предложили *Эли Бихам* и *Ади Шамир*. Это – атака с выборкой исходных данных. Злоумышленник может каким-либо образом получить доступ к компьютеру отправителя и завладеть выборочно частью исходных данных и соответствующим им зашифрованных данных. Цель состоит в том, чтобы найти ключ шифра отправителя.

Алгоритм анализа. Перед тем как злоумышленник предпримет атаку с выборкой исходных данных, он должен проанализировать алгоритм зашифрования, чтобы собрать некоторую информацию об отношениях зашифрованных и исходных данных. Очевидно, злоумышленник не знает ключ шифра. Однако некоторые шифры имеют слабости в структурах, которые могут позволить злоумышленнику найти различия исходных данных и различия зашифрованных данных, не зная ключ.

Пример 3.13. Предположим, что шифр состоит только из одной операции *xor*, как показано на рис. 3.20.

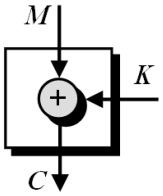


Рис. 3.20. Диаграмма для примера 3.13

Не зная значения ключа, злоумышленник может легко найти отношения между разностями исходных и разностями зашифрованных данных.

Если разность исходных данных обозначить $M_1 \oplus M_2$, а разность зашифрованных данных мы обозначим $C_1 \oplus C_2$, то приведенные следующие преобразования доказывают, что $C_1 \oplus C_2 = M_1 \oplus M_2$:

$$C_1 = M_1 \oplus K;\ C_2 = M_2 \oplus K;$$

Однако этот пример нереалистичен; модемные блочные шифры не настолько просты.

Пример 3.14. В примере 3.13 необходимо добавить один *S*-блок замены, как показано на рис. 3.21.

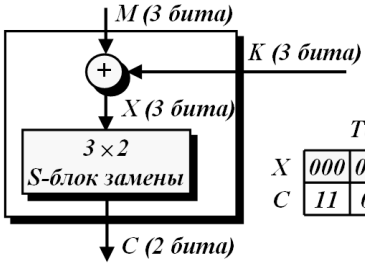


Таблица S-блока замены

X	000	001	010	011	100	101	110	111
C	11	00	10	10	01	00	11	00

Рис. 3.21. Диаграмма для примера 3.14

Хотя эффект зашифрования ключом не действует, когда используются разности между двумя X и двумя $P(X_1 \oplus X_2 = M_1 \oplus M_2)$, существование S-блока замены мешает злоумышленнику найти и определенные отношения между разностями исходных данных и разностями зашифрованных данных. Однако возможно установить вероятностные отношения.

Злоумышленник может составить таблицу (см. табл. 3.5), которая показывает для разности исходных данных, сколько можно создать разностей зашифрованных данных – шифр.

Таблица 3.5

Дифференциальная таблица входов и выходов для шифра в примере 3.14

		$C_1 \oplus C_2$			
		00	01	10	11
$M_1 \oplus M_2$	000	8			
	001	2	2		4
	010	2	2	4	
	011		4	2	2
	100	2	2	4	
	101		4	2	2
	110	4		2	2
	111			2	6

Обратите внимание, что таблица сделана по информации, которая произведена с учетом таблицы входа-выхода S-блока замены по рис. 3.21, потому что $M_1 \oplus M_2 = X_1 \oplus X_2$.

Поскольку размер ключа – 3 бита, может быть восемь случаев для каждой разности во вводе. Таблица показывает, что если вход-

ная разность – $(000)_2$, разность выхода – всегда $(00)_2$. С другой стороны, таблица показывает, что если входная разность – $(100)_2$, то имеется два случая разностей выхода $(00)_2$, два случая разностей выхода $(01)_2$ и четыре случая разностей выхода $(10)_2$.

Пример 3.15. Эвристический результат примера 3.14 может создать вероятностную информацию для злоумышленника, как показано в табл. 3.6. Входы в таблице соответствуют вероятностям появления. Разности с нулевой вероятностью никогда не будут возникать.

Как будет отмечено позже, злоумышленник теперь располагает достаточным количеством информации, чтобы начать атаку. Табл. 3.6 показывает, что вероятности распределены неоднородно из-за слабости в структуре *S*-блока замены. Табл. 3.6 упоминается иногда как дифференциальная таблица распределения или профайл *xor*.

Таблица 3.6
Дифференциальная таблица входов и выходов для шифра в примере 3.14

	00	*01*	*10*	*11*
000	*1*	*0*	*0*	*0*
001	*0,25*	*0,25*	*0*	*0,50*
010	*0,25*	*0,25*	*0,50*	*0*
011	*0*	*0,50*	*0,25*	*0,25*
100	*0,25*	*0,25*	*0,50*	*0*
101	*0*	*0,50*	*0,25*	*0,25*
110	*0,50*	*0*	*0,25*	*0,25*
111	*0*	*0*	*0,25*	*0,75*

Запуск атаки на выбранные исходные данные. После того как эвристический анализ однажды сделан, он может быть сохранен для будущего использования, пока структура шифра не изменится. Злоумышленник может выбрать для атак исходные данные. Дифференциальная таблица распределения вероятности (табл. 3.6) поможет злоумышленнику их выбирать – он возьмет те, которые имеют самую высокую вероятность в таблице.

Предположительное значение ключа. После запуска некоторых атак с соответствующей выборкой исходных данных злоумышленник может найти некоторую пару “исходные данные/зашифрованные данные”, которая позволяет ему предположить некоторое значение ключа. Процесс начинается от *C* и продвигается к *M*.

Пример 3.16. Рассматривая табл. 3.6, злоумышленник знает, что если $M_1 \oplus M_2 = 001$, то $C_1 \oplus C_2 = 11$ с вероятностью *0,5*. Он пробует взять $C_1 = 00$ и получает $M_1 = 010$ (атака с выборкой зашифрованных данных). Он еще пробует $C_2 = 11$ и получает $M_2 = 011$ (другая атака с выборкой зашифрованных данных). Теперь злоумышленник

пробует вернуться к анализу, основанному на первой паре, M_1 и C_1 (см. рис. 3.21):

$$C_1 = 00 \rightarrow X_1 = 001 \text{ или } X_1 = 111.$$

Если $X_1 = 001$, то $K = X_1 \oplus M_1 = 011$.
Если $X_1 = 111$, то $K = X_1 \oplus M_1 = 101$.
Используя пару M_2 и C_2, получим

$$C_2 = 11 \rightarrow X_2 = 000 \text{ або } X_1 = 110.$$

Если $X_2 = 000$, то $K = X_2 \oplus M_2 = 011$.
Если $X_2 = 110$, то $K = X_2 \oplus M_2 = 101$.

Два испытания показывают, что $K = 011$ или $K = 101$. Хотя злоумышленник не уверен, какое из них точное значение ключа, он знает, что самый правый бит – *1* (общий бит между двумя значениями). Продолжая атаку, можно учитывать, что самый правый бит в ключе – *1*. Таким образом, можно определить другие биты в этом ключе.

Общая процедура. Современные блочные шифры имеют большую сложность, чем, та, которая обсуждалась в этом разделе. Кроме того, они могут содержать различное количество раундов. Злоумышленник может использовать следующую стратегию.

1. Поскольку каждый раунд содержит одни и те же операции, злоумышленник может создать таблицу дифференциальных распределений (профайл *xor*) для каждого *S*-блока замены и комбинировать их, чтобы создать распределение для каждого раунда.

2. Предположим, что каждый раунд независим (справедливое предположение). Злоумышленник может создать таблицу распределения для всего шифра, умножая соответствующие вероятности.

3. Злоумышленник может теперь делать список исходных данных для атак, основанных на таблице распределений, созданной на втором шаге. Заметим, что таблица, созданная при выполнении шага 2, помогает злоумышленнику выбирать только меньшее количество пар “исходные данные/зашифрованные данные”.

4. Злоумышленник выбирает зашифрованные данные и находит соответствующие исходные данные. Затем он анализирует результат, чтобы найти некоторые биты в ключе.

5. Злоумышленник повторяет действие шага 4, чтобы найти больше битов в ключе.

6. После нахождения достаточного количества битов в ключе злоумышленник может использовать атаку "*грубой силы*", чтобы найти весь ключ.

Дифференциальный криптографический анализ базируется на таблице неоднородных дифференциальных распределений, *S*-блоков замены в блочном шифре.

Более детально дифференциальный криптографический анализ приводится в разд. 5.

3.3.2. Линейный криптографический анализ

Линейный криптоанализ был представлен *Митцури Мацуи* (*Mitsuru Matsui*) в 1993 году. Анализ использует атаки на известные исходные данные (в отличие от атак на выбранные исходные данные в дифференциальном криптографическом анализе). Полное обсуждение этой атаки базируется на некоторых понятиях теории вероятностей. Чтобы рассмотреть главную идею этой атаки, предположим, что шифр состоит из одного раунда, как показано на рис. 3.22, где c_0, c_1 и c_2 представляют три бита на выходе и x_0, x_1 и x_2 – три бита на входе *S*-блока подстановки (замены).

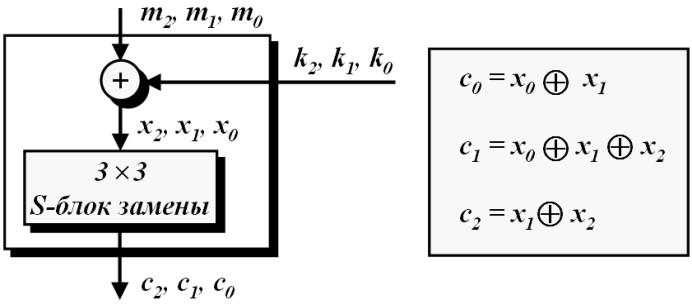


Рис. 3.22. Простой шифр с линейным *S*-блоком подстановки (замены)

S-блок подстановки (замены) – линейное преобразование, в котором каждый вывод является линейной функцией ввода, как было обсуждено выше. С этим линейным компонентом можно создать три линейных уравнения между исходными данными и битами зашифрованных данных, как будет показано ниже:

$$c_0 = m_0 \oplus k_0 \oplus m_1 \oplus k_1;$$

$$c_1 = m_0 \oplus k_0 \oplus m_1 \oplus k_1 \oplus m_2 \oplus k_2;$$

$$c_2 = m_1 \oplus k_1 \oplus m_2 \oplus k_2.$$

Решая систему уравнений для трех неизвестных, получаем

$$k_1 = (m_1) \oplus (c_0 \oplus c_1 \oplus c_2);$$

$$k_2 = (m_2) \oplus (c_0 \oplus c_1);$$

$$k_0 = (m_0) \oplus (c_1 \oplus c_2).$$

Это означает, что три атаки на известные исходные данные могут найти значения k_1 и k_2. Однако реальные блочные шифры не так просты, как этот, они имеют больше компонентов, и *S*-блоки замены не линейны.

Линейная аппроксимация. В некоторых современных блочных шифрах может случиться, что некоторые *S*-блоки замены не полностью на нелинейные; тогда они могут быть в вероятностном смысле аппроксимированы некоторыми линейными функциями. Вообще, задавая исходные и зашифрованные данные в *n* бит и ключ *m* бит, ищутся некоторые уравнения, имеющие вид

$$(k_0 \oplus k_1 \oplus \ldots \oplus k_x) = (m_0 \oplus m_1 \oplus \ldots \oplus m_y) \oplus$$

$$\oplus (c_0 \oplus c_1 \oplus \ldots \oplus c_z).$$

Контрольные вопросы и задания

1. Укажите различия между современными и традиционными шифрами с симметричным ключом.
2. Объясните, почему современные блочные шифры спроектированы как шифры подстановки вместо того, чтобы применять шифры транспозиции.
3. Перечислите компоненты современного блочного шифра.
4. Определите *P*-блок и назовите его три варианта. Какой вариант является обратным?
5. Определите *S*-блок и покажите необходимое условие обратных *S*-блоков.
6. Определите составной шифр и назовите два класса составных шифров.
7. Поясните различие между рассеянием и перемешиванием.

8. Поясните различие между блочным шифром *Фейстеля* и не-*Фейстеля*.

9. Какая разница между линейным и дифференциальным криптографическим анализом. Какой криптографический анализ использует атаку на выбранные исходные данные, а какой – атаку на известные исходные данные?

10. Сообщение имеет *1500* символов для представления, которых используются *ASCII* коды. Это сообщение будет зашифровано блочным шифром длиной *64* бита. Найдите размер дополнения и количество зашифрованных блоков.

11. Блок транспозиции имеет *4* входа и *4* выхода. Каков порядок группы перестановки и размер ключевой последовательности в битах?

12. Блок подстановки имеет *4* входа и *4* выхода. Каков порядок группы перестановки и размер ключевой последовательности в битах?

13. Используя слово 10011011_2, покажите результат циркулярного (циклического) сдвига влево на *3* бита.

14. Используя слово 10011011_2, покажите результат циркулярного (циклического) сдвига вправо на *3* бита.

15. Найдите результат таких операций:

а) $(01001101)_2 \oplus (01001101)_2$; б) $(01001101)_2 \oplus (10110010)_2$;

в) $(01001101)_2 \oplus (00000000)_2$; г) $(01001101)_2 \oplus (11111111)_2$.

16. Совершите перестановку битов для прямого *P*-блока, показанного на рис. 3.4, если на его входе действует последовательность: $(10110)_2$.

17. Совершите перестановку битов для *P*-блока сжатия, показанного на рис. 3.4, если на его входе действует последовательность: $(10110)_2$.

18. Совершите перестановку битов для *P*-блока расширения, показаного на рис. 3.4, если на его входе действует последовательность: $(101)_2$.

19. Покажите *P*-блок, определен таблицей:

8	*1*	*2*	*3*	*4*	*5*	*6*	*7*

К какому типу блоков принадлежит показанный *P*-блок: прямой; сжатия или расширения?

20. Определите, является ли *P*-блок с таблицей перестановки:

1	*1*	*2*	*3*	*4*	*4*

прямым *P*-блоком, *P*-блоком сжатия или *P*-блоком расширения?

21. Определите, является ли *P*-блок с таблицей перестановки:

1	*3*	*5*	*6*	*7*

прямым *P*-блоком, *P*-блоком сжатия или *P*-блоком расширения?

22. Определите, является ли *P*-блок с таблицей перестановки:

7	*3*	*1*	*4*	*8*	*5*	*2*	*6*

прямым *P*-блоком, *P*-блоком сжатия или *P*-блоком расширения?

23. Отношение вход-выход *2×2 S*-блока показаны в следующей таблице

		Вход: правый бит	
		0	*1*
Вход: левый бит	*0*	*01*	*11*
	1	*00*	*10*

Покажите таблицу для инверсного блока.

24. *S*-блок подстановки производит операцию *xor* с нечетными битами, чтобы получить левый бит выхода, и *xor* с четными битами, чтобы получить правый бит выхода. Если вход – $(110010)_2$, то что является выходом? Если вход – $(101101)_2$, то что является выходом?

25. Крайний левый бит *S*-блока подстановки розміром *4×3* определяет смещение других трех бит. Если крайний левый бит равен *0*, то три других бита перемещаются вправо на один бит. Если крайний левый бит – *1*, три других бита перемещаются влево на один бит. Если вход – *1011*, то какой результат будет на выходе? Если вход – *0110*, то какой результат будет на выходе?

Раздел 4

ПОТОКОВЫЕ ШИФРЫ И ГЕНЕРАТОРЫ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ

4.1. ОБЩИЕ СВЕДЕНИЯ О ПОТОКОВЫХ ШИФРАХ

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

Работа типичного потокового шифра представлена на рис. 4.1.

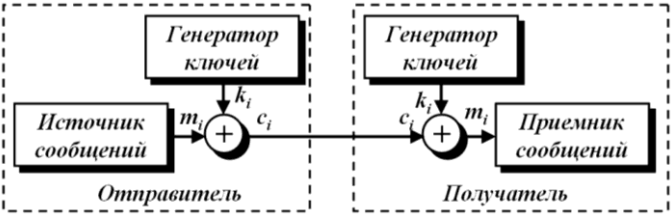


Рис. 4.1. Принцип работы потокового шифра

Генератор ключей выдает поток бит k_i, которые будут использоваться в качестве гаммы. Источник сообщений генерирует биты открытого текста m_i, которые складываются по модулю *2* с гаммой, в результате чего получаются биты зашифрованного сообщения c_i:

$$c_i = m_i \oplus k_i, \; i = 1, 2, \ldots, n.$$

Чтобы из зашифрованного сообщения $c_1, c_2, ..., c_n$ восстановить сообщение $m_1, m_2, ..., m_n$, необходимо сгенерировать точно такую же ключевую последовательность $k_1, k_k, ..., k_n$, что и при зашифровании, и использовать для расшифрования формулу

$$m_i = c_i \oplus k_i,\ i = 1, 2, \ldots, n,$$

так как операция поразрядного сложения по модулю *2* обладает свойством обратимости.

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

Пример 4.1. Пусть известно, что исходное сообщение представляло собой двоично-десятичное число, то есть число, каждая тетрада (четыре бита) которого получена при переводе десятичной цифры *0...9* в двоичный вид. Перехвачено *24* бита зашифрованного сообщения *M*, то есть шесть тетрад m_1, m_2, m_3, m_4, m_5 и m_6, а именно значение $C = 1100\ 1101\ 1110\ 1111\ 0000\ 0001$. Известно, что ключ зашифрования состоял из четырех бит, которые тоже представляют собой однозначное десятичное число, то есть одно и тоже значение $0 \le K \le 9$ использовалось для зашифрования каждых четырех бит исходного сообщения. Таким образом, зашифрование числа $m_1, m_2,$

m_3, m_4, m_5 и m_6 ключом K можно представить в виде системы уравнений:

$$m_1 \oplus K = 1100; \quad m_2 \oplus K = 1101; \quad m_3 \oplus K = 1110;$$

$$m_4 \oplus K = 1111; \quad m_5 \oplus K = 0000; \quad m_6 \oplus K = 0001.$$

Исходя из условия, что m_i принимает десятичные значения от *0* до *9*, для поиска неизвестного K определим все возможные значения m'_1 и K, сумма которых по модулю *2* приводит к результату *1100*:

$$\begin{array}{ll} & K = \ 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111\ 1000\ 1001 \\ \oplus & c_1 = \ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100\ 1100 \\ \hline & m'_1 = \ 1100\ 1101\ 1110\ 1111\ 1000\ 1001\ 1010\ 1011\ 0100\ 0101 \end{array}.$$

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

Для дальнейшего поиска неизвестного K определим все возможные значения m_2' и оставшихся вариантов ключа, сумма которых по модулю *2* приводит к результату $c_2 = 1101$:

$$\begin{array}{ll} & K = \ 0100\ 0101\ 1000\ 1001 \\ \oplus & c_2 = \ 1101\ 1101\ 1101\ 1101 \\ \hline & m'_2 = 1001\ 1000\ 0101\ 0100 \end{array}.$$

Видно, что этот этап не позволил отбросить ни одного из оставшихся вариантов ключа. Попытаемся это сделать, используя c_3=*1110*:

$$\begin{array}{ll} & K = \ 0100\,0101\,1000\,1001 \\ \oplus & C_3 = 1110\,1110\,1110\,1110 \\ \hline & m'_3 = 1010\,1011\,0110\,0111 \end{array}.$$

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

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

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

4.2. ПРИНЦИПЫ ИСПОЛЬЗОВАНИЯ ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ЧИСЕЛ ПРИ ПОТОКОВОМ ШИФРОВАНИИ

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

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

сообщения можно будет только с использованием того же ключа, который использовался при зашифровании.

Для использования в криптографических целях *ГПСЧ* должен обладать следующими свойствами:

- период последовательности должен быть очень большой;
- порождаемая последовательность должна быть "*почти*" неотличима от действительно случайной;
- вероятности появления (порождения) различных значений должны быть в точности равны;
- для того, чтобы только законный получатель мог расшифровать сообщение, следует при получении потока ключевых бит k_i использовать и учитывать некоторый секретный ключ, причем вычисление числа k_{i+1} по известным предыдущим элементам последовательности k_i без знания ключа должно быть трудной задачей.

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

4.2.1. Линейный конгруэнтный генератор псевдослучайных чисел

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

$$k_i = (a*k_{i-1}+b) \bmod c,$$

где a, b, c – некоторые константы, а k_{i-1} – предыдущее псевдослучайное число.

Для получения k_1 задается начальное значение k_0. Возьмем в качестве примера $a = 5$, $b = 3$, $c = 11$ и пусть $k_0 = 1$. В этом случае можно по приведенной выше формуле получать значения от *0* до *10* (так как $c = 11$). Вычислим несколько элементов последовательности:

$$k_1 = (5 * 1 + 3) \bmod 11 = 8; \quad k_2 = (5 * 8 + 3) \bmod 11 = 10;$$

$$k_3 = (5 * 10 + 3) \bmod 11 = 9; \quad k_4 = (5 * 9 + 3) \bmod 11 = 4;$$

$$k_5 = (5 * 4 + 3) \bmod 11 = 1.$$

Полученные значения (*8, 10, 9, 4, 1*) выглядят похожими на случайные числа. Однако следующее значение k_6 будет снова равно *8*:

$$k_6 = (5 * 1 + 3) \bmod 11 = 8,$$

а значения k_7 и k_8 будут равны *10* и *9* соответственно:

$$k_7 = (5 * 8 + 3) \bmod 11 = 10; \quad k_8 = (5 * 10 + 3) \bmod 11 = 9.$$

Выходит, *ГПСЧ* формирует повторяющуюся последовательность чисел, порождая периодически числа *8, 10, 9, 4, 1*. К сожалению, это свойство характерно для всех линейных конгруэнтных генераторов. Изменяя значения основных параметров *a, b* и *c*, можно влиять на длину периода и на сами порождаемые значения k_i. Так, например, увеличение числа *c* в общем случае ведет к увеличению периода. Если параметры *a, b* и *c* выбраны правильно, то генератор будет порождать случайные числа с максимальным периодом, равным *c*. При программной реализации значение *c* обычно устанавливается равным 2^{b-1} или 2^b, где *b* – длина слова *электронной вычислительной машины* (*ЭВМ*) или *автоматизированной системы* (*АС*) в битах.

Достоинством линейных конгруэнтных *ГПСЧ* является их простота и высокая скорость получения псевдослучайных значений. Линейные конгруэнтные генераторы находят применение при решении задач моделирования и математической статистики, однако в криптографических целях их нельзя рекомендовать к использованию, так как специалисты по криптографическому анализу научились восстанавливать всю последовательность *псевдослучайных чисел* (*ПСЧ*) по нескольким значениям. Например, предположим, что злоумышленник может определить значения k_0, k_1, k_2, k_3. Тогда:

$$k_1 = (a*k_0+b) \bmod c; \quad k_2 = (a*k_1+b) \bmod c; \quad k_3 = (a*k_2+b) \bmod c.$$

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

Для получения *ПСЧ* предлагалось использовать также квадратичные и кубические генераторы:

$$k_i = (a_1^2*k_{i-1}+a_2*k_{i-1}+b) \bmod c;$$

$$k_i = (a_1^3*k_{i-1}+a_2^2*k_{i-1}+a_3*k_{i-1}+b) \bmod c.$$

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

4.2.2. Генератор псевдослучайных чисел на основе метода Фибоначчи с запаздыванием

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

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

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

$$k_i = \begin{cases} k_{i-a} - k_{i-b}, & \text{если} \quad k_{i-a} \geq k_{i-b}; \\ k_{i-a} - k_{i-b} + 1, & \text{если} \quad k_{i\ a} < k_{i\ b}, \end{cases}$$

где k_i – вещественные числа из диапазона *[0,1]*, *a, b* – целые положительные числа, параметры генератора.

Для работы датчику Фибоначчи требуется знать *max {a,b}* предыдущих сгенерированных случайных чисел. При программной реализации для хранения сгенерированных случайных чисел необходим некоторый объем памяти, зависящих от параметров *a* и *b*.

Пример 4.2. Вычислим последовательность из первых десяти чисел, генерируемую методом Фибоначчи с запаздыванием начиная с k_5 при следующих исходных данных: $a = 4$, $b = 1$, $k_0 = 0{,}1$; $k_1 = 0{,}7$; $k_2 = 0{,}3$; $k_3 = 0{,}9$; $k_4 = 0{,}5$:

$$k_5 = k_1 - k_4 = 0{,}7 - 0{,}5 = 0{,}2;$$

$$k_6 = k_2 - k_5 = 0{,}3 - 0{,}2 = 0{,}1;$$

$$k_7 = k_3 - k_6 = 0{,}9 - 0{,}1 = 0{,}8;$$

$$k_8 = k_4 - k_7 + 1 = 0{,}5 - 0{,}8 + 1 = 0{,}7;$$

$$k_9 = k_5 - k_8 + 1 = 0{,}2 - 0{,}7 + 1 = 0{,}5;$$

$$k_{10} = k_6 - k_9 + 1 = 0{,}1 - 0{,}5 + 1 = 0{,}6;$$

$$k_{11} = k_7 - k_{10} = 0{,}8 - 0{,}6 = 0{,}2;$$

$$k_{12} = k_8 - k_{11} = 0{,}7 - 0{,}2 = 0{,}5;$$

$$k_{13} = k_9 - k_{12} + 1 = 0{,}5 - 0{,}5 + 1 = 1;$$

$$k_{14} = k_{10} - k_{13} + 1 = 0{,}6 - 1 + 1 = 0{,}6,$$

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

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

В результате значения (*a, b*) = (*17, 5*) рекомендуются для простых приложений. Значения (*a, b*) = (*55, 24*) позволяют получать числа, удовлетворительные для большинства криптографических алгоритмов, требовательных к качеству случайных чисел. Значения (*a, b*) = (*97, 33*) позволяют получать очень качественные случайные числа и используются в алгоритмах, работающих со случайными векторами высокой размерности [32].

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

4.2.3. Генератор псевдослучайных чисел на основе алгоритма BBS

Широкое распространение получил алгоритм генерации *псевдослучайной последовательности* (*ПСП*), называемый *алгоритмом BBS* (от фамилий авторов – *L. Blum, M. Blum, M. Shub*) или *генератором с квадратичным остатком*. Для целей криптографии этот метод предложен в 1986 году [3, 32].

Сущность этого алгоритма заключается в следующем. Вначале выбираются два больших простых числа p и q. Числа p и q должны быть оба *сравнимы* с *3* по модулю *4*, то есть при делении p и q на *4* должен получаться одинаковый остаток *3*. Далее вычисляется число $M = p \cdot q$, называемое *целым числом Блюма*. Затем выбирается другое случайное целое число x, взаимно простое (то есть не имеющее общих делителей, кроме единицы) с M. Вычисляется $x_0 = x^2 \bmod M$, где x_0 называется *стартовым числом генератора*.

На каждом n-м шаге работы генератора вычисляется

$$x_{n+1} = x_n^2 \bmod M.$$

Результатом n-го шага является один (обычно младший) бит числа x_{n+1}. Иногда в качестве результата принимают бит чётности, то есть количество единиц в двоичном представлении элемента. Если количество единиц в записи числа четное – бит четности принимается равным *0*, нечетное – бит четности принимается равным *1*.

Пример 4.2. Пусть $p = 11$, $q = 19$ (убеждаемся, что $11 \bmod 4 = 3$, $19 \bmod 4 = 3$). Тогда $M = p \cdot q = 11 \cdot 19 = 209$. Выберем x, взаимно простое с M: пусть $x = 3$. Вычислим стартовое число генератора x_0:

$$x_0 = x^2 \bmod M = 3^2 \bmod 209 = 9 \bmod 209 = 9.$$

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

$x_1 = 9^2 \bmod 209 = 81 \bmod 209 = 81$ – младший бит: *1*;

$x_2 = 81^2 \bmod 209 = 6561 \bmod 209 = 82$ – младший бит: *0*;

$x_3 = 82^2 \bmod 209 = 6724 \bmod 209 = 36$ – младший бит: *0*;

$x_4 = 36^2 \bmod 209 = 1296 \bmod 209 = 42$ – младший бит: *0*;

$x_5 = 42^2 \bmod 209 = 1764 \bmod 209 = 92$ – младший бит: *0;*

$x_6 = 92^2 \bmod 209 = 8464 \bmod 209 = 104$ – младший бит: *0;*

$x_7 = 104^2 \bmod 209 = 10816 \bmod 209 = 157$ – младший бит: *1*;

$x_8 = 157^2 \bmod 209 = 24649 \bmod 209 = 196$ – младший бит: *0*;

$x_9 = 196^2 \bmod 209 = 38416 \bmod 209 = 169$ – младший бит: *1*;

$x_{10} = 169^2 \bmod 209 = 28561 \bmod 209 = 137$ – младший бит: *1*.

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

$$x_n = x_0^{2^{n \bmod ((p-1)(q-1))}} \bmod M .$$

Например, вычислим x_{10} сразу из x_0:

$$x_{10} = x_0^{2^{10 \bmod ((11-1)(19-1))}} \bmod 209 = 9^{1024 \bmod 180} \bmod 209 = 137.$$

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

Возможность "*прямого*" получения x_n позволяет использовать алгоритм *BBS* при потоковом шифровании, например, для файлов с произвольным доступом или фрагментов файлов с записями базы данных.

Безопасность алгоритма *BBS* основана на сложности разложения большого числа *M* на множители. Утверждается, что если *M* достаточно велико, его можно даже не держать в секрете; до тех пор, пока *M* не разложено на множители, никто не сможет предсказать выход *ГПСЧ*. Это связано с тем, что задача разложения чисел вида $n = p \cdot q$ (p и q – простые числа) на множители является вычислительно очень трудной, если известно только *n*, а *p* и *q* – большие

числа, состоящие из нескольких десятков или сотен бит (это так называемая *задача факторизации*).

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

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

Этот алгоритм выдает действительно хорошую *ПСП* с большим периодом (при соответствующем выборе исходных параметров), что позволяет использовать его для криптографических целей при генерации ключей для шифрования.

4.2.4. Генераторы псевдослучайных чисел на основе сдвиговых регистров с обратной связью

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

Сдвиговые регистры с обратной связью могут применяться для получения потока псевдослучайных бит. Сдвиговый регистр с обратной связью состоит из двух частей: собственно *n*-битного сдвигового регистра и устройства обратной связи (рис. 4.2).

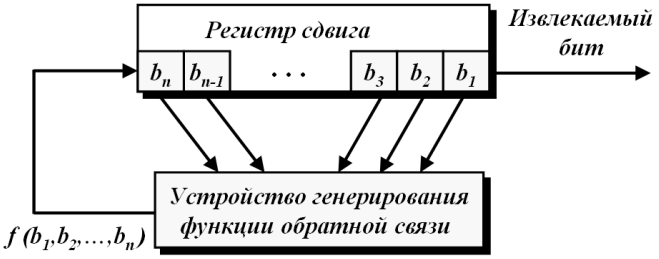


Рис. 4.2. Сдвиговый регистр с обратной связью

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

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

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

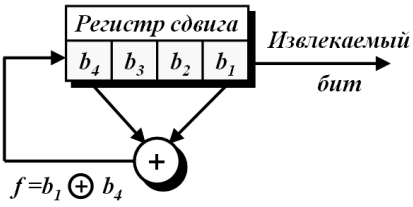


Рис. 4.3. Пример *4*-разрядного линейного регистра сдвига

Для того, чтобы стало понятнее, как работает регистр сдвига с обратной связью, рассмотрим *4*-битовый *LFSR* с отводом от первого и четвертого разрядов, представленный на рис. 4.3.

Запишем в изображенный на рис. 4.3 регистр начальное значение, которого *1011*. Вычислять последовательность внутренних состояний регистра удобно с помощью таблицы, представленной на табл. 4.1. В таблице отражены первые девять состояний регистра.

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

Линейный сдвиговый регистр размером *n* бит может находиться в одном из $2^n–1$ состояний. Поэтому теоретически такой регистр может генерировать псевдослучайную последовательность с максимальным периодом $2^n–1$.

Таблица 4.1

Последовательность работы линейного регистра сдвига

Номер состояния	Внутреннее состояние регистра b_4, b_3, b_2, b_1	Результат вычисления функции обратной связи $f = b_1 \oplus b_4$	Извлекаемый бит (b_1)
0	*1 0 1 1*	*0*	*1*
1	*0 1 0 1*	*1*	*1*
2	*1 0 1 0*	*1*	*0*
3	*1 1 0 1*	*0*	*1*
4	*0 1 1 0*	*0*	*0*
5	*0 0 1 1*	*1*	*1*
6	*1 0 0 1*	*0*	*1*
7	*0 1 0 0*	*0*	*0*
8	*0 0 1 0*	*0*	*0*

Линейный регистр сдвига с обратной связью будет генерировать циклическую последовательность бит с максимальным периодом только при выборе в качестве отводной последовательности определенных бит. Разработана математическая теория, позволяющая выбрать подходящие номера разрядов для бит отводной последовательности [3, 8, 9].

Линейные сдвиговые регистры сдвига с обратной связью часто использовались и используются до сих пор при шифровании потоков данных. Для повышения криптографической стойкости в таких устройствах шифрования применяются комбинации нескольких регистров сдвига с обратной связью, а также вводятся дополнительные перемешивающие операции. Такие электронные схемы предлагались и выпускались еще до Второй мировой войны. Аналогичные принципы заложены и в некоторые потоковые шифры, созданные в конце XX века, например, в алгоритм *A5*, использовавшийся в Европе для шифрования сотовых цифровых каналов связи стандарта *GSM*. Несмотря на то, что некоторые криптографические аналитики высказывают сомнения в надежности алгоритмов потокового

шифрования с использованием линейных регистров сдвига с обратной связью, они положены в основу функционирования различных военных и гражданских устройств связи, используемых до настоящего времени [32].

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

4.3. КЛАССИФИКАЦИЯ ПОТОКОВЫХ ШИФРОВ

Допустим, например, что в режиме гаммирования для потоковых шифров при передаче по каналу связи произошло искажение одного знака зашифрованного сообщения. Очевидно, что в этом случае все знаки, принятые без искажения, будут расшифрованы правильно. Произойдёт потеря лишь одного знака сообщения. А теперь представим, что один из знаков зашифрованного сообщения при передаче по каналу связи был потерян. Это приведёт к неправильному расшифрованию всего сообщения, следующего за потерянным знаком. Практически во всех каналах передачи данных для потоковых систем шифрования присутствуют помехи. Поэтому для предотвращения потери информации решают проблему синхронизации зашифрования и расшифрования сообщения. По способу решения этой проблемы криптографические системы подразделяются на синхронные системы и системы с самосинхронизацией.

4.3.1. Синхронные потоковые шифры

Синхронные потоковые шифры (*СПШ*) – это шифры, в которых поток ключей генерируется независимо от открытого и зашифрованного сообщения.

При зашифровании генератор потока ключей выдаёт биты потока ключей, которые идентичны битам потока ключей при расшифровании. Потеря знака зашифрованного сообщения приведёт к нарушению синхронизации между этими двумя генераторами и невозможности расшифрования оставшейся части сообщения. Очевидно, что в этой ситуации отправитель и получатель должны повторно синхронизоваться для продолжения работы.

Обычно синхронизация производится вставкой в передаваемое сообщение специальных маркеров. В результате этого пропущенный при передаче знак приводит к неверному расшифрованию лишь до тех пор, пока не будет принят один из маркеров.

Заметим, что выполняться синхронизация должна так, чтобы ни одна часть потока ключей не была повторена. Поэтому переводить генератор в более раннее состояние не имеет смысла.

Основными преимуществами *СПШ* являются:

- отсутствие эффекта распространения ошибок (только искажённый бит будет расшифрован неверно);
- предохраняет от любых вставок и удалений зашифрованного сообщения, так как они приведут к потере синхронизации и будут обнаружены.

Недостатками *СПШ* являются: уязвимые к изменениям отдельные биты зашифрованного сообщения. Если злоумышленнику известно открытое сообщение, он может изменить эти биты так, чтобы они расшифровывались, как ему надо.

4.3.2. Самосинхронизирующиеся потоковые шифры

Самосинхронизирующиеся потоковые шифры (*асинхронные потоковые шифры* (*АПШ*)) – это шифры, в которых поток ключей создаётся функцией ключа и фиксированного числа знаков зашифрованного сообщения.

Итак, внутреннее состояние генератора потока ключей является функцией предыдущих *n* бит зашифрованного сообщения. Поэтому генератор потока ключей, который производит расшифрование, приняв *n* бит, автоматически синхронизируется с генератором потока ключей, который производит зашифрование.

Реализация этого режима происходит следующим образом: каждое сообщение начинается случайным заголовком длиной *n* бит; заголовок зашифровывается, передаётся и расшифровывается; расшифрование будет неправильным, зато после этих *n* бит оба генератора будут синхронизированы.

Основными преимуществами АПШ является размешивание статистики открытого сообщения. Так как каждый знак открытого сообщения влияет на следующее зашифрованное сообщение, статистические свойства открытого сообщения распространяются на все зашифрованное сообщение. Следовательно, *АПШ* может быть более

устойчивым к атакам на основе избыточности открытого сообщения, чем *СПШ*.

Недостатками АПШ являются:

- распространение ошибки (каждому неправильному биту зашифрованного сообщения соответствуют *n* ошибок в открытом сообщении);
- чувствительны к вскрытию повторной передачей.

4.4. ПОТОКОВЫЙ ШИФР А5

4.4.1. История создания потокового шифра А5

А5 – это потоковый алгоритм шифрования, используемый для обеспечения конфиденциальности передаваемых данных между телефоном и базовой станцией в европейской системе мобильной цифровой связи *GSM* (*Group Special Mobile*).

Шифр основан на побитовом сложении по модулю *2* (булева операция *xor*) генерируемой псевдослучайной последовательности и информации, которая зашифровывается. В *А5* псевдослучайная последовательность реализуется на основе трёх линейных регистров сдвига с обратной связью. Регистры имеют количество разрядов (длины) *19, 22* и *23* бита соответственно. Сдвигами управляет специальная схема, организующая на каждом шаге смещение как минимум двух регистров, что приводит к неравномерному движению содержания их последовательностей. Последовательность формируется путём операции *xor* над выходными битами регистров.

Изначально французские военные специалисты – криптографы разработали потоковый шифр для использования исключительно в военных целях. В конце 80-х г. XX века для стандарта *GSM* потребовалось создание новой, современной системы безопасности. В её основу легли три секретных алгоритма: аутентификации – *А3*; шифрования потока – *А5*, генерации ключа сеанса – *А8*. В качестве алгоритма *А5*, была использована французская разработка. Этот шифр обеспечивал достаточно хорошую защищённость потока, что обеспечивало конфиденциальность разговора. Изначально экспорт стандарта из Европы не предполагался, но вскоре в этом появилась необходимость. Именно поэтому, *А5* переименовали в *А5/1* и стали распространять в Европе и США. Для остальных стран алгоритм модифицировали, значительно понизив криптографическую стойкость шифра. *А5/2* был специально разработан как экспортный ва-

риант для стран, не входивших в Европейский союз. В *A5/0* шифрование отсутствует совсем. В настоящее время разработан также алгоритм *A5/3*, основанный на *алгоритме Касуми* и утверждённый для использования в сетях *3G*. Эти модификации обозначают *A5/x*.

Официально данная криптографическая схема не публиковалась и её структура не предавалась гласности. Это связанно с тем, что разработчики полагались на безопасность за счёт неизвестности, то есть алгоритмы труднее взломать, если они не доступны публично. Данные предоставлялись операторам *GSM* только по необходимости. Тем не менее, к 1994 году детали алгоритма *A5* стали известны: британская телефонная компания (*British Telecom*) передала всю документацию, касающуюся стандарта, Брэдфордскому университету для анализа, не заключив соглашение о неразглашении информации. Кроме того, материалы о стандарте появились на одной конференции в Китае. В результате, его схема постепенно "просочилась" в широкие круги. В этом же году кембриджские учёные *Росс Андерсон* (*Ross Anderson*) и *Майкл Рое* (*Michael Roe*) опубликовали восстановленную по этим данным криптографическую схему и дали оценку её криптографической стойкости. Окончательно алгоритм был представлен в работе *Йована Голича* на конференции *Eurocrypt'97* [3, 7].

4.4.2. Потоковое шифрование данных с помощью A5

Алгоритм *A5* в настоящее время – это целое семейство шифров. Для описания возьмем *A5/1* (рис. 4.4) как родоначальника этого семейства. Изменения в производных алгоритмах опишем отдельно.

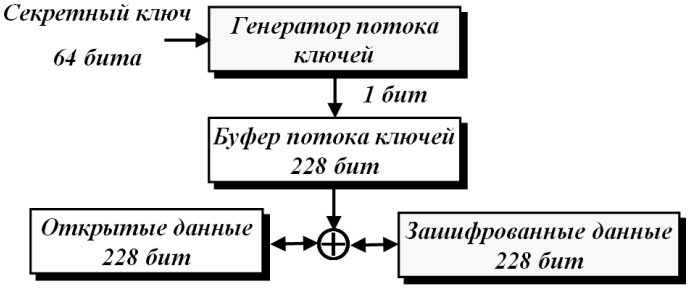


Рис. 4.4. Принцип построения алгоритма потокового шифрования *A5/1*

A5/1 используется в Глобальной системе мобильной связи (*GSM*). Телефонная связь в *GSM* осуществляется как последовательность кадров на *228* бит, причем каждый кадр – *4,6* миллисекунды. *A5/1* создает поток бит, исходя из ключа длиной *64* бита. Разрядные потоки собраны в буферы по *228* бит, чтобы складывать их по модулю *2* с кадром на *228* бит, как показано на рис. 4.4.

На рис. 4.4 стрелки в обе стороны показывают, что во время зашифрования принимают открытые данные, а после суммирования с потоком ключей по модулю *2* выходят зашифрованные данные; во время расшифрования берут зашифрованные данные, а после суммирования с потоком ключей по модулю *2* выходят открытые данные.

В этом алгоритме каждому символу открытого сообщения соответствует символ зашифрованного сообщения. Сообщение не делится на блоки (как в блочном шифровании) и не изменяется в размере. Для упрощения аппаратной реализации и, следовательно, увеличения быстродействия используются только простейшие операции: сложение по модулю *2* (*xor*) и сдвиг бит регистра.

Формирование выходной последовательности происходит путём сложения потока исходного сообщения с генерируемой последовательностью (гаммой). Особенность операции *xor* заключается в том, что применённая чётное число раз, она приводит к начальному значению. Отсюда, расшифрование сообщения происходит путём сложения зашифрованного сообщения с известной последовательностью (гаммой).

Таким образом, безопасность шифра *A5* полностью зависит от свойств последовательности. В идеальном случае каждый бит гаммы – это независимая случайная величина, и сама последовательность является случайной. Такая схема была изобретена *Вернамом* в 1917 году и названа в его честь. Как доказал *Клод Шеннон* в 1949 году, это обеспечивает абсолютную криптографическую стойкость. Но использование случайной последовательности означает передачу по защищённому каналу этой последовательности равной по объёму открытому сообщению, что значительно усложняет задачу и практически нигде не используется.

В реальных системах создаётся ключ заданного размера, который без труда передаётся по закрытому каналу. Последовательность генерируется на его основе и является псевдослучайной. Большой

класс потоковых шифров (в том числе *A5*) составляют шифры, генератор псевдослучайной последовательности который основан на регистрах сдвига с линейной обратной связью (*LFSR*).

LFSR состоит из собственно регистра (последовательности бит заданной длины) и обратной связи (рис. 4.5).

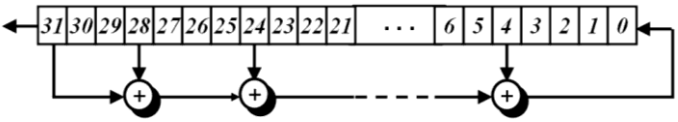


Рис. 4.5. *LFSR*, с многочленом обратной связи $x^{32}+x^{29}+x^{25}+x^5+1$

На каждом такте происходят следующие действия: крайний левый бит (старший бит) извлекается, последовательность в регистре сдвигается влево и в опустевшую правую ячейку (младший бит) записывается значение функции обратной связи. Эта функция является суммированием по модулю *2* определённых бит регистра и записывается в виде многочлена, где степень указывает номер бита. Извлечённые биты формируют выходную последовательность.

Для *LFSR* основным показателем является период псевдослучайной последовательности. Он будет максимален (и равен 2^n-1), если многочлен функции обратной связи примитивен по модулю *2*. Выходная последовательность в таком случае называется *M*-последовательностью (последовательность максимально возможной неповторяющейся длины).

Сам по себе *РСЛОС* легко поддаётся криптографическому анализу и не является достаточно надёжным, для использования в шифровании. Практическое применение имеют системы регистров переменного тактирования, с различными длинами и функциями обратной связи.

Схема потокового шифра *A5* включает в себя (рис. 4.6):

- три регистра (R_1, R_2, R_3), имеющие длину *19*, *22* и *23* разрядов соответственно, многочлены обратных связей для которых:
- для $R_1 - x^{19} + x^{18} + x^{17} + x^{14} + 1$;
- для $R_2 - x^{22} + x^{21} + 1$;
- для $R_3 - x^{23} + x^{22} + x^{21} + x^8 + 1$;
- схему управления тактированием.

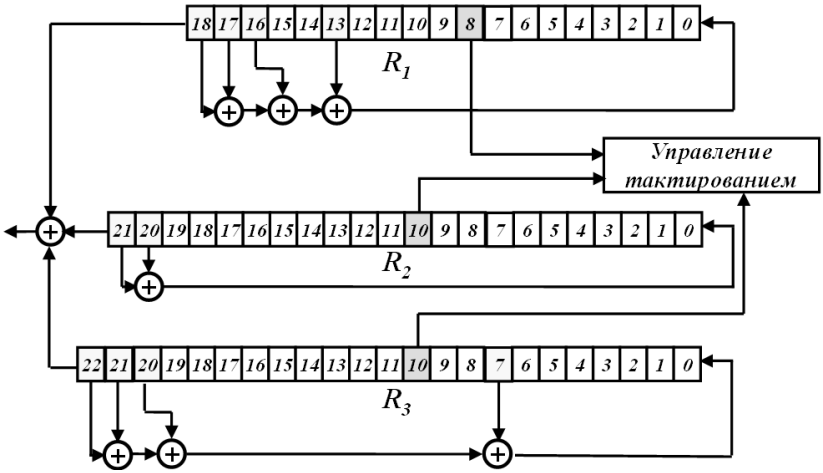


Рис. 4.6. Структура система регистров в алгоритме *A5/1*

Для управления тактированием в каждом регистре есть биты синхронизации: *8* (R_1), *10* (R_2), *10* (R_3). С использованием этих бит вычисляется функция:

$$f = x \,\&\, y \mid x \,\&\, z \mid y \,\&\, z,$$

где & – булево *and*; | – булево *or*; *x*, *y* и *z* – биты синхронизации R_1, R_2 и R_3 соответственно.

После вычисления функции *f* будут сдвинуты разряды только тех регистров, у которых бит синхронизации равен *f*. Фактически, сдвигаются биты тех регистров, синхробит которых принадлежит большинству.

Выходным битом системы является результат операции *xor* над выходными битами регистров.

Рассмотрим особенности функционирования алгоритма, на основе известной схемы (рис. 4.6). Передача данных осуществляется в структурированном виде – с разбивкой на кадры (*114* бит). Перед инициализацией регистры обнуляются, на вход алгоритма поступают: ключ сеанса (*K – 64* бита), сформированный алгоритмом *A8*, и номер кадра (*Fn – 22* бита). Далее последовательно выполняются следующие действия: однобитовый выход обеспечивает тактовыми импульсами буфер на *228* бит, который используется для зашифрования или расшифрования.

Инициализация. Инициализация выполняется для каждого кадра зашифрования или расшифрования и использует ключ сеанса на *64* бита и *22* бита соответствующего номера кадра. Во время инициализации выполняются следующие шаги:

1. Изначально все биты в трех линейных регистрах сдвига устанавливаются в *0* (обнуляются).

2. Ключ на *64* бита смешивается со значением регистра согласно с последующим псевдокодом (каждый линейный регистр сдвигается на один шаг, то есть обеспечивается синхронизация):

```
for (i = 0 to 63)
{
    Сложение по модулю 2 ключа K[i] с крайними левыми битами
    всех трех регистров.
    Синхронизация всех трех линейных регистров сдвига
}
```

3. Повторить предыдущий процесс, но использовать *22*-битовый кадр согласно такому псевдокоду:

```
for (i = 0 to 21)
{
    Сложение по модулю 2 номера кадра (i) с крайними левыми
    битами всех трех регистров.
    Синхронизация всех трех линейных регистров сдвига
    }
```

4. В течение *100* циклов синхронизируется весь генератор согласно с таким псевдокодом, при этом используется *мажоритарная функция* для того чтобы определить, какой линейный регистр сдвига должен быть синхронизирован:

```
for (i = 0 to 99)
{
    Синхронизация всего генератора на основе
    мажоритарной функции
}
```

Заметим, что иногда синхронизация здесь означает, что два, а то и все три линейных регистры сдвига сдвигаются.

Мажоритарная функция. Значение мажоритарной функции (*f*) с параметрами (*x, y, z*) равно *1*, если значение большинства бит – *1*;

если это *0*, то ее значение – *0*. Например, *f(1,0,1) = 1*, но *f(0,0,1) = 0*. Значение мажоритарной функции определяется перед поступлением тактового импульса; три входные бита названы синхронизирующими битами: если крайний правый бит равен нулю, это – биты линейных регистров $R_1[10]$, $R_2[11]$ и $R_3[11]$. Необходимо обратить внимание на то, что в литературе эти биты $R_1[8]$, $R_2[10]$ и $R_3[10]$ отчисляют слева (как это показано на рис. 4.6). Будем рассматривать биты линейных регистров $R_1[10]$, $R_2[11]$ и $R_3[11]$ справа. Это условие соответствует месту бита в характеристическом полиноме.

Ключевые биты потока. Генератор ключей создает ключевой поток в один бит по каждому тактовому импульсу. Прежде чем ключ будет создан, вычисляется мажоритарная функция. Затем каждый линейный регистр сдвига синхронизируется, если его бит синхронизации соответствует результату мажоритарной функции; иначе – он не синхронизируется.

Пример 4.4. В некоторый момент времени биты синхронизации $R_1[10]$, $R_2[11]$ и $R_3[11]$ равны *1*, *0* и *1* соответственно. Каким должно быть содержание регистров сдвига?

Решение: Результат мажоритарной функции: *f(1,0,1) = 1*. Следовательно, содержание регистров R_1 и R_3 сдвигаются, а R_2 – нет.

Зашифрование/расшифрование. Разрядные потоки, созданные генератором ключей, записываются в буфер, чтобы в дальнейшем сформировать ключ на *228* бит, который затем складывается по модулю *2* с кадром исходных данных, чтобы создать кадр зашифрованных данных. Одновременно осуществляется зашифрование/расшифрование одного кадра.

К алгоритму *A5/2* добавлен еще один регистр на *17* бит (R_4), который управляет движением (тактированием) других (рис. 4.7).

Изменения структуры следующие:

- добавлен регистр R_4 длиной *17* бит с многочленом обратной связи для $R_4 – x^{17} + x^{10} + 1$;

- управление тактированием осуществляет регистр R_4, биты которого: *3, 7, 10* являются битами синхронизации;

- вычисляемая мажоритарная функция:

$$f = x \,\&\, y \mid x \,\&\, z \mid y \,\&\, z,$$

где & – булево *and*; | - булево *or*; *x*, *y* и *z* – биты синхронизации R_4 (*3*-й бит), R_4 (*7*-й бит) и R_4 (*10*-й бит) соответственно.

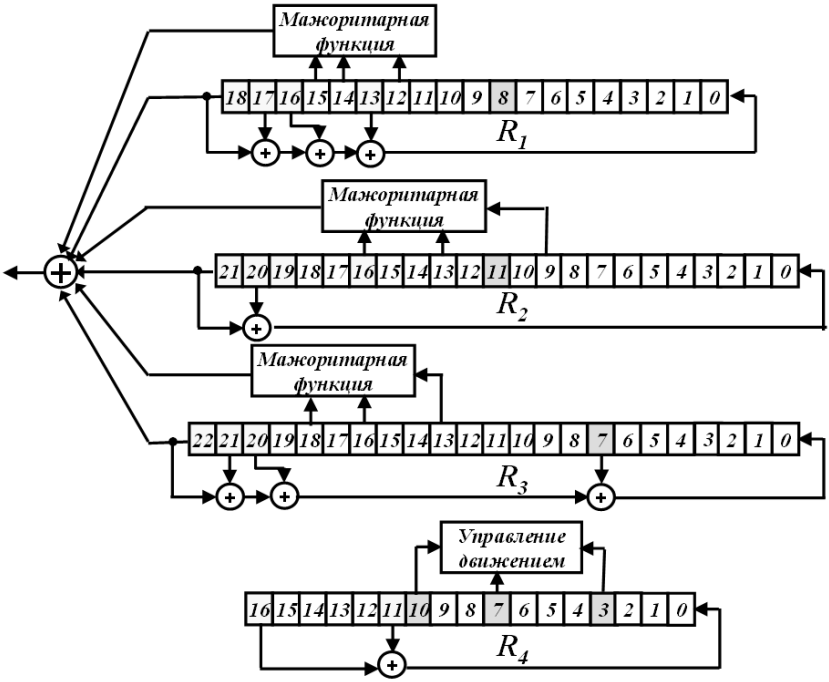


Рис. 4.7. Структура системы регистров в алгоритме *A5/2*

Сдвиги бит в регистрах осуществляются в таком случае:

- биты регистра R_1 сдвигаются, если $R_4\ (10) = f$;
- биты регистра R_2 сдвигаются, если $R_4\ (3) = f$;
- биты регистра R_3 сдвигаются, если $R_4\ (7) = f$.

Фактически, сдвигаются биты тех регистров, синхробит которых принадлежит большинству.

Выходным битом системы является результат операции *xor* над старшими битами регистров и мажоритарных функций от определённых бит регистров:

- регистр R_1 – биты: *12*, *14* и *15*;
- регистр R_2 – биты: *9*, *13* и *16*;
- регистр R_3 – биты: *13*, *16* и *18*.

Изменения в функционировании не такие существенные и касаются только инициализации:

- *64 + 22* такта заполняется ключом сеанса и номером кадра также R_4;

- один такт: $R_4(3)$, $R_4(7)$ и $R_4(10)$ заполняются единицами;

- *99* тактов с управлением сдвигами регистров, но без генерации последовательности.

Видно, что инициализация занимает такое же время (*100* тактов без генерации разбиты на две части).

Алгоритм *A5/3* разработан в 2001 году и должен сменить *A5/1* в третьем поколении мобильных систем. Также он называется алгоритм *Касуми*. При его создании за основы взят шифр *MISTY*, корпорации *Mitsubishi*. В настоящее время считается, что *A5/3* обеспечивает требуемую стойкость.

Алгоритм *A5/0* не содержит шифрования.

4.4.3. Криптографическая стойкость потокового шифра A5

Разработка стандарта *GSM* подразумевала мощный аппарат шифрования, не поддающийся взлому (особенно в реальном времени). Используемые разработки при надлежащей реализации обеспечивали качественное шифрование передаваемых данных. Именно такую информацию можно получить от компаний распространяющих этот стандарт. Но стоит отметить важный нюанс: прослушивание разговоров – неотъемлемый атрибут, используемый спецслужбами. Они были заинтересованы в возможности прослушивания телефонных разговоров для своих целей. Поэтому в алгоритм были внесены изменения, дающие возможность взлома за приемлемое время. Помимо этого, для экспорта *A5* модифицировали в *A5/2*. В *MoU* (*Memorandum of Understand Group Special Mobile standard*) признают, что целью разработки *A5/2* было понижение криптографической стойкости шифрования, однако в официальных результатах тестирования говорится, что неизвестно о каких-либо недостатках алгоритма [32].

С появлением данных о стандарте *A5*, начались попытки взлома алгоритма, а также поиска уязвимостей. Огромную роль оказали особенности стандарта, резко ослабляющие защиту, а именно:

- *10* бит ключа принудительно занулены;

- отсутствие перекрестных связей между регистрами (кроме управления сдвигами);

- излишняя избыточность шифруемой служебной информации, известной криптографическому аналитику;

- свыше 40% ключей приводит к минимальной длине периода генерируемой последовательности, а именно $(2^{23}–1)\cdot 3/4$ [32];

- вначале сеанса осуществляется обмен нулевыми сообщениями (по одному кадру);

- в *A5/2* движение осуществляется отдельным регистром длиной *10* бит.

На основе этих "*дыр*" в алгоритме, построены схемы взлома.

Ключом является сессионный ключ длиной *64* бита, номер кадра считается известным. Таким образом, сложность атаки основанной на прямом переборе равна 2^{64}.

Первые обзоры шифра (работа *Росса Андерсона*) сразу выявили уязвимость алгоритма – из-за уменьшения эффективной длины ключа (зануление *10* бит) сложность упала до 2^{45} (сразу на *6* порядков). *Атака Андерсона* основана на предположении о начальном заполнении коротких регистров и по выходным данным получения заполнения третьего.

В 1997 году *Йован Голич* опубликовал результаты анализа *A4*. Он предложил способ определения первоначального заполнения регистров по известному отрезку гаммы длиной всего *64* бита. Этот отрезок получают из нулевых сообщений. Атака имеет среднюю сложность 2^{40} [32].

В 1999 году *Вагнеру* и *Голдбергу* без труда удалось продемонстрировать, что для вскрытия системы, достаточно перебором определить начальное заполнение R_4. Проверка осуществляется за счёт нулевых кадров. Сложность этой атаки равна 2^{17}, таким образом, на современном компьютере вскрытие шифра занимает несколько секунд.

В декабре 1999 года группа израильских учёных (*Ади Шамир, Алекс Бирюков*, а позже и американец *Дэвид Вагнер* (англ.)) опубликовали весьма нетривиальный, но теоретически очень эффективный метод вскрытия *A5/1*.

4.5. ПОТОКОВЫЙ ШИФР RC4

4.5.1. История создания шифра RC4

Алгоритм *RC4* разработан *Р. Ривестом* в 1987 р. специально как генератор потока ключевой информации с ключом переменной длины. Хотя официальное сокращение – *Rivest Cipher 4*, его часто считают сокращением от *Ron's Code* [3, 19].

Шифр был коммерческой тайной, но в сентябре 1994 г. его описание было анонимно отправлено в рассылку *Cypherpunks* [3, 19]. Вскоре описание *RC4* было опубликовано в ньюс-группе *sci.crypt*. Именно оттуда входной код попал на многие сайты в сети *Интернет*. Опубликованный шифр давал те же зашифрованные данные на выходе, которые давал настоящий *RC4*. Наверное, эти данные были получены в результате анализа исполняемого кода. Опубликованный шифр совместим с имеющимися продуктами, которые используют *RC4*, а некоторые участники телеконференции, которые имели по их словам, доступ до входного кода *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.

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* бит, но соглашение, заключённое между *ассоциацией издателей программного обеспечения* (*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.

4.5.2. Описание алгоритма RC4

RC4 базируется на понятии *матрицы состояний*. В каждый момент матрица состояний (*256* байт) активизируется, из нее случайно выбирается один байт, который будет ключом для шифрования. Идея может быть показана в виде массива байтов:

S[0], S[1], S[2], ..., S[254], S[255].

Отметим, что индексы диапазона элементов – между *0* и *255*. Содержание каждого элемента – байт (*8* бит), который может интерпретироваться как целое число от *0* до *255*.

Рис. 4.8 показывает принцип построения потокового шифра *RC4*.

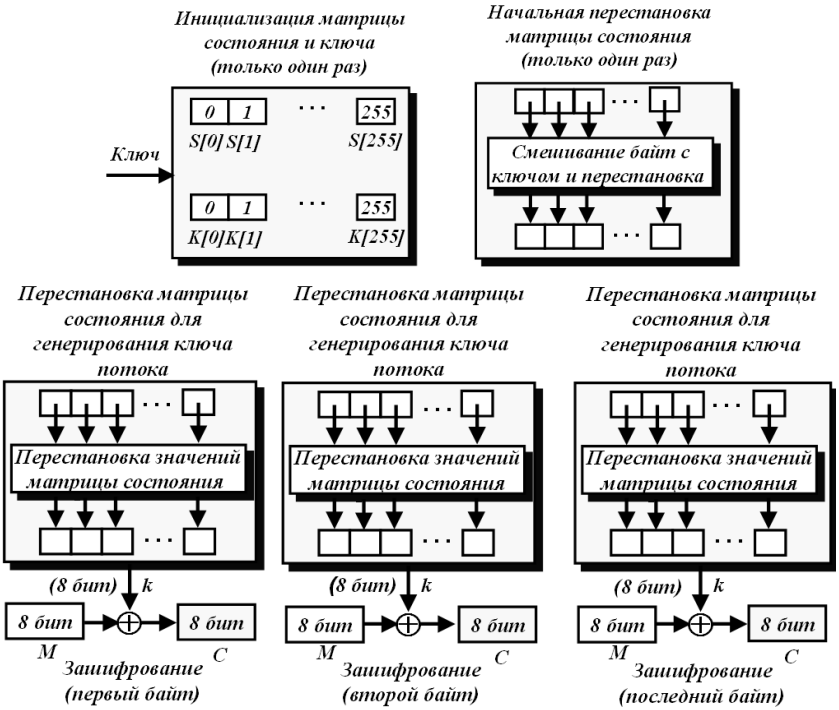


Рис. 4.8. Принцип построения потокового шифра *RC4*

Первые два блока выполняются только один раз (инициализация); перестановки для того, чтобы создавать ключ потока, повторяются, пока есть байты входных данных, предназначенные для шифрования.

Алгоритм *RC4* включает в себя два этапа. На первом, подготовительном, этапе производится инициализация таблицы замен *S* (матрицы состояния), матрицы ключа *K*, а также начальная перестановка матрицы состояния, основанная на значении байтов в *K[i]*. На втором, основном, этапе вычисляются непосредственно псевдослучайные числа *k*.

Инициализация матрицы состояния и массива ключей

Ключ в *RC4* представляет собой последовательность байтов произвольной длины, по которой строится исходное состояние шифра *S* – перестановка всех *256* байтов.

Алгоритм инициализации *RC4*, который также называют алгоритмом ключевого расписания (англ. *Key-Scheduling Algorithm or KSA*), приведены на рис. 4.9.

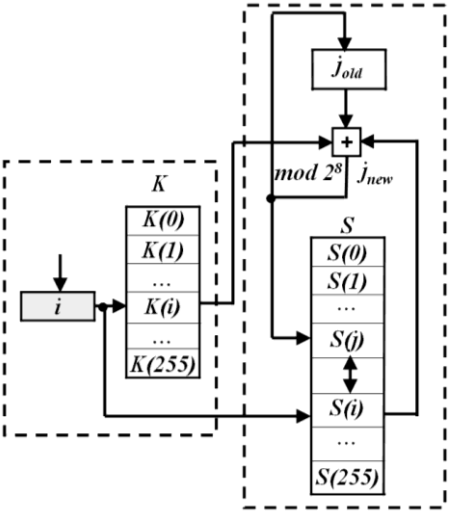


Рис. 4.9. Схема инициализации матрицы состояния *S* и массива ключа *K*

Этот алгоритм использует ключ, сохраненный в *Key*, и имеет длину *L* байт. Инициализация начинается с заполнения массива *S*, далее этот массив перемешивается путем перестановок, определенных ключом. Поскольку одно действие выполняется над *S*, то должно выполняться утверждение, что *S* всегда содержит все значения кодового слова.

Матрица состояния инициализируется для значений *0, 1, 2, ..., 254, 255*. Создается также массив ключей *K[0], K[1], K[2], ..., K[254], K[255]*. Если ключ шифрования имеет точно *256* байтов, байты копируются в массив *K*; иначе – байты повторяются, пока не заполнится массив *K*.

Первоначально матрица состояния *S* заполняется последовательными значениями от *0* до *255*. Затем каждый очередной элемент *S* обменивается местами с элементом, номер которого определяется

элементом ключа *K*, самим элементом и суммой номеров элементов, с которыми происходил обмен на предыдущих итерациях. Значения счетчиков *i* и *j* изначально равны *0*.

Ниже показана программа на псевдокоде, которая инициализирует матрицу состояний *S* и массив ключей *K*:

```
for ( i = 0 to 255 )
{
    S[i] ← i
    K[i] ← Key[ i mod LengthKey ]
}
```

Далее в матрице состояний происходит перестановка (скремблирование элементов), основанная на значении байтов в *K[i]*. Ключевой байт используется только на этом шаге, чтобы определить, какие элементы нужно заменить. После этого шага байта матрицы полностью перетасованы.

Ниже показана программа на псевдокоде, которая осуществляет перестановку байтов матрицы состояний.

```
j ← 0
for ( i = 0 to 255 )
{
    j ← ( j + S[i] + K[i] ) mod 256
    swap ( S[i], S[j] )
}
```

Перестановка матрицы состояния и генерация ключевого потока

Ключи *k* в ключевом потоке генерируются один за другим. Сначала элементы матрицы состояний переставляются на основе значений своих элементов и значений двух индивидуальных переменных *i* и *j*. Затем значения двух элементов матрицы состояний в позициях *i* и *j* используются, чтобы определить индекс элемента матрицы состояний, который служит в качестве ключа *k*. Следующий код повторяется для каждого байта начальных данных текста, чтобы создать новый ключевой элемент в ключевом потоке.

Ядро алгоритма состоит из функции генерации ключевого потока (рис. 4.10).

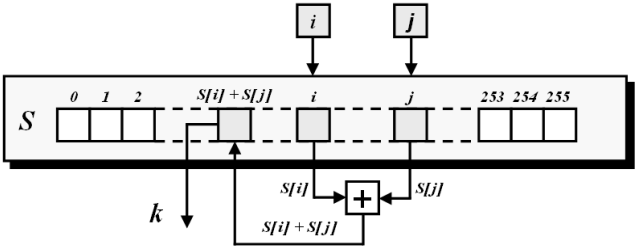


Рис. 4.10. Ядро алгоритму *RC4*

Таблица замен (матрица состояний *S*) медленно меняется во время использования, при этом счетчик *i* (переменная *i*) обеспечивает изменение каждого элемента таблицы, а счетчик *j* (переменная *j*) гарантирует, что элементы таблицы изменяются случайно.

Переменные *i* и *j* инициализирующий в *0* прежде, чем будет проведена первая итерация, но значение копируется от одной итерации к следующей.

Ниже показана программа на псевдокоде, которая осуществляет перестановку байтов матрицы состояний *S[i]* и генерирует ключевой поток *k*:

```
i ← 0;          j ← 0
i ← ( i + 1 ) mod 256
j ← ( j + S[i] ) mod 256
swap ( S[i], S[j] )
k ← S[ (S[i] + S[j]) mod 256 ]
```

Зашифрование (расшифрование) данных

После того как ключевой поток *k* был создан, байт открытых данных зашифровывается с помощью *k*, чтобы создать байт зашифрованных данных. Расшифрование представляет собой обратный процесс.

Ниже показана программа на псевдокоде, которая показывает процесс шифрования данных для *RC4*:

```
RC4_Encryption ( Key )
{
    // Создание начальной матрицы состояний
    // и ключевых байтов
    for ( i = 0 to 255 )
```

```
    {
        S[i] ← i;          K[i] ← Key[ i mod LengthKey ]
    }
    // Перестановка байт матрицы состояния на
    // основе значений байта ключа
    j ← 0
    for ( i = 0 to 255 )
    {
        j ← ( j + S[i] + K[i] ) mod 256;     swap ( S[i], S[j] )
    }
    // Непрерывная перестановка байтов матрицы состояний,
    // генерация ключевого потока и шифрования данных
    i ← 0;   j ← 0
    while (пока есть байты данных для шифрования)
    {
        i ← ( i + 1 ) mod 256;     j ← ( j + S[i] ) mod 256
        swap ( S[i], S[j] );          k ← S[ (S[i] + S[j]) mod 256 ]
        // Ключ готовий, зашифрование даних
        input M
        C ← M ⊕ k
        output C
    }
}
```

Расшифрование данных заключается в регенерации ключевого потока (*k*) и сложение его и зашифрованных данных (*C*) по модулю *2*. Благодаря свойствам суммирования по модулю *2* на выходе получим исходные данные (*M*).

Пример 4.5. Чтобы показать случайность ключа потока, используем ключ засекречивания со всеми нулевыми байтами. Ключевой поток для *20* значений в таком случае будет равна: *129*, *163*, *68*, *228*, *89*, *71*, *76*, *51*, *165*, *213*, *200*, *18*, *6*, *239*, *229*, *236*, *63*, *52*, *170*, *117*.

Пример 4.6. Повторим пример 4.5, но пусть ключ засекречивания будет пять байтов (*15*, *202*, *33*, *6*, *8*). Ключевой поток: *154*, *188*, *65*, *159*, *144*, *167*, *200*, *166*, *202*, *243*, *119*, *75*, *104*, *161*, *206*, *24*, *34*, *216*, *114*, *89*. Снова случайность в ключевом потоке очевидна.

Анализ процессов преобразования данных в алгоритме RC4

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

Пример 4.7. Пусть начальное значение ключа $Key = \{12, 2, 3, 8\}$. Показать процесс инициализации матрицы состояний и массива ключей, а также исходную перестановку матрицы состояний.

Решение. На рис. 4.11 показан пример инициализации *4*-разрядных матрицы состояний и массива ключей и исходную перестановку матрицы состояний (*9* тактов из *16*).

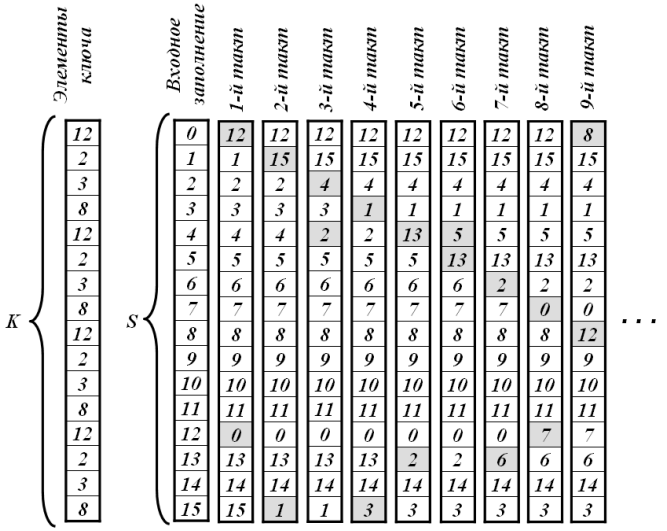


Рис. 4.11. Последовательность тактов перестановки матрицы состояний

Пример 4.8. Показать пример работы *4*-разрядного генератора *ПСЧ RC4* при заданных значениях *i* и *j* и заполнении *4*-разрядной таблицы замен S.

Решение. Пример работы *4*-разрядного генератора *ПСЧ RC4* изображен на рис. 4.12.

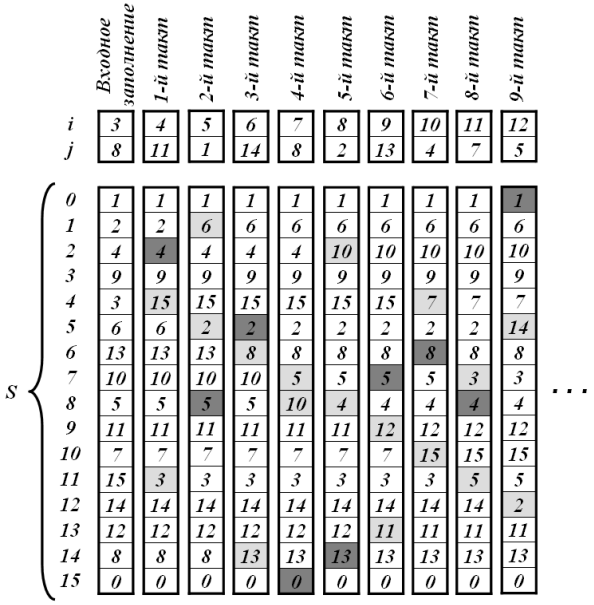


Рис. 4.12. Пример работы *4*-разрядного генератора *ПСЧ RC4*

На выходе генератора *ПСЧ RC4* формируется *4*-разрядная последовательность: *4, 5, 2, 0, 13, 5, 8, 4, 1*, … .

В рассмотренном примере размер *n* слова или блока алгоритма принимался равным четырем. Это значение можно брать и другим, например, *8* или *16*. В случае использования $n = 8$ таблица замен *S* должна состоять из $2^8 = 256$ значений, а элементами таблицы замен должны быть числа от *0* до *255*. Размер счетчиков *i* и *j* должен также изменить до восьми бит (максимальное значение – *255*).

Кроме того, все вычисления в случае $n = 8$ необходимо выполнять по модулю *256*. Аналогичные изменения в алгоритме необходимо производить и при других значениях параметра *n*.

4.5.3. Криптографическая стойкость потокового шифра RC4

Алгоритм *RC4* тщательно изучался криптографическими аналитиками. В нем не обнаружено каких бы то ни было слабых мест. Помимо высокой устойчивости к криптографическому анализу,

этот алгоритм очень быстр и может использоваться для генерации ключевой последовательности при потоковом шифровании.

Все методы криптографического анализа потоковых шифров обычно подразделяют на три класса:

1. *Силовые* (*атака* "*грубой силой*"). Атаки путём полного перебора (перебор всех возможных вариантов). Сложность полного перебора зависит от количества всех возможных решений задачи (размера пространства ключей или пространства открытых данных). Этот вид атаки применим ко всем видам систем потокового шифрования. При разработке систем шифрования разработчики стремятся сделать так, чтобы этот вид атак был наиболее эффективным по сравнению с другими существующими методами взлома.

2. *Статистические*. Делятся на два подкласса:

- метод криптографического анализа статистических свойств шифровальной гаммы, направленный на изучение исходной последовательности криптографической системы (криптографический аналитик пытается установить значение следующего бита последовательности с вероятностью выше вероятности случайного выбора с помощью различных статистических тестов);

- метод криптографического анализа сложности последовательности: криптографический аналитик пытается найти способ генерировать последовательность, аналогичную гамме, но более просто реализуемым способом.

Оба метода используют принцип линейной сложности.

3. *Аналитические методы*. Этот вид атак рассматривается в предположении, что криптографическому аналитику известно описание генератора, открытые и соответствующие закрытые данные. Задача криптографического аналитика – определить используемый ключ (начальное заполнение регистров).

Контрольные вопросы и задания

1. Чем потоковый шифр отличается от блочного шифра?
2. Каким образом организуется шифрование потока данных переменной длины?
3. Какие числа называют псевдослучайными?
4. Назовите существующие *ГПСЧ*. Какие свойства должен иметь *ГПСЧ* для использования с криптографической целью?

5. Перечислите основные характеристики, достоинства и недостатки каждого из рассмотренных *ГПСЧ*.

6. Каким образом могут использоваться для получения псевдослучайных чисел регистры сдвига с обратной связью? Объясните их принцип работы.

7. В чем разница между генераторами случайных и псевдослучайных чисел?

8. Можно ли использовать генератор настоящих случайных чисел для получения гаммы при потоковом шифровании?

9. С какой криптографической целью могут быть использованы генераторы настоящих случайных чисел?

10. Определите последовательность из первых десяти чисел и период линейного конгруэнтного *ГПСЧ* для различных параметров *a*, *b* и *c* (k_0 принять равным – *0*):

а) $a = 5, b = 7, c = 17;$

б) $a = 6, b = 3$ и $c = 23$.

11. Определите последовательность из десяти чисел, генерируемой методом *Фибоначчи* с задержкой, начиная с k_a при таких исходных данных:

а) $a = 3, b = 1, k_0 = 0{,}6; k_1 = 0{,}3; k_2 = 0{,}5$;

б) $a = 4, b = 2, k_0 = 0{,}9; k_1 = 0{,}3; k_2 = 0{,}5; k_3 = 0{,}9$.

12. Значения k_0, k_1, k_2, k_3, полученные с помощью линейного конгруэнтного генератора, равны: $k_0 = 1, k_1 = 8, k_2 = 10, k_3 = 9$. Определить параметры *a*, *b* и *c ГПСЧ*.

13. Вычислить x_{11} по методу генерации *ПСЧ BBS*, если: а) $p = 19, q = 23, x = 3$; б) $p = 23, q = 31, x = 3$.

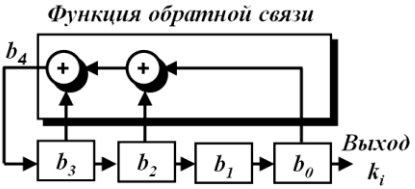
14. Вычислить псевдослучайную двоичную последовательность длиной *12* бит по методу генерации *ПСЧ BBS*, если: а) $p = 19, q = 23, x = 3$; б) $p = 23, q = 31, x = 3$. В качестве случайных бит брать младший бит в двоичной записи числа начиная с x_0.

15. Вычислить значения $x_1 – x_8$ по методу генерации *ПСЧ BBS*, если: а) $p = 19, q = 23, x = 3$; б) $p = 23, q = 31, x = 3$.

16. Для потокового алгоритма *RC4* показать первые *20* элементов ключевого потока, если ключ сеанса – *7* байтов со значениями *1*, *2*, *3*, *4*, *5*, *6* и *7*.

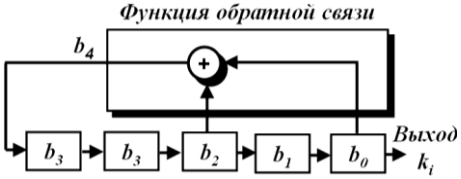
17. Покажите *LFSR* с характеристическим полиномом $x^5 + x^2 + 1$. Какой максимальный период последовательности, который формируется данным *LFSR*?

18. Определить характеристический полином представленного *LFSR*:



Какой максимальный период последовательности, формируемый данным *LFSR*?

19. Определите ключевой поток на *20* бит, который будет сгенерирован представленным *LFSR*:



если начальное значение последовательности – $(1110)_2$.

20. Максимальная длина периода *LFSR – 32* разряда. Сколько разрядов имеет регистр сдвига *LFSR*?

21. Для потокового алгоритма *A5/1* найдите максимальный период двоичной последовательности каждого линейного регистра сдвига.

22. Для потокового алгоритма *A5/1* найдите значения функций: а) *f(x, y, z) = f(1, 0, 0)*; б) *f(x, y, z) = f(0, 1, 1)*; в) *f(x, y, z) = f(0, 0, 0)*; г) *f(x, y, z) = f(1, 1, 1)*. В каждом случае показать, сколько синхронизируется линейных регистров сдвига.

Раздел 5

СТАНДАРТ БЛОЧНОГО СИММЕТРИЧНОГО ШИФРОВАНИЯ ДАННЫХ DATA ENCRIPTION STANDARD

5.1. ИСТОРИЯ РАЗРАБОТКИ АЛГОРИТМА ШИФРОВАНИЯ ДАННЫХ DES

Стандарт шифрования данных (*Data Encryption Standard – DES*) – блочный шифр с симметричными ключами, разработан Национальным Институтом Стандартов и Технологии (*National Institute of Standards and Technology – NIST*) [4, 29, 43].

В 1973 году *NIST* издал запрос для разработки предложения национальной криптографической системы с симметричными ключами.

Предложенная *IBM* модификация проекта, названная "*Люцифер*" (*Lucifer*), была принята как *DES*. *DES* опубликован в эскизном виде в Федеральном Регистре в марте 1975 года как Федеральный Стандарт Обработки Информации (*Federal Information Processing Standard – FIPS*) [4, 29, 43].

После публикации эскиз строго критиковался по двум причинам: *первая* – критиковалась сомнительно маленькая длина ключа (только *56* битов), что могло сделать шифр уязвимым к атаке "*грубой силой*"; вторая причина – критики были обеспокоены некоторым скрытым построением внутренней структуры *DES*. Они подозревали, что некоторая часть структуры (*S*-блоки замены) может иметь скрытую "лазейку", которая позволит расшифровывать сообщения без ключа. Впоследствии проектировщики *IBM* сообщили, что внутренняя структура была доработана, чтобы предотвратить криптографический анализ.

DES был наконец издан как *FIPS 46* в Федеральном Регистре в январе 1977 года. Однако *FIPS* объявил *DES* как стандарт для использования в неофициальных приложениях.

Алгоритм, положенный в основу стандарта, распространялся достаточно быстро, и уже в 1980 г. был одобрен *NIST США*. С этого момента *DES* превращается в стандарт не только по названию (*Data Encryption Standard*), но и фактически. Появляются программное обеспечение и специализированные *микроЭВМ*, предназначенные для зашифрования и расшифрования информации в сетях передачи данных.

DES – название Федерального Стандарта Обработки информации (*FIPS 46-3*), который описывает алгоритм шифрования данных (*Data Encryption Algorithm – DEA*). В терминах *ANSI DEA* определен как стандарт *X9.32*.

В документах *NIST* криптографическая система *DES* называется алгоритмом шифрования данных (*DEA*), а международная организация по стандартизации, ссылаясь на шифр *DES*, пользуется аббревиатурой *DEA-1* [4, 29, 43].

Этот алгоритм являлся мировым стандартом на протяжении более чем двадцати лет и утвердился как первый доступный всем желающим официальный алгоритм. Поэтому его стоит отметить как важнейшую веху на пути криптографии от чисто военного использования к широкомасштабному применению.

DES был наиболее широко используемым блочным шифром с симметричными ключами, начиная с его публикации. Позже *NIST* предложил новый стандарт – *FIPS 46-3*, который рекомендует использование тройного *DES* (трехкратно повторенный шифр *DES*) для будущих приложений.

Основные требования, которые были предъявлены разработчикам алгоритма *DES* следующие:

- алгоритм должен обеспечивать высокий уровень безопасности;
- алгоритм должен быть полностью определен и легко понятен;
- безопасность алгоритма должна основываться на ключе и не должна зависеть от сохранения в тайне самого алгоритма;
- алгоритм должен быть доступен всем пользователям;
- алгоритм должен позволять адаптацию к различным применениям;
- алгоритм должен позволять экономичную реализацию в виде электронных приборов;
- алгоритм должен быть эффективным в использовании;
- алгоритм должен предоставлять возможности проверки;
- алгоритм должен быть разрешен для экспорта.

Основные достоинства алгоритма *DES*:

- используется только один ключ длиной *64* (*56* бит длина ключа и *8* контрольных разрядов) бит;

- зашифровав сообщение с помощью одного пакета программ, для расшифрования можно использовать любой другой пакет программ соответствующий стандарту *DES*;

- относительная простота алгоритма обеспечивает высокую скорость обработки;

- достаточно высокая стойкость алгоритма.

Первоначально метод, лежащий в основе стандарта *DES*, был разработан фирмой *IBM* для своих целей и реализован в виде системы "*Люцифер*". Система "*Люцифер*" основана на комбинировании методов подстановки и перестановки и состоит из чередующейся последовательности блоков перестановки и подстановки. В ней использовался ключ длиной *128* бит, управлявший состояниями блоков перестановки и подстановки. Система "*Люцифер*" оказалась весьма сложной для практической реализации из-за относительно малой скорости шифрования (*2190 байт/с* – программная реализация и *96970 байт/с* – аппаратная реализация).

5.2. ПРИНЦИПЫ ПОСТРОЕНИЯ АЛГОРИТМА ШИФРОВАНИЯ ДАННЫХ DES

Как показано на рис. 5.1, *DES* – криптографическая система блочного симметричного зашифрования и расшифрования данных.

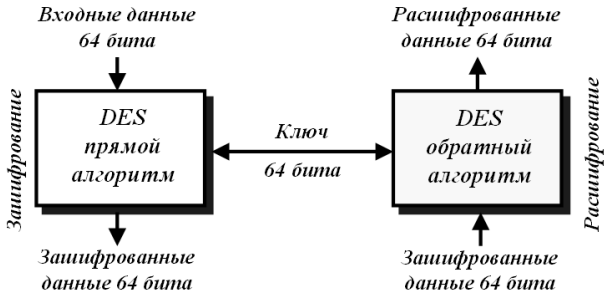


Рис. 5.1. Зашифрование и расшифрование данных с использованием стандарта *DES*

На стороне зашифрования *DES* принимает *64*-битовый исходный блок данных и порождает *64*-битовый зашифрованный блок

данных; на стороне разшифрования *DES* принимает *64*-битовый зашифрованный блок данных и порождает *64*-битовый исходный блок данных. На обеих сторонах для зашифрования и расшифрования применяется один и тот же *64*-битовый ключ.

Алгоритм *DES* также использует комбинацию подстановок и перестановок. *DES* осуществляет зашифрование *64*-битовых блоков данных с помощью *64*-битового ключа, в котором значащими являются *56* бит (каждый восьмой бит используется для контроля четности остальных битов ключа) [12]. Разшифрование в *DES* является операцией, обратной зашифрованию, и выполняется путем повторения операций зашифрования в обратной последовательности.

5.3. СТРУКТУРА АЛГОРИТМА ШИФРОВАНИЯ ДАННЫХ DES

Обобщенная схема процесса зашифрования в алгоритме *DES* показана на рис. 5.2.

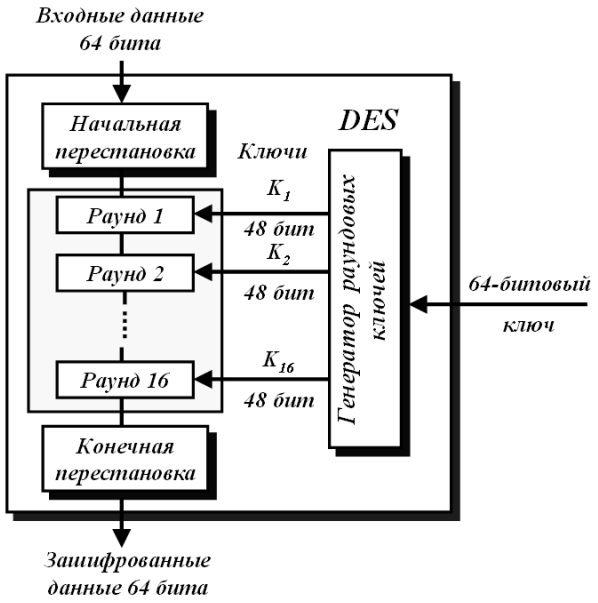


Рис. 5.2. Обобщенная схема процесса зашифрования в алгоритме *DES*

Процесс зашифрования заключается в начальной перестановке битов *64*-битового блока, шестнадцати раундах зашифрования и, наконец, в конечной перестановке битов.

Использование шестнадцати раундов зашифрования обусловлено следующим:

- двенадцать раундов являются минимально необходимыми для обеспечения должного уровня криптографической защиты;

- при аппаратной реализации использование шестнадцати раундов позволяет вернуть преобразованный ключ в исходное состояние для дальнейших преобразований;

- данное количество раундов также необходимо, чтобы исключить возможность проведения атаки на блок зашифрованных данных с двух сторон.

Каждый раунд использует различные (шестнадцать) сгенерированные *48*-битовые ключи.

В некоторых реализациях *DES* блоки открытого сообщения, перед тем, как они будут загружены в регистры самого устройства зашифрования, проходят процедуру начальной перестановки. Данная процедура применяется для того, чтобы осуществить начальное рассеивание статистической структуры сообщения [12, 29].

В случае использования начальной перестановки (*Initial Permutation – IP*), после завершения шестнадцати раундов, к полученному блоку применяется обратная перестановка (IP^{-1}).

IP перестановка битов данных представляет собой проволочную коммутацию с инверсией IP^{-1}, т.е. конечная перестановка обратная начальной. Используемые две перестановки не имеют никакого значения для криптографии в *DES*. Обе перестановки – без ключей и предопределенны. Это позволяет использовать одно и то же программное или аппаратурное обеспечение для двух сторон процесса: зашифрования и расшифрования. Рис. 5.3 показывает начальные и конечные перестановки.

Каждая из перестановок принимает на вход *64*-битовые блоки данных и переставляет его элементы по заданному правилу. На рис. 5.3 показано только небольшое число входных портов и соответствующих выходных портов. Эти перестановки – прямые перестановки без ключей, которые инверсные друг другу. Например, в начальной перестановке (*IP*) *58*-й бит на входе переходит в первый бит на выходе. Аналогично, в конечной перестановке (IP^{-1}) первый входной бит переходит в *58*-й бит на выходе. Другими

словами, если между этими двумя перестановками не существует раунда, *58*-й бит, поступивший на вход устройства начальной перестановки, будет доставлен на *58*-й выход финальной перестановкой.

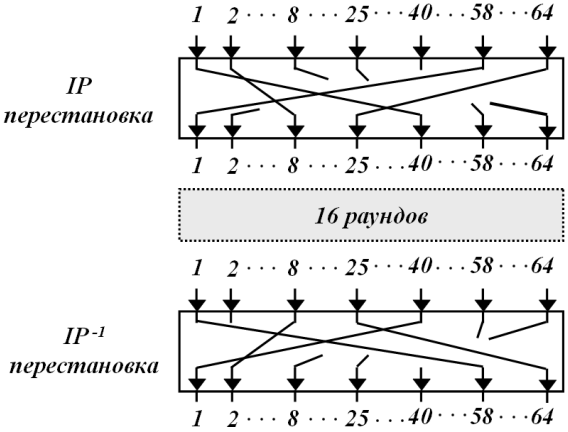


Рис. 5.3. Начальные и конечные шаги перестановки *DES*

Правила начальной *IP* и IP^{-1} конечной перестановок для *64*-битового блока данных показаны в табл. 5.1 и 5.2.

Таблица 5.1

Правила начальной перестановки *IP*

	Номера бит															
Вход	*58*	*50*	*42*	*34*	*26*	*18*	*10*	*02*	*60*	*52*	*44*	*36*	*28*	*20*	*12*	*04*
Выход	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Вход	*62*	*54*	*46*	*38*	*30*	*22*	*14*	*06*	*64*	*56*	*48*	*40*	*32*	*24*	*16*	*08*
Выход	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*
Вход	*57*	*49*	*41*	*33*	*25*	*17*	*09*	*01*	*59*	*51*	*43*	*35*	*27*	*19*	*11*	*03*
Выход	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*	*48*
Вход	*61*	*53*	*45*	*37*	*29*	*21*	*13*	*05*	*63*	*55*	*47*	*39*	*31*	*23*	*15*	*07*
Выход	*49*	*50*	*51*	*52*	*53*	*54*	*55*	*56*	*57*	*58*	*59*	*60*	*61*	*62*	*63*	*64*

Эта замена (*IP* и IP^{-1}) никак не влияет на стойкость шифра, и пользователи часто задавались вопросом: зачем ее вообще делать? Один из членов творческого коллектива, разработавшего *DES*, утверждал, что она облегчает аппаратурную реализацию процедуры шифрования [27, 29, 41].

Таблица 5.2

Правила обратной подстановки IP^{-1}

	Номера бит															
Вход	*40*	*08*	*48*	*16*	*56*	*24*	*64*	*32*	*39*	*07*	*47*	*15*	*55*	*23*	*63*	*31*
Выход	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Вход	*38*	*06*	*46*	*14*	*54*	*22*	*62*	*30*	*37*	*05*	*45*	*13*	*53*	*21*	*61*	*29*
Выход	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*
Вход	*36*	*04*	*44*	*12*	*52*	*20*	*60*	*28*	*35*	*03*	*43*	*11*	*51*	*19*	*59*	*27*
Выход	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*	*48*
Вход	*34*	*02*	*42*	*10*	*50*	*18*	*58*	*26*	*33*	*01*	*41*	*09*	*49*	*17*	*57*	*25*
Выход	*49*	*50*	*51*	*52*	*53*	*54*	*55*	*56*	*57*	*58*	*59*	*60*	*61*	*62*	*63*	*64*

Все перестановки и коды в таблицах подобраны разработчиками таким образом, чтобы максимально затруднить процесс расшифрования путем подбора ключа.

Следует сразу отметить, что все приводимые таблицы являются стандартными и должны включаться в реализацию алгоритма *DES* в неизменном виде.

Пример 5.1. Необходимо определить результат на выходе начального блока перестановки, когда на его вход поступает шестнадцатеричная последовательность, такая как:

$$0002\ 0000\ 0000\ 0001_{16}.$$

Решение. Вход имеет только две единицы (бит *15* и бит *64*); выход должен также иметь только две единицы так как используется прямая перестановка. Используя табл. 5.1, можно найти выход, связанный с этими двумя битами. Бит *15* на входе становится битом *63* в выходе. Бит *64* во входе становится битом *25* в выходе. На выходе будем иметь только две единицы – бит *25* и бит *63*.

Следовательно, результат в шестнадцатеричном исчислении:

$$0000\ 0080\ 0000\ 0002_{16}.$$

Пример 5.2. Доказать, что начальные и конечные перестановки, полученные в примере 5.1 инверсны друг другу.

Решение. Преобразуем полученную выходную последовательность $0000\ 0080\ 0000\ 0002_{16}$ во входную. Единичные биты – *25* и *63*, другие биты равны нулю. В конечной перестановке *25*-й бит переходит в *64*-й, а *63*-й – в *15*-й. Результат:

0002000000000001_{16}.

Как видно из результата начальные и конечные перестановки – это прямые блоки перестановки, которые инверсны друг другу.

5.4. ГЕНЕРАЦИЯ РАУНДОВЫХ КЛЮЧЕЙ ДЛЯ ШИФРОВАНИЯ ДАННЫХ В DES

Генератор ключей создает шестнадцать ключей по *48* бит из ключа шифра длинной *56* бит. Однако ключ шифра обычно дается как ключ из *64*-х бит, в котором *8* дополнительных бит являются битами проверки. Они фактическим отбрасываются перед процессом генерации раундовых ключей, который показан на рис. 5.4.

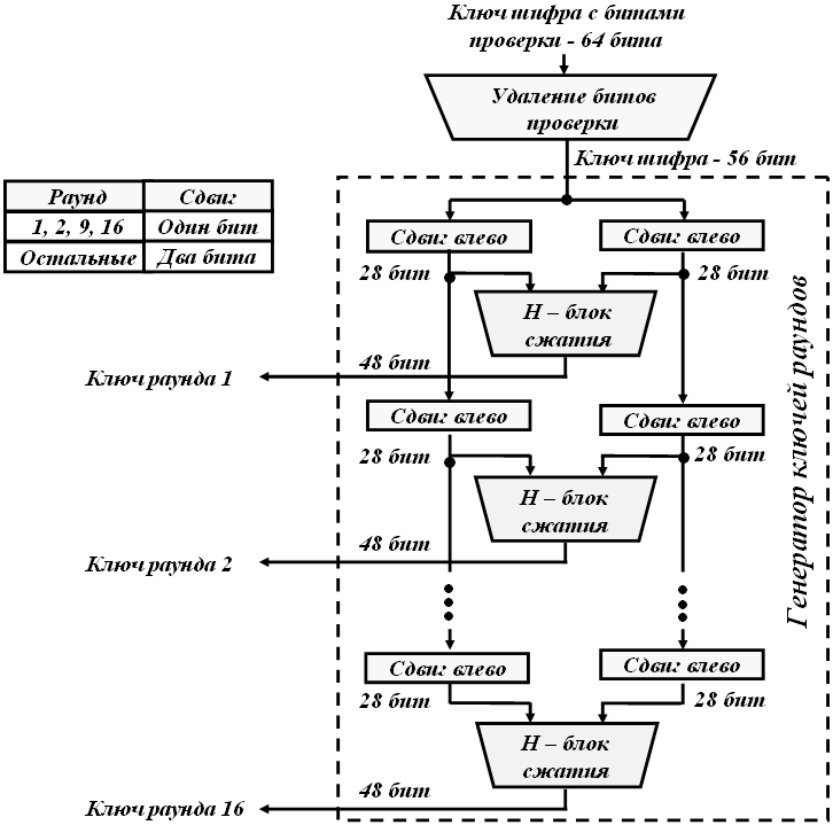


Рис. 5.4. Структурная схема генерации раундовых ключей

5.4.1. Удаление битов проверки ключа шифра

Предварительный процесс перед расширением ключей – перестановка сжатия, которую называют удалением бит проверки ключа шифра. Он удаляет биты четности (биты *08, 16, 24, 32, 40, 48, 56, 64*) из *64*-битового ключа. Эти биты не влияют на шифрование. Перечисленные биты ключа отвечают за то, чтобы каждый байт ключа состоял из нечетного числа единичных битов.

Таким образом, в действительности ключ шифра является *56*-битовым. Для удаления контрольных бит и подготовка ключа к работе используется функция *G* первоначальной подготовки ключа (табл. 5.3). Результатом выполнения функции *G* будет перестановка, которая в литературе называется *PC-1* [12, 29].

Таблица 5.3

Функция *G* первоначальной подготовки ключа

	Номера бит													
Вход K	*57*	*49*	*41*	*33*	*25*	*17*	*09*	*01*	*58*	*50*	*42*	*34*	*26*	*18*
Выход	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*
Вход K	*10*	*02*	*59*	*51*	*43*	*35*	*27*	*19*	*11*	*03*	*60*	*52*	*44*	*36*
Выход	*15*	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*
Вход K	*63*	*55*	*47*	*39*	*31*	*23*	*15*	*07*	*62*	*54*	*46*	*38*	*30*	*22*
Выход	*29*	*30*	*31*	*32*	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*
Вход K	*14*	*06*	*61*	*53*	*45*	*37*	*29*	*21*	*13*	*05*	*28*	*20*	*12*	*04*
Выход	*43*	*44*	*45*	*46*	*47*	*48*	*49*	*50*	*51*	*52*	*53*	*54*	*55*	*56*

Табл. 5.3 разделена на две части. Результат преобразования *G(K)* разбивается на две половины C_0 и D_0 по *28* бит каждая. Первые две строки табл. 5.3 определяют, как выбираются биты последовательности C_0 (первым битом C_0 будет *57*-й бит ключа шифра, затем *49*-й бит и т.д., а последними битами – *44*-й и *36*-й бит ключа). Следующие две строки табл. 5.3 определяют, как выбираются биты последовательности D_0 (т.е. последовательность D_0 будет состоять из битов *63, 55, 47*, … , *12, 04* ключа шифра).

Пример 5.3. Необходимо определить результат на выходе функции первоначальной подготовки ключа *G*, если ключ шифра равен

$$K = abab19283746cdcd_{16}.$$

Решение. Заменяя значения ключа шифра на двоичное

$$1010\ 1011\ 1010\ 1011\ 0001\ 1001\ 0010\ 1000_2$$
$$0011\ 0111\ 0100\ 0110\ 1100\ 1101\ 1100\ 1101_2$$

и используя функцию G первоначальной подготовки ключа (табл. 5.3) получим последовательности:

$$C_0 = 1100\ 0011\ 1100\ 0000\ 0011\ 0011\ 1010_2 = c3c033a_{16};$$

$$D_0 = 0011\ 0011\ 1111\ 0000\ 1100\ 1111\ 1010_2 = 33f0cfa_{16}.$$

5.4.2. Циклический сдвиг влево последовательностей C_i и D_i

После прямой перестановки ключ разделен на две части по *28* бит. После формирования последовательностей C_0 и D_0 рекурсивно определяются последовательности C_i и D_i для каждого раунда из *16*-ти ($i = 1,2,...,16$). Для этого применяются операции циклического сдвига влево на один или два бита в зависимости от номера раунда, как показано в табл. 5.4.

Таблица 5.4

Количество бит циклического сдвига влево последовательностей C_i и D_i

Номера раундов															
1	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
1	*1*	*2*	*2*	*2*	*2*	*2*	*2*	*1*	*2*	*2*	*2*	*2*	*2*	*2*	*1*
Количество сдвигаемых битов влево															

Операции циклического сдвига влево выполняются для последовательностей C_i и D_i независимо. Например, последовательность C_3 получается посредством циклического сдвига влево на две позиции последовательности C_2, а последовательность D_3 - посредством циклического сдвига влево на две позиции последовательности D_2. Последовательности C_{16} и D_{16} получаются из C_{15} и D_{15} соответственно посредством их циклического сдвига влево на одну позицию.

Общее количество циклических сдвигов (общий циклический сдвиг последовательностей C_i и D_i составляет *28* бит) в схеме генерации раундовых ключей выбрано таким образом, чтобы после формирования последнего раундового ключа значения последовательностей C_{16} и D_{16} были равны значениям последовательностей C_0 и D_0.

Пример 5.4. Необходимо определить результат после операции циклического сдвига влево последовательностей C_0 и D_0 для формирования второго раундового ключа k_2, если значения C_0 и D_0 равны

$$C_0 = 1100\ 0011\ 1100\ 0000\ 0011\ 0011\ 1010_2 = c3c033a_{16};$$

$$D_0 = 0011\ 0011\ 1111\ 0000\ 1100\ 1111\ 1010_2 = 33f0cfa_{16}.$$

Решение. Для формирования второго раундового ключа k_2 в соответствие с табл. 5.4 последовательности C_0 и D_0 должны быть циклически сдвинуты влево на два разряда. В результате циклического сдвига влево последовательностей C_0 и D_0 на два разряда получим

$$C_2 = 0000\ 1111\ 0000\ 0000\ 1100\ 1110\ 1011_2 = 0f00ceb_{16};$$

$$D_2 = 1100\ 1111\ 1100\ 0011\ 0011\ 1110\ 1000_2 = cfc33e8_{16}.$$

5.4.3. Объединение последовательностей C_i и D_i, их перестановка и сжатие

После циклического сдвига последовательностей C_i и D_i влево, они объединяются, чтобы создать блок в *56* бит, т.е. получается последовательность $C_i||D_i$, где || – знак математической операции конкатенации (объединения).

Перестановка и сжатие (*H*-функция) изменяет *56* битов $C_i||D_i$ на *48* битов k_i, которые и используются как раундовые ключи. Перестановка и сжатие последовательности $C_i||D_i$ показана в табл. 5.5.

Таблица 5.5

Функция завершающей обработки раундовых ключей *H*

Вход	*14*	*17*	*11*	*24*	*01*	*05*	*03*	*28*	*15*	*06*	*21*	*10*	*23*	*19*	*12*	*04*
Выход k_i	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Вход	*26*	*08*	*16*	*07*	*27*	*20*	*13*	*02*	*41*	*52*	*31*	*37*	*47*	*55*	*30*	*40*
Выход k_i	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*
Вход	*51*	*45*	*33*	*48*	*44*	*49*	*39*	*56*	*34*	*53*	*46*	*42*	*50*	*36*	*29*	*32*
Выход k_i	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48

Раундовые ключи k_i, определяемые на каждом шаге итерации, есть результат выбора конкретных битов из *56*-битовой последовательности $C_i||D_i$, их перестановки и сжатия. Другими словами, раундовый ключ

$$k_i = H(C_i||D_i), \qquad (5.1)$$

где H – функция завершающая обработку ключа, длиною *48* бит, определяемая по табл. 5.5. Результатом выполнения функции H будет перестановка, которая в литературе называется *PC-2* [12, 29].

Как следует из табл. 5.5, первым битом ключа k_i будет *14*-й бит последовательности $C_i||D_i$, вторым – *17*-й бит, *47*-м битом ключа k_i будет *29*-й бит $C_i||D_i$, а *48*-м битом – *32*-й бит $C_i||D_i$.

На входе функции H была последовательность $C_i||D_i$ длиною *56* бит, а после ее выполнения длина раундового ключа k_i стала равной *48* бит. Сжатие произошло за счет отказа от участия в перестановки с помощью функции H восьми бит из последовательности $C_i||D_i$ с номерами *9, 18, 22, 25, 35, 38, 43* и *54* (см. табл. 5.5).

Пример 5.5. Необходимо определить результат после операции объединения последовательностей C_2 и D_2 для последующего формирования второго раундового ключа k_2, если значения C_2 и D_2 равны

$$C_2 = 0f00ceb_{16}; \quad D_2 = cfc33e8_{16}.$$

Решение. Объединение последовательностей C_2 и D_2 можно осуществить следующим образом:

$$C_2||D_2 = 0f00ceb16||cfc33e8_{16} = 0f00cebcfc33e8_{16}.$$

Пример 5.6. Необходимо сгенерировать второй раундовый ключ k_2, т.е. определить результат на выходе функции завершающей обработки раундовых ключей H, если на ее входе объединенные последовательности C_2 и D_2 равны: $C_2||D_2 = 0f00cebcfc33e8_{16}$.

Решение. Используя двоичное значение $C_2||D_2$

$$C_2||D_2 = 0f00cebcfc33e8_{16} =$$

$$= 0000\ 1111\ 0000\ 0000\ 1100\ 1110\ 1011$$

$$1100\ 1111\ 1100\ 0011\ 0011\ 1110\ 1000_2$$

и произведя замену с помощью табл. 5.5, получим второй раундовый ключ k_2

$$k_2 = 0100\ 0101\ 0110\ 1000\ 0101\ 1000\ 0001\ 1010$$

$$1011\ 1100\ 1100\ 1110_2 = 4568581abcce_{16}.$$

5.4.4. Алгоритм генерации раундовых ключей

Теперь представим простой алгоритм для создания раундовых ключей из ключа шифра с проверочными битами, который использует несколько процедур алгоритма:

```
Key_Generator (key[64], RoundKeys[16, 48]. SinfTable[16])
{
    permute (64, 56, key, cipherKey, ParityDropTable)
    spilt (56, 28, cipherKey, leftKey, rightKey)
    for (round = 1 to 16)
    {
            shiftLeft (leftKey, ShiftTable[round])
            shiftLeft (rightKey, ShiftTable[round])
            combine (28, 56, leftKey, rightKey, preRoundKey)
            permute (56, 48, preRoundKey, RoundKeys[round],
            KevCompressionTable)
    }
}
shiftLeft (block[28], NumOfShifts)
{
    for (i = 1 to numShifts)
    {
            T ← block[1]
            for (j = 2 to 28) { block [j-1] ← block [j] }
            block[28] ← T
    }
}
```

Здесь *ShiftLeft* процедура левого сдвига. Обратите внимание, что *T* – временный блок.

5.5. ШИФРОВАНИЕ ДАННЫХ В DES

Рассмотрим сначала зашифрование, а потом расшифрование данных в *DES*.

После осуществления начальной перестановки *IP* над входным блоком данных *x*, полученная *64*-битная последовательность $x_0 = IP(x)$ представляется в виде:

$$x_0 = L_0 \,||\, R_0,$$

где L_0 – левая (*left*) и R_0 – правая (*right*) последовательности битов, которые имеют одинаковую длину, равную *32* битам.

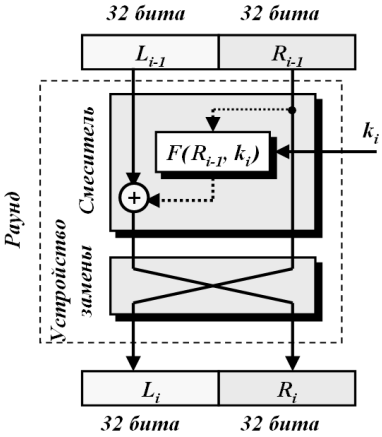


Рис. 5.5. Один раунд зашифрования данных в *DES*

Непосредственно процесс шифрования в *DES* состоит из шестнадцати раундов, поэтому блок данных $x_0 = L_0 \| R_0$ преобразуется далее шестнадцать раз по так называемой схеме Фейстеля. Каждый раунд *DES* использует схему Фейстеля, как это показано на рис. 5.5.

Процесс шифрования данных с использованием шестнадцати раундов можно представить математически:

$$L_i = R_i;$$
$$R_i = L_{i-1} \oplus F(R_{i-1}, k_i),\ i = 1,2,.,16, \tag{5.2}$$

где $F(\bullet)$ – функция зашифрования; k_i – раундовые ключи; $\oplus$ – операция побитового сложения данных по модулю *2* (*xor*).

После выполнения шестнадцати раундов шифрования блок данных $x_{16} = L_{16} \| R_{16}$, подвергается обратной IP^{-1} перестановке. В результате получается *64*-битный блок зашифрованных данных y

$$y = IP^{-1}(x_{16}) = IP^{-1}(L_{16} \| R_{16}). \tag{5.3}$$

Каждый текущий раунд принимает L_{i-1} и R_{i-1} от предыдущего раунда (или начального блока перестановки) и создает для следующего раунда L_i и R_i, которые поступают на следующий раунд (или конечный блок перестановки). Как было выше указано, можно принять, что каждый раунд имеет два элемента шифра: смеситель и устройство замены. Каждый из этих элементов является обратимым. Устройство замены – очевидно обратимо, оно меняет местами левую половину данных с правой половиной. Смеситель является обратимым, потому что операция *xor* обратима. Все необратимые элементы сосредоточены в *функции Фейстеля* $F(R_{i-1}, k_i)$.

5.5.1. Функция *F* смесителя *DES*

Основой схемы Фейстеля в *DES* является функция *F*. Функция *F* с помощью *48*-битового ключа зашифровывает *32* самых правых бит (R_{i-1}), чтобы получить на выходе *32*-битовое слово. Эта функция содержит, как показано на рис. 5.6, четыре операции: *E*-блок перестановки и расширения; поразрядного суммирования по модулю *2* ($\oplus$); группу *S*-блоков замены и сжатия; *P*-блок прямой перестановки.

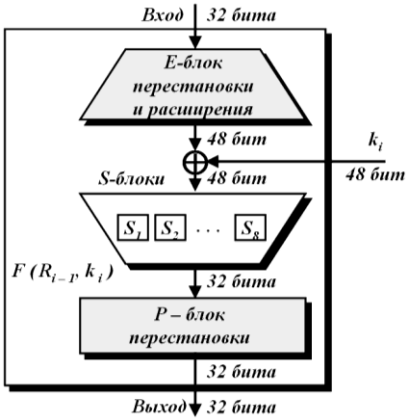


Рис. 5.6. Функция *F DES*

Операция перестановки и расширения функции F

Так как на входе R_{i-1} имеет длину 32 бита, а ключ k_i – длину *48* битов, сначала нужно расширить R_{i-1} до *48* бит. Для этого вначале R_{i-1} разделяется на *8* секций по *4* бита. Затем каждая секция на *4* бита расширяется до *6* бит. Эта перестановка с расширением следует по заранее определенным правилам. Для секции значения входных бит *1, 2, 3* и *4* присваиваются битам *2, 3, 4* и *5* соответственно на выходе. Выходной бит *1* формируется на основе входного бита *4* из предыдущей секции; бит выхода *6* формируется из бита *1* в следующей секции. Если секции *1* и *8* рассматривать как соседние секции, то те же самые правила применяются к битам *1* и *32*.

Таким образом, после *E*-блока перестановки и расширения получается последовательность R_{i-1}^{E}

$$R_{i-1}^{E} = E(R_{i-1}). \qquad (5.4)$$

Рис. 5.7 показывает входы и выходы в *E*-блоке перестановки и расширения.

Хотя отношения между входом и выходом могут быть определены математически, проще определить этот *E*-блок перестановки и расширения используя табл. 5.6.

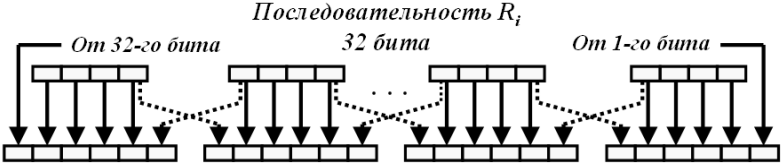


Рис. 5.7. Пояснение процессов *E*-блока перестановки и расширения

Таблица 5.6

Правила *E*-блока перестановки и расширения

Вход	*32*	*01*	*02*	*03*	*04*	*05*	*04*	*05*	*06*	*07*	*08*	*09*	*08*	*09*	*10*	*11*
Выход R_{i-1}^E	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Вход	*12*	*13*	*12*	*13*	*14*	*15*	*16*	*17*	*16*	*17*	*18*	*19*	*20*	*21*	*20*	*21*
Выход R_{i-1}^E	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*
Вход	*22*	*23*	*24*	*25*	*24*	*25*	*26*	*27*	*28*	*29*	*28*	*29*	*31*	*31*	*32*	*01*
Выход R_{i-1}^E	*33*	*34*	*35*	*36*	*37*	*38*	*39*	*40*	*41*	*42*	*43*	*44*	*45*	*46*	*47*	*48*

Обратите внимание, что число выходов *48*, а входов - только *32*. Некоторые из входов идут к более чем одному выходу. Например, значение входного бита *5* становится значением битов выхода *6* и *8*.

Пример 5.7. Необходимо определить последовательность R_0^E на выходе *E*-блока перестановки и расширения, если на его вход поступает последовательность $R_0 = 18ca18ad_{16}$.

Решение. Используя двоичное значение R_0

$$R_0 = 18ca18ad_{16} = 0001\ 1000\ 1100\ 1010\ 0001\ 1000\ 1010\ 1101_2$$

и произведя замену с помощью табл. 5.6, получим:

$$R_0^E = 1000\ 1111\ 0001\ 0110\ 0101\ 0100\ 0000\ 1111$$
$$0001\ 0101\ 0101\ 1010_2 = 8f16540f155_{16}.$$

Операция поразрядного суммирования функции F

После расширения последовательности R_{i-1} длиной *32* бита до последовательности R_{i-1}^E длиной *48* битов с использованием *E*-блока перестановки и расширения использует операцию *xor* над расши-

ренной частью правой секции R_{i-1}^{E} и ключом раунда k_i. Заметим, что правая секция R_{i-1}^{E} и ключ k_i имеют длину *48* бит. Также заметим, что ключ раунда использует только эту операцию.

Таким образом после выполнения операции *xor* получим последовательность $R_{i-1}^{\oplus}$ длинной *48* бит:

$$R_{i-1}^{\oplus} = R_{i-1}^{E} \oplus k_i. \qquad (5.5)$$

Пример 5.8. Необходимо определить последовательность $R_0^{\oplus}$ на выходе сумматора по модулю *2*, если на его вход поступает последовательность $R_0^{E} = 8f16540f155_{16}$ и раундовый ключ $k_1 = 194cd072de8c_{16}$.

Решение. Представляя значения R_0^{E} и k_1 в двоичном виде и используя правило поразрядного суммирования данных по модулю *2*, получим

$$\begin{array}{cll} & R_0^{E} & 100011110001011001010100000011110001010101011010_2 \\ \oplus & k_1 & 000110010100110011010000011100101101111010001100_2 \\ \hline & R_0^{\oplus} & 100101100101101010000100011111011100101111010110_2 \end{array}$$

Преобразованное значение $R_0^{\oplus}$ в шестнадцатеричное будет иметь вид $R_0^{\oplus} = 965a847dcbd6_{16}$.

Операция замены и сжатия функции F

S-блоки замены и сжатия смешивают информацию (операция перемешивания). *DES* использует *S*-блоки, каждый с *6* входными битами и *4* выходными (рис. 5.8).

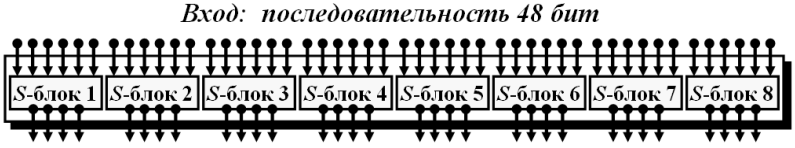


Рис. 5.8. *S* - блоки замены и сжатия

Последовательность $R_{i-1}^{\oplus}$ из *48* битов после второй операции функции *F* разделяется на восемь секций по *6* битов, и каждая секция поступает в соответствующий *S*-блок замены и сжатия. Результат каждого *S*-блока замены и сжатия – данные длиной *4* бита; когда они объединяются, то результат будет представлен последовательностью R_{i-1}^{S} в 32 бита:

$$R_{i-1}^{S} = S(R_{i-1}^{\oplus}). \tag{5.6}$$

Для определения значения R_{i-1}^{S} вычисленная сумма $R_{i-1}^{\oplus}$ записывается в виде конкатенации восьми *6*-битовых слов:

$$R_{i-1}^{\oplus} = B_1 || B_2 || B_3 || B_4 || B_5 || B_6 || B_7 || B_8. \tag{5.7}$$

На этом этапе каждое слово B_j поступает на соответствующий S_i-блок замены и сжатия. Основная задача *S*-блоков заключается в преобразовании *48*-битного вектора в *32*-битный. Всего в *DES* используется восемь *S*-блоков с *6*-битными входами и *4*-битными выходами. S_j-блок – это матрица размером 4×16 с целыми элементами в диапазоне от *0* до *15*.

Каждое слово B_j представляется в виде:

$$B_j = \underbrace{b_1 || b_6}_{\substack{Номер \\ строки}} || \underbrace{b_2 || b_3 || b_4 || b_5}_{Номер\ столбца}. \tag{5.8}$$

Первый и шестой биты слова B_j, если их рассматривать как двоичную запись числа, определяют номер строки матрицы S_j-блока, а четыре остальных бита определяют номер столбца. На пересечении определенного номера строки и столбца заданного S_j-блока находится элемент матрицы. Его двоичная запись и является выходом.

Совокупность *6*-битовых блоков (5.7) обеспечивает выбор *4*-битового элемента в каждом из блоков:

$$S = S_1, S_2, S_3, S_4, S_5, S_6, S_7, S_8. \tag{5.9}$$

В результате получаем:

$$R_{i-1}^{S} = S_1(B_1)||S_2(B_2)||S_3(B_3)||S_4(B_4)||S_5(B_5)||S_6(B_6)||S_7(B_7)||S_8(B_8)||.$$

Поскольку для каждого *S*-блока есть собственная таблица, необходимо иметь восемь таблиц, например таких, как это показано в табл. 5.7–5.14. Значение входа (номер строки и номер столбца) и значения выхода даются как десятичные номера, чтобы сэкономить место на странице.

В реальности они могут быть заменены двоичными числами.

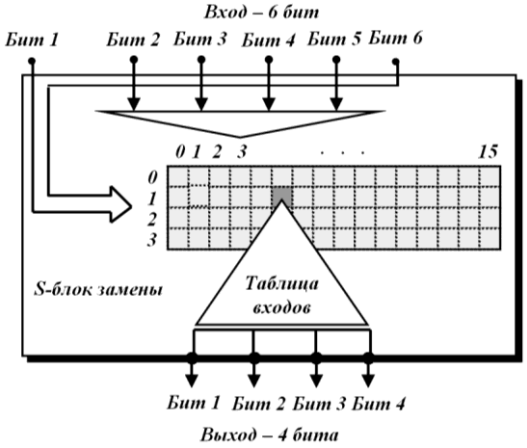


Рис. 5.9. Правила для *S*-блока замены и сжатия

Таблица 5.7

S_1-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*14*	*04*	*13*	*01*	*02*	*15*	*11*	*08*	*03*	*10*	*06*	*12*	*05*	*09*	*00*	*07*
	01	*00*	*15*	*07*	*04*	*14*	*02*	*13*	*01*	*10*	*06*	*12*	*11*	*09*	*05*	*03*	*08*
	02	*04*	*01*	*14*	*08*	*13*	*06*	*02*	*11*	*15*	*12*	*09*	*07*	*03*	*10*	*05*	*00*
	03	*15*	*12*	*08*	*02*	*04*	*09*	*01*	*07*	*05*	*11*	*03*	*14*	*10*	*00*	*06*	*13*

Таблица 5.8

S_2-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*15*	*01*	*08*	*14*	*06*	*11*	*03*	*04*	*09*	*07*	*02*	*13*	*12*	*00*	*05*	*10*
	01	*03*	*13*	*04*	*07*	*15*	*02*	*08*	*14*	*12*	*00*	*01*	*10*	*06*	*09*	*11*	*05*
	02	*00*	*14*	*07*	*11*	*10*	*04*	*13*	*01*	*05*	*08*	*12*	*06*	*09*	*03*	*02*	*15*
	03	*13*	*08*	*10*	*01*	*03*	*15*	*04*	*02*	*11*	*06*	*07*	*12*	*00*	*05*	*14*	*09*

Таблица 5.9

S_3-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*10*	*00*	*09*	*14*	*06*	*03*	*15*	*05*	*01*	*13*	*12*	*07*	*11*	*04*	*02*	*08*
	01	*13*	*07*	*00*	*09*	*03*	*04*	*06*	*10*	*02*	*08*	*05*	*14*	*12*	*11*	*15*	*01*
	02	*13*	*06*	*04*	*09*	*08*	*15*	*03*	*00*	*11*	*01*	*02*	*12*	*05*	*10*	*14*	*07*
	03	*01*	*10*	*13*	*00*	*06*	*09*	*08*	*07*	*04*	*15*	*14*	*03*	*11*	*05*	*02*	*12*

Таблица 5.10

S_4-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*07*	*13*	*14*	*03*	*00*	*06*	*09*	*10*	*01*	*02*	*08*	*05*	*11*	*12*	*04*	*15*
	01	*13*	*08*	*11*	*05*	*06*	*15*	*00*	*03*	*04*	*07*	*02*	*12*	*01*	*10*	*14*	*09*
	02	*10*	*06*	*09*	*00*	*12*	*11*	*07*	*13*	*15*	*01*	*03*	*14*	*05*	*02*	*08*	*04*
	03	*03*	*15*	*00*	*06*	*10*	*01*	*13*	*08*	*09*	*04*	*05*	*11*	*12*	*07*	*02*	*14*

Таблица 5.11

S_5-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*02*	*12*	*04*	*01*	*07*	*10*	*11*	*06*	*08*	*05*	*03*	*15*	*13*	*00*	*14*	*09*
	01	*14*	*11*	*02*	*12*	*04*	*07*	*13*	*01*	*05*	*00*	*15*	*10*	*03*	*09*	*08*	*06*
	02	*04*	*02*	*01*	*11*	*10*	*13*	*07*	*08*	*15*	*09*	*12*	*05*	*06*	*03*	*00*	*14*
	03	*11*	*08*	*12*	*07*	*01*	*14*	*02*	*13*	*06*	*15*	*00*	*09*	*10*	*04*	*05*	*03*

Таблица 5.12

S_6-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*12*	*01*	*10*	*15*	*09*	*02*	*06*	*08*	*00*	*13*	*03*	*04*	*14*	*07*	*05*	*11*
	01	*10*	*15*	*04*	*02*	*07*	*12*	*09*	*05*	*06*	*01*	*13*	*14*	*00*	*11*	*03*	*08*
	02	*09*	*14*	*15*	*05*	*02*	*08*	*12*	*03*	*07*	*00*	*04*	*10*	*01*	*13*	*11*	*06*
	03	*04*	*03*	*02*	*12*	*09*	*05*	*15*	*10*	*11*	*14*	*01*	*07*	*06*	*00*	*08*	*13*

Таблица 5.13

S_7-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*04*	*11*	*02*	*14*	*15*	*00*	*08*	*13*	*03*	*12*	*09*	*07*	*05*	*10*	*06*	*01*
	01	*13*	*00*	*11*	*07*	*04*	*09*	*01*	*10*	*14*	*03*	*05*	*12*	*02*	*15*	*08*	*06*
	02	*01*	*04*	*11*	*13*	*12*	*03*	*07*	*14*	*10*	*15*	*06*	*08*	*00*	*05*	*09*	*02*
	03	*06*	*11*	*13*	*08*	*01*	*04*	*10*	*07*	*09*	*05*	*00*	*15*	*14*	*02*	*03*	*12*

Таблица 5.14

S_8-блок замены и сжатия

	Номера столбцов																
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера строк	*00*	*13*	*02*	*08*	*04*	*06*	*15*	*11*	*01*	*10*	*09*	*03*	*14*	*05*	*00*	*12*	*07*
	01	*01*	*15*	*13*	*08*	*10*	*03*	*07*	*04*	*12*	*05*	*06*	*11*	*00*	*14*	*09*	*02*
	02	*07*	*11*	*04*	*01*	*09*	*12*	*14*	*02*	*00*	*06*	*10*	*13*	*15*	*03*	*05*	*08*
	03	*02*	*01*	*14*	*07*	*04*	*10*	*08*	*13*	*15*	*12*	*09*	*00*	*03*	*05*	*06*	*11*

Аналитическая сложность расшифрования *DES* зависит от математических свойств *S*-блоков замены, поскольку именно в них реализуются нелинейные преобразования.

Все остальные операции в этом алгоритме носят линейный характер. Аналитическое вычисление линейного преобразования не представляет труда для криптографического аналитика.

Нелинейность преобразований, осуществляемых *DES*, определяется только *S*-блоками замены и сжатия. Их выбор не имеет достаточно подробного обоснования. Высказывались мнения о том, что *S*-блоки замены и сжатия имеют некоторую “лазейку”, позволяющую осуществлять контроль за шифрованной перепиской.

Официальная же версия такова: в 1976 г. Национальное бюро стандартов США заявило, что выбор *S*-блоков замены и сжатия определен следующими требованиями [18, 29, 40]:

- каждая строка табличного задания каждого *S*-блока замены и сжатия и сжатия должна быть перестановкой множества *{0,1, ... , 15}*;

- *S*-блоки замены и сжатия не должны быть линейными или аффинных функциями своих входов;

- изменение одного бита входа *S*-блока замены и сжатия должно приводить к изменению по крайней мере двух битов выхода;

- для каждого *S*-блока замены и сжатия и любого входа *x* значение $S(x)$ и $S(x \oplus (0,0,1,1,0,0))$ должны различаться минимум двумя битами.

Поясним процессы подстановки (замены) с помощью *S*-блоков на примерах.

Пример 5.9. Входная последовательность S_1-блока замены и сжатия равна $\underline{1}\,0\,1\,1\,1\,\underline{0}$. Определить какая последовательность будет на выходе *S*-блока?

Решение. Если записать первый и шестой биты вместе, то получим в двоичном исчислении 10_2, которое выражается как число 2_{10} при десятичном исчислении. Остающаяся часть битов 0111_2 в двоичном исчислении является 7_{10} в десятичном исчислении.

На пересечении строки *2* и столбца *7* в табл. 5.7 (S_1-блок замены и сжатия) находится значение результата – 11_{10} в десятичном исчислении или 1011_2 в двоичном исчислении. Тогда вход 101110_2 дает выход 1011_2.

Пример 5.10. Входная последовательность S_8-блока замены и сжатия равна $\underline{0}\,0\,0\,0\,0\,\underline{0}$. Определить какая последовательность будет на выходе S_8-блока?

Решение. Если записать первый и шестой биты вместе, то получим в двоичном исчислении 00_2, которое выражается как число 0_{10} при десятичном исчислении. Остающаяся часть битов 0000_2 в двоичном исчислении является 0_{10} в десятичном исчислении.

На пересечении строки *0* и столбца *0* в табл. 5.14 (S_8-блок замены и сжатия) находится значение результата – 13_{10} в десятичном исчислении или 1101_2 в двоичном исчислении. Тогда вход 000000_2 дает выход 1101_2.

Операция прямой перестановки функции F

P-блок прямой перестановки – последняя операция в функции *F* – прямая перестановка с *32* битами на входе и *32* битами на выходе:

$$R_{i-1}^{P} = P(R_{i-1}^{S}). \qquad (5.10)$$

Отношения “вход-выход” для этой операции показаны в табл. 5.15. Они следуют тем же самым общим правилам, как и предыду-

щие таблицы перестановки. Например, седьмой бит входа становится вторым битом выхода.

Таблица 5.15

Правила *P*-блока прямой перестановки

Вход R_{i-1}^S	*16*	*07*	*20*	*21*	*29*	*12*	*28*	*17*	*01*	*15*	*23*	*26*	*05*	*18*	*31*	*10*
Выход R_{i-1}^P	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Вход R_{i-1}^S	*02*	*08*	*24*	*14*	*32*	*27*	*03*	*09*	*19*	*13*	*30*	*06*	*22*	*11*	*04*	*25*
Выход R_{i-1}^P	*17*	*18*	*19*	*20*	*21*	*22*	*23*	*24*	*25*	*26*	*27*	*28*	*29*	*30*	*31*	*32*

Пример 5.11. Входная последовательность *P*-блока прямой перестановки равна $R_0^S = ec5965b5_{16}$. Определить какая последовательность будет на выходе *P*-блока прямой перестановки?

Решение. Используя двоичное значение R_0^S:

$$R_0^S = ec5965b5_{16} = 1110\ 1100\ 0101\ 1001\ 0110\ 0101\ 1011\ 0101_2$$

и произведя замену с помощью табл. 5.15, получим:

$$R_0^P = 1000\ 0110\ 1000\ 1101\ 1010\ 1110\ 1111\ 1001_2 = 868daef9_{16}.$$

5.5.2. Поразрядное суммирование в раунде *DES*

Результат преобразования с помощью *P*-блока прямой перестановки суммируется по модулю *2* (результат операции *xor*) с *32*-битной последовательностью L_{i-1} и получается *32*-битная последовательность R_{i-1}^L (см. рис. 5.6):

$$R_{i-1}^L = R_{i-1}^P \oplus L_{i-1}. \qquad (5.11)$$

Пример 5.12. Необходимо определить последовательность R_0^L на выходе сумматора по модулю *2* раунда *DES*, если на его вход поступает последовательность $R_0^P = 868daef9_{16}$ и $L_0 = acf014a7_{16}$.

Решение. Представляя значения R_0^P и L_0 в двоичном виде и используя правило поразрядного суммирования данных по модулю *2* получим:

$$\begin{array}{cll} & R_0^P & 10000110100011011010111011111001_2 \\ \oplus & L_0 & 10101100111100000001010010100111_2 \\ \hline & R_0^L & 00101010011111101101110100101111 0_2 \end{array}$$

Преобразованное значение R_{i-1}^L в шестнадцатеричное будет иметь вид $R_{i-1}^L = 2a7dba5e_{16}$.

5.5.3. Устройство замены секций раунда *DES*

Устройство замены секций предназначено для завершения операций одного раунда *DES*, т.е оно формирует значения последовательностей L_i и R_i для следующего раунда. В соответствии с (5.2) и (5.11) последовательности L_i и R_i будут определяться следующим образом:

$$\begin{gathered} L_i = R_{i-1}; \\ R_i = R_{i-1}^L; \ i = 1,2,\ldots,16. \end{gathered} \tag{5.12}$$

Пример 5.13. Необходимо определить последовательности L_1 и R_1 для второго раунда зашифрования данных, если на вход устройства замены поступают последовательности $R_0 = 305a18ca_{16}$ и $R_0^L = 2a7dba5e_{16}$.

Решение. Используя выражение (5.12) и значения R_0 и R_0^L получим:

$$L_1 = R_0 = 305a18ca_{16}; \quad R_1 = R_0^L = 2a7dba5e_{16}.$$

5.5.4. Алгоритм шифрования данных в *DES*

Используя смеситель и устройство замены, можно создать прямой и обратный шифр для каждого из *16* раундов. Прямой шифр используется на стороне зашифрования; обратный шифр – на стороне расшифрования.

В соответствие с выражениями (5.2)…(5.6) и (5.10)…(5.12) алгоритм шифрования можно представить в виде [21, 22]:

$$\begin{gathered} C = IP(L_0,R_0)\times F(L_0,R_0 \oplus k_1)\times \\ \times F(L_1,R_1 \oplus k_2)\times \ldots \times F(L_{15},R_{15} \oplus k_{16})\times IP^{-1}(R_{16},L_{16}). \end{gathered} \tag{5.13}$$

Как следует из (5.13) в процессе зашифрования данных последовательно используются преобразования IP, $F(\bullet)$ и IP^{-1}.

Каждое из преобразований *Фейстеля* $F(\bullet)$ является композицией двух преобразований.

Первое из них – транспозиция (перестановка) L_i и R_i [21, 22]:

$$T(L_i,R_i) = (L_i,R_i),$$

где T – операция по перестановке местами левой L_i и правой R_i половины блока данных (операция транспозиции).

Второе – фактически является преобразованием *Вернама* *32*-битовых слов, т.е. [21, 22]:

$$V(L_i,R_i) = (L_i,R_i \oplus C),$$

где V – преобразование *Вернама*; C – последовательность, которая зависит от раундового ключа k_i и правой секции L_i.

Таким образом, для любого раунда

$$F_i(\bullet) = T_i(\bullet) \times V_i(\bullet).$$

Кроме того, легко убедиться, что [21, 22]:

$$V^2_i(\bullet) \equiv T^2_i(\bullet)$$

тождественное преобразование. Такие преобразования (элементы группы) принято называть *инволюциями*. Для них, в частности, верно:

$$V_i^{-1}(\bullet) = V_i(\bullet), \quad T_i^{-1}(\bullet) = T_i(\bullet).$$

По правилу обращения произведения элементов неабелевой группы имеем

$$\begin{aligned} M = C^{-1} = IP^{-1}(L_0,R_0) \times V(L_0 \oplus F(R_0 \oplus k_1)) \times T \times \\ \times V(L_1,F(R_1 \oplus k_2)) \times T \times \ldots \times V(L_{15},F(R_{15}) \oplus k_{16})) \times T \times \\ \times T \times IP(R_{16},L_{16}). \end{aligned} \qquad (5.14)$$

Осуществляя преобразования в (5.14) окончательно получим алгоритм расшифрования данных

$$M = IP(R_{16},L_{16}) \times F(R_{15} \oplus F(L_{15} \oplus k_{16})) \times$$
$$\times F(R_{14},F(L_{14} \oplus k_{15})) \times \ldots \times F(R_1,F(L_1) \oplus k_1)) \times IP^{-1}(L_0,R_0). \quad (5.15)$$

Это означает, что расшифрование осуществляется тем же алгоритмом зашифрования (5.13) и раундовыми ключами, но в расписании раундовых ключей (порядке их следования в раундах) надо внести некоторое изменение: поменять на обратный порядок генерации раундовых ключей, т.е. выборка раундовых ключей при расшифровании будет обратной, т.е.:

$$k_{16}, k_{15}, k_{14}, k_{13}, k_{12}, k_{11}, k_{10}, k_9, k_8, k_7, k_6, k_5, k_4, k_3, k_2, k_1,$$

если в процессе зашифрования выборку раундовых ключей обозначать как

$$k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_8, k_9, k_{10}, k_{11}, k_{12}, k_{13}, k_{14}, k_{15}, k_{16}.$$

Один из методов, чтобы достигнуть поставленной цели (зашифрование и расшифрование), состоит в том, чтобы сделать последний раунд отличающимся от других; он будет содержать только смеситель и не будет содержать устройства замены, как это показано на рис. 5.10.

В разделе 3 доказано, что смеситель и устройство замены самоинверсны. Конечные и начальные перестановки также инверсны друг другу. Левая секция исходных данных на стороне зашифрования шифруется: L_0 как L_{16}, а L_{16} расшифровывается на стороне расшифрования L_0. Аналогичная ситуация с R_0 и R_{16}.

В первом методе последний раунд не имеет устройства замены.

Пример 5.14. Псевдокоды для *DES* прямого и обратного шифра. Приведен пример в псевдокодах для шифрования и соответствует четырем шагам первого метода. Коды для остальных могут быть сделаны как упражнение.

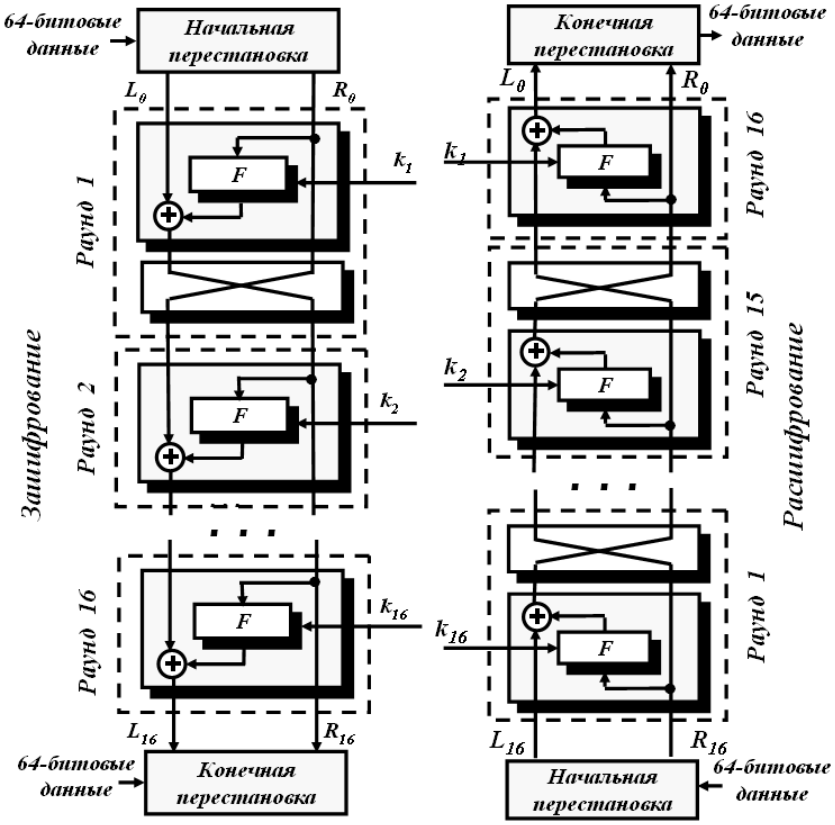


Рис. 5.10. *DES* прямой и обратный шифр для первого способа

```
Cipher (plainBlock[64], Round Keys[16,48])
{
    pemute (64,64, plainBlock, inBlock, IntialPermutationTable)
    spilt (64, 32, inBlock, leftBlock, right Block)
    for (round = 1 to 16)
    {
            mixer (leftBlock, right Block, RoundKey[round])
            if (round!=16) swapper(leftBlock, right Block)
    }
    combine (32, 64, leftBlock, right Block, outBlock)
    pemute (64,64, outBlock, cipherBlock, FinalPermutationTable)
```

```
    mixer (leftBlock[48], right Block[48], RoundKey[48])
    {
        copy (32, rightBlock, T1)
        function (T1, RoundKey,T2)
        exclusive Or (32, leftBlock, T2,T3)
        copy (32, rightBlock, T1)
    }
    swapper (leftBlock[32], right Block[32])
    {
        copy (32, leftBlock, T)
        copy (32, rightBlock, leftBlock)
        copy (32, T, rightBlock)
    }
    Substitute (in block[32], outblock[48], SubstituteTables[8,4,16])
    {
        for(I = 1 to 8)
        {
            row ← 2 × inBlock[i × 6+1] + inBlock[i × 6+6]
            col ← 8 × inBlock[i × 6+2] + 4 × inBlock[i ×
            × 6+3]+2 × inBlock[i × 6+4] + inBlock[i × 6+5]
            value = SubstituteTables[i][row][col]
            outBlock[i × 4+1] ← value/ 8
            value ← value mod 8
            outBlock[i × 4+2] ← value/ 4
            value ← value mod 4
            outBlock[i × 4+3] ← value/ 2
            value ← value mod 8
            outBlock[i × 4+4] value
        }
    }
}
```

При первом способе раунд *16* отличается от других раундов тем, что там не применяется устройство замены. Это необходимо, чтобы сделать последний и первый смесители в шифре одинаковыми.

При втором способе можно сделать все *16* раундов одинаковыми, добавляя к *16*-му раунду дополнительное устройство замены (два устройства замены позволяют нейтрализовать друг друга). Данная схема прямого и обратного шифра показана на рис. 5.11.

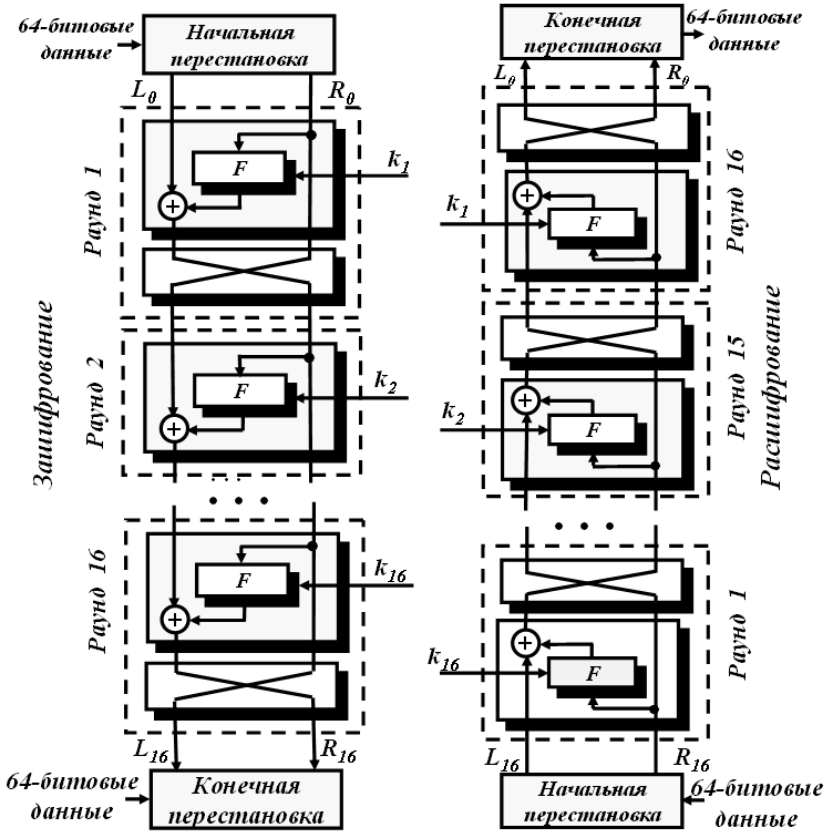


Рис. 5.11. *DES* прямой и обратный шифр для второго способа

5.6. ПРИМЕРЫ ШИФРОВАНИЯ ДАННЫХ В DES

Перед анализом *DES* рассмотрим несколько примеров, чтобы понять, как зашифрование и расшифрование изменяют значение битов в каждом раунде.

Пример 5.15. Выберем случайный блок исходных данных и случайный ключ шифра и определим, каким должен быть блок зашифрованных данных при использовании второго способа шифрования (все цифры даны в шестнадцатеричном исчислении):

- входные данные: $123456abcd132536_{16}$;
- ключ шифра: $abba08192637cddc_{16}$.

Табл. 5.16 показывает результат каждого раунда и данных, созданных до и после раундов.

Таблица 5.16

Трассировка данных в примере 5.16

Раунды	*Левые секции* L_i	*Правые секции* R_i	*Раундовые ключи* k_i
1	$18ca18ad_{16}$	$5a78e394_{16}$	$194cd072de8c_{16}$
2	$5a78e394_{16}$	$4a1210f6_{16}$	$4568581abcce_{16}$
3	$4a1210f6_{16}$	$b8089591_{16}$	$06eda4acf5b5_{16}$
4	$b8089591_{16}$	$236779c2_{16}$	$da2d032b6ee3_{16}$
5	$236779c2_{16}$	$a15a4b87_{16}$	$69a629fec913_{16}$
6	$a15a4b87_{16}$	$2e8f9c65_{16}$	$c1948e87475e_{16}$
7	$2e8f9c65_{16}$	$a9fc20a3_{16}$	$708ad2ddb3c0_{16}$
8	$a9fc20a3_{16}$	$308bee97_{16}$	$34f822f0c66d_{16}$
9	$308bee97_{16}$	$10af9d37_{16}$	$84bb4473dccc_{16}$
10	$10af9d37_{16}$	$6ca6cb20_{16}$	$02765708b5bf_{16}$
11	$6ca6cb20_{16}$	$ff3c485f_{16}$	$6d5560af7ca5_{16}$
12	$ff3c485f_{16}$	$22a5963b_{16}$	$c2c1e96a4bf3_{16}$
13	$22a5963b_{16}$	$387ccdaa_{16}$	$99c31397c91f_{16}$
14	$387ccdaa_{16}$	$bd2dd2ab_{16}$	$251b8bc717d0_{16}$
15	$bd2dd2ab_{16}$	$cf26b472_{16}$	$3330c5d9a36d_{16}$
16	$cf26b472_{16}$	$19ba9212_{16}$	$181c5d75c66d_{16}$

Таблица показывает результат *16* раундов, которые включают смешивание и замену (исключая последний раунд).

Исходные (открытые) данные – прошедшие через начальную перестановку для получения различных *64* бит (*16* шестнадцатеричных цифр): $14a7d67818ca18ad_{16}$.

После разбиения на левую и правую секции: $L_0 = 14a7d678_{16}$ и $R_0 = 18ca18ad_{16}$.

После *16* раунда и объединения: $L_{16}||R_{16} = cf26b47219ba9212_{16}$.

В результате выполнения конечной перестановки получим зашифрованные данные:

$$C = c0b7a8d05f3a829c_{16}.$$

Следует отметить некоторые положения. Правая секция каждого раунда совпадает с левой секцией следующего раунда. Причина в том, что правая секция проходит через смеситель без изменения, а устройство замены переносит ее в левую секцию. Например, R_1 передается через смеситель второго раунда без изменения, но затем, пройдя устройство замены, она становится L_2.

Пример 5.16. Давайте рассмотрим, как получатель в пункте назначения может расшифровать зашифрованные данные, полученные от отправителя, с помощью совпадающего ключа. Табл. 5.17 показывает результат каждого раунда и данных, созданных до и после раундов.

Таблица 5.17

Трассировка данных в примере 5.17

Раунды	*Левые секции* L_i	*Правые секции* R_i	*Раундовые ключи* k_i
1	$cf26b472_{16}$	$bd2dd2ab_{16}$	$181c5d75c66d_{16}$
2	$bd2dd2ab_{16}$	$387ccdaa_{16}$	$3330c5d9a36d_{16}$
3	$387ccdaa_{16}$	$22a5063b_{16}$	$251b8bc717d0_{16}$
4	$22a5063b_{16}$	$ff3c485f_{16}$	$99c31397c91f_{16}$
5	$ff3c485f_{16}$	$6ca6cb20_{16}$	$c2c1e96a4bf3_{16}$
6	$6ca6cb20_{16}$	$10af9d37_{16}$	$6d5560af7ca5_{16}$
7	$10af9d37_{16}$	$308bee97_{16}$	$02765708b5bf_{16}$
8	$308bee97_{16}$	$a9fc20a3_{16}$	$84bb4473dccc_{16}$
9	$a9fc20a3_{16}$	$2e8f9c65_{16}$	$34f822f0c66d_{16}$
10	$2e8f9c65_{16}$	$a15a4b87_{16}$	$708ad2ddb3c0_{16}$
11	$a15a4b87_{16}$	$236779c2_{16}$	$c1948e87475e_{16}$
12	$236779c2_{16}$	$b8089591_{16}$	$69a629fec913_{16}$
13	$b8089591_{16}$	$4a1210f6_{16}$	$da2d032b6ee3_{16}$
14	$4a1210f6_{16}$	$5a78e394_{16}$	$06eda4acf5b5_{16}$
15	$5a78e394_{16}$	$18ca18ad_{16}$	$4568581abcce_{16}$
16	$18ca18ad_{16}$	$14a7d678_{16}$	$194cd072de8c_{16}$

Первое правило: ключи раунда должны использоваться в обратном порядке. Сравните табл. 5.16 и табл. 5.17. Ключ раунда *1* такой же, как ключ для раунда *16*. Значения L_0 и R_0 при расшифровании те же самые, что и значения R_{16} и L_{16} соответственно при зашифровании. Аналогичные совпадения будут получены и в других раундах. Это доказывает не только, что прямой и обратный шифр инверсны друг другу, но также то, что каждый раунд при зашифровании имеет соответствующий раунд при расшифровании в обратном шифре. Результат свидетельствует, что начальные и конечные перестановки также являются инверсиями друг друга.

Зашифрованные данные: $C = c0b7a8d05f3a829c_{16}$.

После первоначальной перестановки: $19ba9212cf26b472_{16}$.

После разбиения на левую и правую секции: $L_0 = 19ba9212_{16}$ и $R_0 = cf26b472_{16}$.

После объединения: $L_{16}||R_{16} = 18ca18ad14a7d678_{16}$.

В результате выполнения конечной перестановки получим расшифрованные данные:

$$L_{16}||R_{16} = 18ca18ad14a7d678_{16}.$$

5.7. АНАЛИЗ АЛГОРИТМА DES

DES был подвергнут тщательному анализу, чтобы измерить интенсивность некоторых желательных свойств в блочном шифре. Элементы *DES* прошли исследования на соответствие некоторым критериям.

5.7.1. Лавинный эффект и эффект полноты DES

Два желательных свойства блочного шифра – *лавинный эффект* и *законченность (эффект полноты)*.

Лавинный эффект означает, что небольшие изменения в исходных данных (или ключе шифра) могут вызвать значительные изменения в зашифрованных данных. Было доказано, что *DES* имеет все признаки этого свойства [20, 37, 40].

Пример 5.17. Чтобы проверить лавинный эффект в *DES*, попробуем зашифровать два блока исходных данных, которые отличаются только одним битом, с помощью одного и того же ключа шифра и определим разницу в числе бит в каждом раунде.

Первый случай:

- открытые данные: 0000000000000000_{16};
- ключ шифра: $23234513987aba23_{16}$;
- зашифрованные данные: $4789fd476e82a5f1_{16}$.

Второй случай:

- открытые данные: 0000000000000001_{16};
- ключ шифра: $23234513987aba23_{16}$;
- зашифрованные данные: $0a4ed5c15a63fea3_{16}$.

Хотя два блока исходного текста отличаются только самым правым битом, блоки зашифрованного текста отличаются на *29* бит. Это означает, что изменение приблизительно в *1,5 %* исходных данных создают изменение приблизительно *45 %* зашифрованных данных.

Табл. 5.18 показывает изменение в каждом раунде по сравнению L_i и R_i в двух указанных случаях. Можно увидеть, что существенные изменения возникают уже в третьем раунде.

Таблица 5.18

Число различных бит в примере 5.17

Раунд	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*	*16*
Разница в битах	*1*	*6*	*20*	*29*	*30*	*33*	*32*	*29*	*32*	*39*	*33*	*28*	*30*	*31*	*30*	*29*

Эффект полноты заключается в том, что каждый бит зашифрованных данных должен зависеть от многих битов исходных данных. Рассеивание и перемешивание, произведенное *E*-блоками перестановки и расширения, *S*-блоками замены и сжатия и *P*-блоками прямой перестановки в *DES*, указывают на очень сильный эффект полноты.

5.7.2. Критерии разработок DES

Многочисленные испытания *DES* показали, что он удовлетворяет некоторым из заявленных критериев. Ниже кратко обсуждаются некоторые проблемы разработок *DES*.

Проблемы разработки S-блоков

В разд. 3 были обсуждены общие критерии построения *S*-блоков. Обсудим критерии, выбранные для *DES*. Структура блоков обеспечивает перемешивание и рассеивание от каждого раунда до последующего. Согласно этому положению и некоторому анализу, можно упомянуть несколько свойств *S*-блоков замены и сжатия.

1. Входы каждой строки есть перестановки значений между *0* и *15*.

2. *S*-блоки - нелинейные. Другими словами, выход – не аффинное преобразование.

3. Если изменять один единственный бит на входе, на выходе будут изменены два или больше бита.

4. Если два входа *S*-блока отличаются только двумя средними битами (битами *3* и *4*), выходная информация должна отличаться по крайней мере двумя битами. Другими словами, $S(x)$ и $S(x \oplus 001100)$ должны отличаться по крайней мере двумя битами, где x – вход и $S(x)$ – выход.

5. Если два входа в *S*-блок отличаются первыми двумя битами (биты *1* и *2*) и последними двумя битами (*5* и *6*), два выхода должны быть различны. Другими словами, мы должны иметь следующее отношение: $S(x) \neq S(x \oplus 0011bc00)$, в котором b и c – произвольные биты.

6. Есть только 32 шестибитовые пары "вход-выход" (x_i и x_j), в которых $x_i \oplus x_j \neq 000000_2$. Эти *32* входных пары создают *32* пары слова выхода по *4* бита. Если создать какие-то различия между *32* выходами пар, $d = y_i \oplus y_j$, то из этих d должны быть одинаковыми не больше чем *8* пар.

7. Такой же критерий, как в пункте 6, применяется к трем *S*-блокам.

8. В любом *S*-блоке, если единственный входной бит сохраняется как константа (*0* или *1*), то другие биты изменяются случайно так, чтобы разности между числом нулей и единиц были минимизированы.

Проблемы разработки E-блоков и P-блоков

Между двумя рядами *S*-блоков (в двух последующих раундах), есть один *E*-блок перестановки и расширения (*32* на *48*) и один *P*-блок прямой перестановки (*32* на *32*). Эти два блока вместе обеспечивают рассеивание битов. Об общем принципе построения блока замены указано в разд. 3. Обсудим только прикладные блоки, используемые в *DES*. В структуре блоков *DES* были реализованы следующие критерии:

1. Каждый вход *S*-блока подключается к выходу другого *S*-блока (в предыдущем раунде).

2. Ни один вход к данному *S*-блоку не соединяется с выходом от того же самого блока (в предыдущем раунде).

3. Четыре бита от каждого *S*-блока идут в шесть различных *S*-блоков (в следующем раунде).

4. Ни один из двух битов выхода от *S*-блока не идет в тот же самый *S*-блок (в следующем раунде).

5. Выход S_{j-2}-блока замены переходит в один из первых двух битов S_j-блока замены (в следующем раунде).

6. Бит выхода от S_{j-1}-блока замены переходит в один из последних двух битов S_j-блока замены (в следующем раунде).

7. Выход S_{j+1}-блока замены переходит в один из двух средних битов S_j-блока замены (в следующем раунде).

8. Для каждого *S*-блока замены два бита выхода идут в первые или последние два бита *S*-блока замены в следующем раунде. Другие два бита выхода идут в средние биты *S*-блока замены в следующем раунде.

9. Если выход от S_j -блока замены переходит в один из средних битов в S_k -блока замены (в следующем раунде), то бит выхода от S_k-блока замены не может идти в средний бит S_j-блока замены. Если допустить, что $j = k$, то ни один средний бит *S*-блока замены не может идти в один из средних битов того же самого *S*-блока замены в следующем раунде.

Количество раундов

DES использует шестнадцать раундов шифра *Фейстеля*. Доказано [20, 37, 40], что после того как каждый блок исходных данных зашифрован восемь раундов, каждый бит зашифрованных данных – функция каждого бита исходного блока данных и каждого ключевого бита. Зашифрованные данные – полностью случайная функция исходных данных и ключа. Отсюда вроде бы следует, что восьми раундов должно быть достаточно для хорошего шифрования. Однако эксперименты показывают, что некоторые версии *DES* с менее чем шестнадцатью раундами более уязвимы к атакам знания исходных данных, чем к атаке “грубой силы”, которая требует использования шестнадцати раундов *DES*.

5.7.3. Слабые места в DES

В течение прошлых многих лет криптографическими аналитиками были найдены некоторые слабые (уязвимые) места в *DES*. Укажем на некоторые слабые места, которые были обнаружены в структуре шифра.

S-блоки замены и сжатия. В литературе указываются по крайней мере три проблемы *S*-блоков:

1. В S_4 -блоке три бита выхода могут быть получены тем же самым способом, что и первый бит выхода: дополнением некоторых из входных битов.

2. Два специально выбранных входа к массиву *S*-блока могут создать тот же самый выход.

3. Можно получить тот же самый выход в одном единственном раунде, изменяя биты только в трех соседних *S*-блоках.

Блоки начальной IP и конечной IP^{-1} перестановок. В структуре этих блоков была найдена одна загадка. Не ясно, почему проектировщики *DES* использовали начальную и конечную перестановки.

Эти перестановки не вносят никаких новых свойств с точки зрения безопасности.

Е-блок перестановки и расширения. В блоке перестановки и расширения первые и четвертые биты последовательностей на *4* бита повторяются.

5.7.4. Слабость в ключе шифра DES

Размер ключа шифра DES

Критики утверждают, что самая серьезная слабость *DES* – это размер ключа (*56* битов). Чтобы предпринять атаку "грубой силы" блока зашифрованных данных, злоумышленник должен проверить 2^{56} ключей:

1. Используя доступную сегодня технологию, можно проверить один миллион ключей в секунду. Это означает, что потребуется более чем две тысячи лет, чтобы выполнить атаку "грубой силы" на *DES*, используя компьютер только с одним процессором.

2. Если можно сделать компьютер с одним миллионом чипов процессоров (параллельная обработка), то можно проверить все множество ключей приблизительно за *20* часов. Когда был введен *DES*, стоимость такого компьютера была более чем несколько миллионов долларов, но она быстро снизилась. Специальный компьютер был создан в 1998 году, благодаря ему ключ был найден за *112* часов.

3. Компьютерные сети могут моделировать параллельную обработку. В 1977 году команда исследователей использовала *3500* компьютеров, подключенных к *Internet*, чтобы найти ключ *DES* за *120* дней. Множество ключей было разделено среди всех этих компьютеров, и каждый компьютер был ответственен за проверку части домена *DES*. Если *3500* связанных в сеть компьютеров смогли найти ключ через *120* дней, то общество из *42000* членов может найти ключ через *10* дней.

Приведенное выше показывает, что *DES* с размером ключа шифра *56* битов не обеспечивает достаточной безопасности.

Слабые ключи шифра DES

Четыре ключа из 2^{56} возможных ключей называются слабыми ключами. *Слабые ключи* – это одни из тех, которые после операции удаления проверочных бит (используя табл. 5.1) состоят из всех нулей или всех единиц или половины нулей и половины единиц. Такие ключи показаны в табл. 5.19.

Таблица 5.19

Слабые ключи шифра *DES*

Ключи до удаления проверочных бит и перестановки (64 бита)	*Действующие ключи (56 бит)*
$0101\ 0101\ 0101\ 0101_{16}$	$0000\ 000\ 0000\ 0000_{16}$
$1f1f\ 1f1f\ 1f1f\ 1f1f_{16}$	$0000\ 000\ ffff\ ffff_{16}$
$e0e0\ e0e0\ e0e0\ e0e0_{16}$	$ffff\ ffff\ 0000\ 0000_{16}$
$fefe\ fefe\ fefe\ fefe_{16}$	$ffff\ ffff\ ffff\ ffff_{16}$

Ключи раунда, созданные от любого из этих слабых ключей, – те же самые и имеют тот же самый тип, что и ключ шифра. Например, эти шестнадцать ключей раунда создают первый ключ, который состоит из всех нулей или всех единиц или наполовину из нулей и единиц.

Это происходит по той причине, что алгоритм генерирования ключей сначала делит ключ шифра на две половины. Смещение или перестановка блока не изменяют блок, если он состоит из всех нулей, или всех единиц, или наполовину из нулей и единиц.

В чем опасность использования слабых ключей? Если зашифровать блок данных слабым ключом и впоследствии еще раз зашифровать результат тем же самым слабым ключом, получается первоначальный блок данных. Процесс создает один и тот же первоначальный блок данных, если расшифровывать блок дважды. Другими словами, каждый слабый ключ есть инверсия самого себя:

$$E_k\,(E_k\,(M)) = M,$$

как это показано на рис. 5.12.

Слабых ключей надо избегать, потому что противник может легко распознать их на перехваченном шифре. Если после двух этапов расшифрования результат тот же самый, противник определяет, что он нашел ключ.

Пример 5.18. Попробуем применить первый слабый ключ из табл. 5.19, чтобы два раза зашифровать блок данных. После того как проведено два шифрования с тем же самым ключом, в результате получим блок данных. Обратите внимание, что при этом ни разу не использовался алгоритм расшифрования, а только проведено два раза зашифрование.

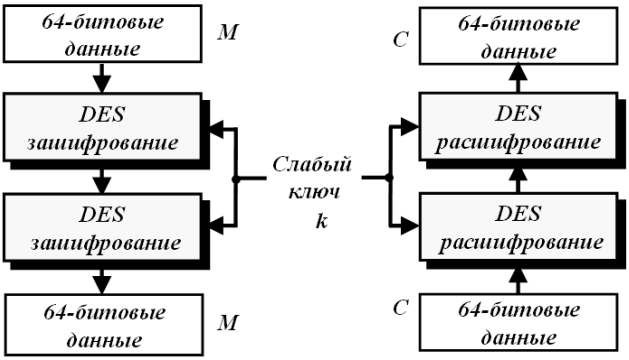


Рис. 5.12. Двойное зашифрование и расшифрование данных со слабым ключом

Ключ: 0101010101010101_{16}.
Исходный блок данных: 1234567887654321_{16}.
Зашифрованный блок данных: $814fe938589154f7_{16}$.
Ключ шифра: 0101010101010101_{16}.
Исходный блок данных: $814fe938589154f7_{16}$.
Зашифрованный блок данных: 1234567887654321_{16}.

Полуслабые ключи шифра DES

Имеются шесть ключевых пар, которые названы *полуслабыми ключами*. Эти шесть пар показаны в табл. 5.20 (формат на *64* бита перед удалением проверочных бит).

Полуслабые ключи создают только два различных ключа раунда и затем повторяют их восемь раз. Кроме того, ключи раунда, созданные от каждой пары, одни и те же в различном порядке.

Чтобы проиллюстрировать идею, создадим ключи раунда от первой пары, как показано табл. 5.21.

Таблица 5.20

Полуслабые ключи шифра *DES*

Первый ключ в паре	Второй ключ в паре
01fe 01fe 01fe 01fe$_{16}$	*fe01 fe01 fe01 fe01*$_{16}$
1fe0 1fe0 0ef1 0ef1$_{16}$	*e01f e01f f10e f10e*$_{16}$
01e0 01e1 01f1 01f1$_{16}$	*e001 e001 f101 f101*$_{16}$
1ffe 1ffe 0efe oefe$_{16}$	*fe1f fe1f feoe fe0e*$_{16}$
011f 011f 010e 010e$_{16}$	*1f01 1f01 0e01 0e01*$_{16}$
e0fe e0fe f1fe f1fe$_{16}$	*fee0 fee0 fef1 fefl*$_{16}$

Таблица 5.21

Раундовые ключи для полуслабых ключей шифра *DES*

Раунды	Раундовые ключи для первого ключа в паре	Раундовые ключи для второго ключа в паре
1	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
2	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$
3	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$
4	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$
5	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$
6	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$
7	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$
8	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$
9	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
10	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
11	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
12	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
13	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
14	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
15	*9153e54319bd*$_{16}$	*6eac1abce642*$_{16}$
16	*6eac1abce642*$_{16}$	*9153e54319bd*$_{16}$

Как показывает анализ табл. 5.21, имеется восемь одинаковых ключей раунда в каждом полуслабом ключе. Кроме того, ключи раунда *1* в первом множестве те же самые, что и ключи раунда *16* во втором; ключи раунда *2* в первом те же самые, что и ключи раунда *15* во втором, и так далее. Это означает, что ключи инверсны друг другу:

$$E_{k_2}(E_{k_1}(M))=M,$$

как показано на рис. 5.13.

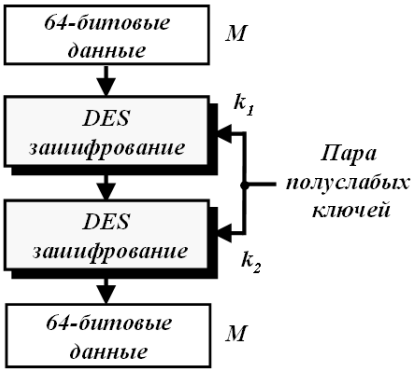


Рис. 5.13. Двойное зашифрование данных с полуслабым ключом

Возможно слабые ключи шифра DES

Также имеется *48* ключей, которые называются *возможно слабыми ключами*. *Возможно слабый ключ* создает только четыре различных ключа раунда; другими словами, шестнадцать ключей раундов разделены на четыре группы, и каждая группа состоит из четырех одинаковых ключей раунда.

Пример 5.19. Какова вероятность случайного выбора слабого, полуслабого или возможно слабого ключа?

Решение. Множество ключей *DES* равно 2^{56}. Общее количество вышеупомянутых ключей – *64: 4 + 12 + 48*. Вероятность выбора одного из этих ключей равна $8{,}8 \cdot 10^{-16}$, т.е. исключительно мала.

Ключевое дополнение DES

Среди множества ключей (2^{56}) некоторые ключи могут быть получены инверсией (изменение из *0* в *1* или *1* в *0*) каждого бита в ключе. *Ключевое дополнение* упрощает процесс криптографического анализа. Злоумышленник может использовать только половину возможных ключей (2^{56}), чтобы выполнить атаку “грубой силы”, потому что:

$$M \oplus M_Д = \text{ffffffffffffffff}_{16}, \quad K \oplus K_Д = \text{ffffffffffffffff}_{16}$$

и

$$C \oplus C_Д = \text{ffffffffffffffff}_{16}.$$

Другими словами, если зашифровать дополнение исходного блока данных дополнением ключа, получится *дополнение зашифрованного блока данных*. Злоумышленник не должен проверять все 2^{56} возможных ключей, он может проверить только половину из них и затем дополнить результат.

Пример 5.20. Проверим эти сведения о ключевое дополнение. Воспользуемся произвольным ключом и исходным блоком данных, для того чтобы найти соответствующий зашифрованный блок данных. Если имеется ключевое дополнение и исходный блок данных, то получится и дополнение предыдущего зашифрованного блока данных (табл. 5.22).

Таблица 5.22

Результаты примера 5.20

	Оригинал ключа	Ключевое дополнение
Ключ	1234123412341234_{16}	$edcbedcbedcbedcb_{16}$
Исходный блок данных	$12345678abcdef12_{16}$	$edcba987543210ed_{16}$
Зашифрованный блок данных	$e112be1defc7a367_{16}$	$1eed41e210385c98_{16}$

Кластерный ключ шифра DES

Кластерный ключ рассматривает ситуации, в которых два или более различных ключа шифра создают один и тот же зашифрованный блок данных из одного и того же исходного блока данных. Очевидно, каждая пара полуслабых ключей – *ключевой кластер*. Однако больше кластеров не было найдено.

5.8. МНОГОКРАТНОЕ ПРИМЕНЕНИЕ DES

Как уже было отмечено, основная критика *DES* направлена на длину ключа шифра. Возможные технологии и возможности параллельных процессоров делают реальной атаку “грубой силы”.

Одно из решений для улучшения безопасности – это отказ от *DES* и разработка нового алгоритма шифрования. Это решение – применении алгоритма шифрования *Advanced Encryption Standard* (*AES*). Будет рассмотрен в разделе 8.

Второе решение – многократное (каскадное) применение алгоритма шифрования *DES* с разными ключами шифра. Это решение, которое использовалось некоторое время, не требовало существен-

ных инвестиций в новое программное обеспечение и аппаратные средства. Рассмотрим такой подход.

Как известно из раздела 3, подстановка, которая размещает все возможные входы во все возможные выходы, является группой с отображениями элементов множества и набором операций. В этом случае использование двух последовательных отображений бесполезно, потому что можно всегда найти третье отображение, которое эквивалентно композиции этих двух (*свойство замкнутости*). Это означает, что если *DES* – группа, то однократный *DES* с ключом k_3 делает то же самое (см. рис. 5.14).

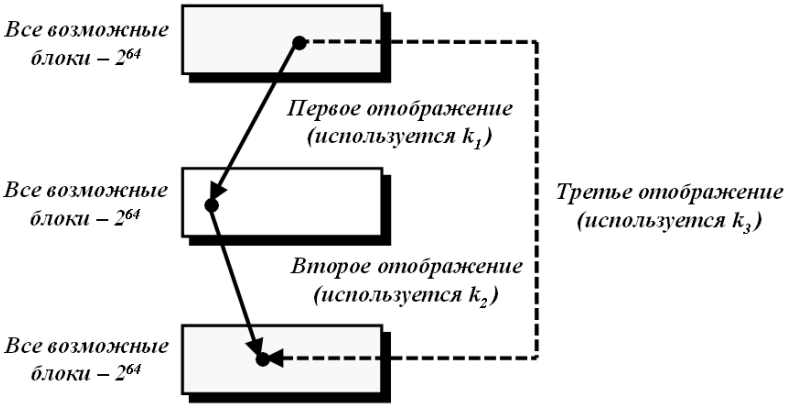


Рис. 5.14. Композиция отображений

К счастью *DES* не группа. Он базируется на следующих двух параметрах:

1. Номер возможных входов или выходов в *DES* – $N = 2^{64}$. Это означает, что $N! = (2^{64}!) = 10\ 347\ 380\ 000\ 000\ 000\ 000\ 000$ отображений. Один из способов сделать *DES* группой – это поддержать все эти отображения с размером ключа $log_2\ (2^{64}!) \approx 270$ битов. Так как длина ключа шифра *DES* – *56* бит, то это только маленькая часть требуемого огромного ключа.

2. Другой способ сделать *DES* группой – нужно, чтобы множество отображений было подмножеством множества в смысле зависимости от первого параметра. Было доказано [20, 21, 22], что группы, созданные из группы с помощью первого параметра, имеют ключевой размер *56* битов.

Если *DES* не является группой, то очень маловероятно, что можно найти ключ k_3, такой, что:

$$E_{K_2}(E_{K_1}(M)) = E_{K_3}(M).$$

Это означает, что можно использовать двукратные или трехкратные *DES*, чтобы увеличить размер ключа.

5.8.1. Двукратный DES

Первый подход состоит в том, чтобы использовать *двукратный DES* (*Double DES*). При этом подходе применяется два типа прямых шифров *DES* для зашифрования и два типа обратных шифров для расшифрования. Каждый тип использует различный ключ шифра, это означает, что размер ключа в данном случае удваивается (становится *112* битов). Однако двукратный *DES* уязвим к атаке знания открытых данных, как это будет далее обсуждено.

На первый взгляд двукратные *DES* увеличивают число испытаний при поиске ключа от 2^{56} (в однократном *DES*) к 2^{112} (в двукратном *DES*). Однако при использовании атаки знания исходных данных, называемой *атакой "встречи по средине"*, можно доказать, что двукратный *DES* улучшает эту устойчивость (только до 2^{57} по испытаниям), но не чрезвычайно (к 2^{112}).

Рис. 5.15. показывает диаграмму для двукратного *DES*.

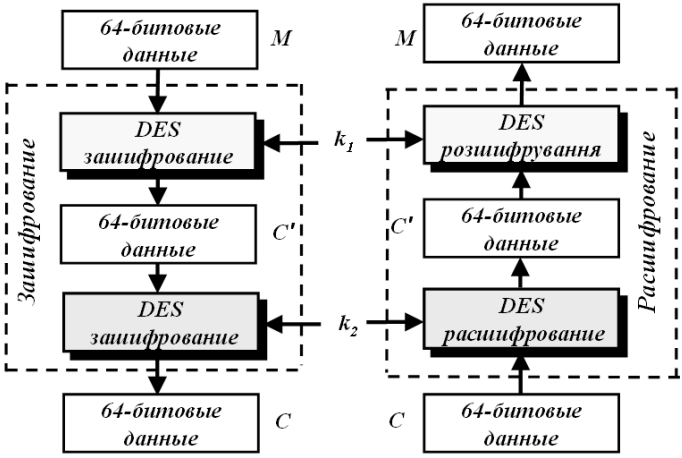


Рис. 5.15 Атака "встреча посредине" в двукратном *DES*

Отправитель использует два ключа шифра, k_1 и k_2, чтобы зашифровывать исходный блок данных M в зашифрованный блок данных C; получатель использует зашифрованный блок данных C и два ключа шифра, k_1 и k_2, для восстановления исходного блока данных M.

В средней точке C' – блок данных, созданный первым зашифрованием или первым расшифрованием. Для обеспечения правильной работы он должен быть одинаковым для зашифрования и расшифрования. Другими словами, имеем два соотношения:

$$C' = E_{K_1}(M) \quad \text{и} \quad C' = E_{K_2}(C).$$

Предположим, что злоумышленник перехватил предыдущую пару M и C (атака знания исходных данных). Базируясь на первом соотношении из упомянутых выше, злоумышленник зашифровывает M, используя все возможные значения (2^{56}) ключа k_1, и записывает все значения, полученные для C'. Базируясь на вторых соотношениях, упомянутых выше, злоумышленник расшифровывает C', используя все возможные значения (2^{56}) ключа k_2. Он записывает все значения, полученные для C'. Далее злоумышленник создает две таблицы, отсортированные согласно значениям C'. Он сравнивает значения для C', пока не находит те пары k_1 и k_2, для которых значение C' является одним и тем же в обеих таблицах (как показано на рис. 5.16).

Обратите внимание, что должна быть по крайней мере одна пара, потому что она делает исчерпывающий поиск комбинации двух ключей:

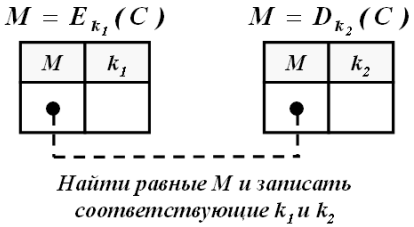


Рис. 5.16. Таблицы для атаки “встреча посредине”

1. Если есть только одно соответствие. Злоумышленник нашел два ключа (k_1 и k_2). Если есть больше чем один кандидат, злоумышленник переходит к следующему шагу.

2. Злоумышленник берет другую перехваченную пару зашифрованного блока данных и исходного блока данных и использует каждого кандидата для получения пары ключей, чтобы установить, может ли он получить зашифрованный блок данных из исходного блока данных. Если он находит больше чем одного кандидата в ви-

де пары ключей, он повторяет шаг 2, пока, наконец, не находит уникальную пару.

Было доказано [20, 21, 22], что после применения второго шага к нескольким перехваченным парам "*зашифрованные данные – исходные данные*" ключи были найдены. Это означает, что вместо того чтобы использовать поиск ключей шифра с помощью 2^{112} испытаний, злоумышленник использует 2^{56} испытаний поиска ключа и проверяет два раза (несколько больше испытаний требуется, если найден на первом шаге единственный кандидат). Другими словами, двигаясь от однократного *DES* до двукратного *DES*, увеличивается объем испытаний от 2^{56} до 2^{57} (а не до 2^{112}, как это кажется при поверхностном подходе).

5.8.2. Трехкратный DES

Для того чтобы улучшить безопасность *DES*, был предложен *трехкратный DES* (*Triple DES*). Он использует три каскада *DES* для зашифрования и расшифрования. Сегодня используются две версии трехкратных *DES*: *трехкратный DES с двумя ключами шифра* и *трехкратный DES с тремя ключами шифра*.

Трехкратный DES с двумя ключами шифра

В *трехкратном DES* с двумя ключами шифра есть только два ключа шифра: k_1 и k_2. Первый и третий каскады используют k_1; второй каскад использует k_2. Чтобы сделать трехкратный *DES* совместимым с *DES*, средний каскад применяет расшифрование (обратный шифр) на стороне зашифрования и зашифрование (прямой шифр) на стороне расшифрования. Таким способом сообщение, зашифрованное *DES*-ключом шифра k, может быть расшифровано трехкратным *DES*, если только $k_1 = k_2 = k$. Хотя трехкратный *DES* с двумя ключами шифра также уязвим при атаке "знания исходных данных", он гораздо устойчивее, чем двукратный *DES*. Он был принят в свое время для банков.

Рис. 5.17 показывает трехкратный *DES* с двумя ключами шифра.

Трехкратный DES с тремя ключами шифра

Возможность атак "знания исходных данных" при трехкратном *DES* с двумя ключами шифра вынудила некоторые приложения ис-

пользовать трехкратный *DES* с тремя ключами шифра. Алгоритм может применять три каскада прямого шифра *DES* на стороне зашифрования и три каскада обратных шифров на стороне расшифрования (рис. 5.18).

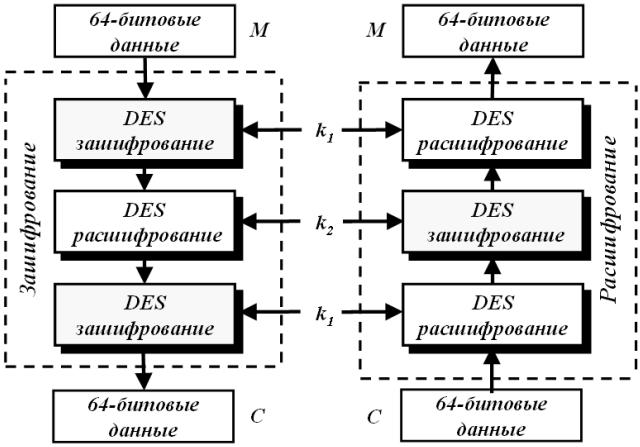


Рис. 5.17. Трехкратный *DES* с двумя ключами шифра

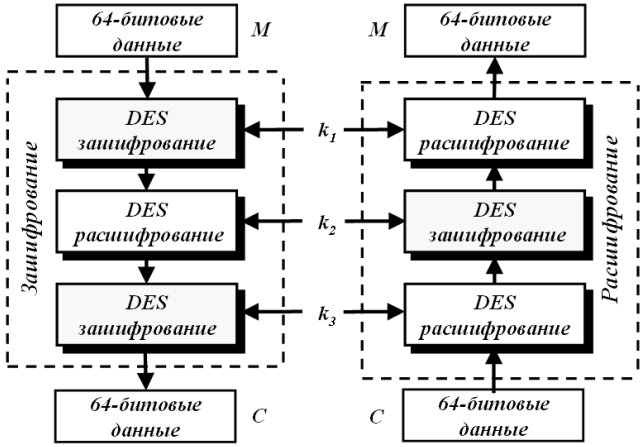


Рис. 5.18. Трехкратный *DES* с тремя ключами шифра

Для совместимости с однократным *DES* сторона зашифрования использует *EDE*, а сторона расшифрования – *DED*. *E* (*encryption*) – каскад зашифрования, *D* (*decryption*) – каскад расшифрования.

Общая длина ключа шифра в таком методе возрастает (*112 + + 56=168* бит).

Совместимость с однократным *DES* обеспечивается при $k_1 = k$ и установкой k_2 и k_3 к одному и тому же произвольному ключу, выбранному приемником. Трехкратный *DES* с тремя ключами шифра используется многими приложениями, такими как *PGP* (*Pretty Good Privacy – достаточно "хорошая" конфиденциальность*), поскольку обеспечивает высокую защищенность свойства информации – конфиденциальность.

Трехкратный *DES* является достаточно популярной альтернативой *DES* и используется при управлении ключами в стандартах *ANSI X9.17* и *ISO 8732* и *ISO 8732*. Некоторые криптографические аналитики предлагают для еще более надежного шифрования использовать пятикратный *DES* с тремя или пятью ключами.

5.9. БЕЗОПАСНОСТЬ DES

DES, как первый блочный шифр, имеющий важное значение, прошел через много испытаний на безопасность. Среди предпринятых атак лишь три представляют интерес: атака "*грубой силы*", дифференциальный и линейный криптографические анализы.

5.9.1. Атака "грубой силы" DES

Уже обсуждалась слабость *DES* с коротким ключом шифра. Слабость ключа шифра совместно с другими рассмотренными недостатками, делает очевидным, что *DES* может быть взломан с числом испытаний 2^{55}. Однако сегодня большинство приложений использует либо *Double DES* с двумя ключами шифра (размер ключа 2^{112}), либо *Triple DES* с тремя ключами шифра (размер ключа 2^{168}). Эти две многократных версии *DES* позволяют ему показывать существенную стойкость к атакам "грубой силы".

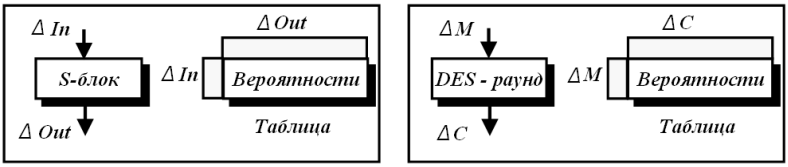
5.9.2. Дифференциальный криптографический анализ DES

В разделе 3 уже обсуждалась методика дифференциального криптографического анализа для современных блочных шифров. *DES* не является устойчивым к такому виду атаки. Однако многое указывает, что разработчики *DES* уже знали об этом типе атаки и проектировали *S*-блоки и специально выбрали число раундов,

чтобы сделать *DES* стойким к этому типу атаки. Сегодня показано, что *DES* может быть взломан, используя дифференциальный криптографический анализ, если имеется 2^{47} выборок исходных данных или 2^{55} известных исходных данных. Хотя это выглядит более эффективно, чем в атаке "грубой силы", предположить, что кто-то знает 2^{47} выборок исходных данных или 2^{55} выборок известных исходных данных, практически невозможно. Поэтому можно сказать, что *DES* является стойким к дифференциальному криптографическому анализу. Также показано, что увеличение числа раундов до *20* увеличивает число требуемых выборок исходного текста для атаки более чем 2^{64}. Такое увеличение невозможно, потому что число блоков исходного текста в *DES* только 2^{64}.

Дифференциальный криптографический анализ для *DES* был разработан *Бихамом* (*Biham*) и *Шамиром* (*Shamir*). В этом криптографическом анализе злоумышленник концентрируется на атаках с выборкой исходных данных. Анализ использует разность в прохождении различных входных сигналов через устройство или программу шифрования. Термин "разность" здесь применяется, чтобы рассмотреть с помощью операции *xor* несовпадение двух различных входных сообщений (исходных данных). Другими словами, злоумышленник анализирует, как $M \oplus M'$ различаются при обработке в каждом раунде.

Идея относительно дифференциального криптографического анализа базируется на вероятностных отношениях между разностями входа и разностями выхода. Два отношения представляют конкретный интерес в анализе: дифференциальный профайл и характеристика раунда, как это показано на рис. 5.19.



а) б)

Рис.5.19. Диференціальний профайл (а) і характеристика раунда (б) в *DES*

Дифференциальный профайл (он же профайл *xor*) показывает вероятностное отношение между разностями входа и разностями

выхода *S*-блока. Подобные профайлы могут быть созданы для каждого из восьми *S*-блоков в *DES*.

Характеристика раунда подобна дифференциальному профайлу, но вычисляется для целого раунда. Она показывает вероятность, с которой одна разность входа создала бы разность определенного выхода. Обратите внимание, что характеристика одна и та же для каждого раунда, потому что любое отношение, которое включает разности, не зависит от ключей раунда. Рис. 5.20 показывает четыре разные характеристики раунда.

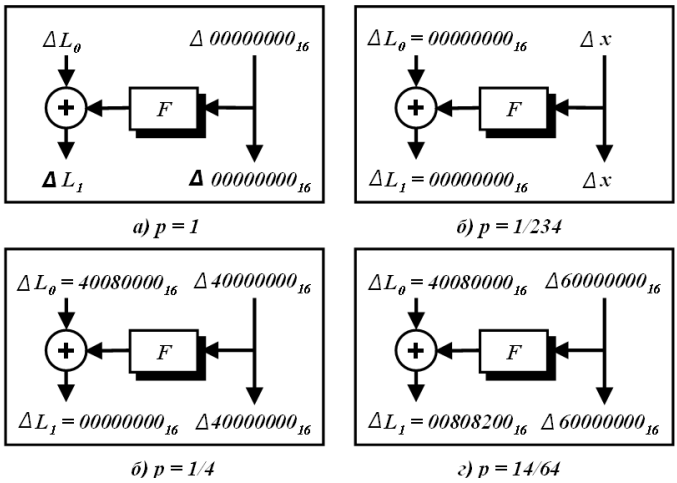


Рис. 5.20. Некоторые характеристики раунда для дифференциального криптографического анализа

Хотя существует много характеристик для раунда, рис. 5.20 показывает только четыре из них. В каждой характеристике разделены разности входа и разности выхода в левые и правые секции. Каждая левая или правая разности состоят из *32* битов или восьми шестнадцатеричных цифр.

Все эти характеристики могут быть найдены программами, которые могут найти отношение “входа-выхода” в раунде *DES*. Рис. 5.20, а показывает, что входная разность (x, 00000000_{16}) дает на выходе разность (x, 00000000_{16}) с вероятностью *1*. Рис. 5.20, б показывает ту же самую характеристику, как рис. 5.20, а, за исключением того, что левые и правые вход и выход поменялись местами; вероятность изменится чрезвычайно. Рис. 5.20, в показывает, что

разность входа (40080000_{16}, 04000000_{16}) дает разность выхода (00000000_{16}, 0400000_{16}) с вероятностью *1/4*.

Наконец, рис. 5.20, г показывает, что разность входа (00000000_{16}, 60000000_{16}) дает разность выхода (00808200_{16}, 6000000_{16}) с вероятностью *14/64*.

После создания и хранения одноraундных характеристик криптографический аналитик может комбинировать различное количество раундов, чтобы создать множественную характеристику раунда. Рис. 5.21 показывает случай трехраундного *DES*.

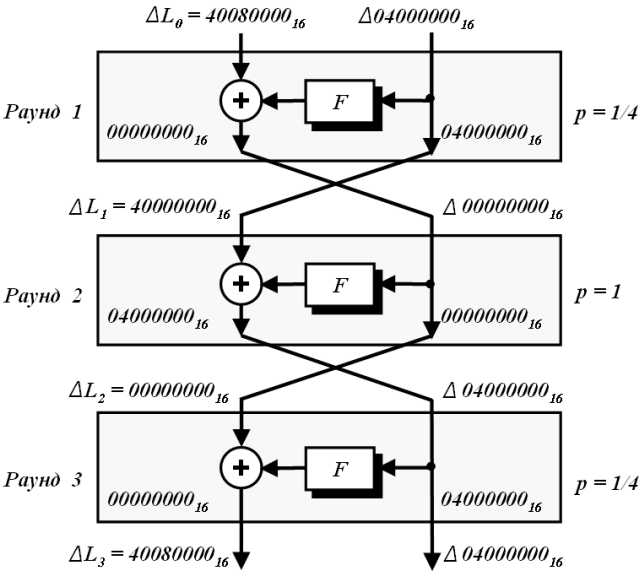


Рис. 5.21. Трехраундная характеристика для дифференциального криптографического анализа

На рис. 5.21, используется три смесителя и только два устройства замены, потому что последний раунд не нуждается ни в каком устройстве замены. Характеристики, показанные в смесителях первых и третьих раундов, те же самые, как и на рис. 5.20, б. Характеристика смесителя во втором раунде – та же самая, что и на рис. 5.20, а. Очень интересно отметить, что точки, в этом конкретном случае, разности "входа-выхода" – те же самые ($\Delta L_3 = \Delta L_0$ и $\Delta R_3 = \Delta R_0$).

Для шифра с шестнадцатью раундами можно скомпилировать много различных характеристик. Рис. 5.22 показывает пример.

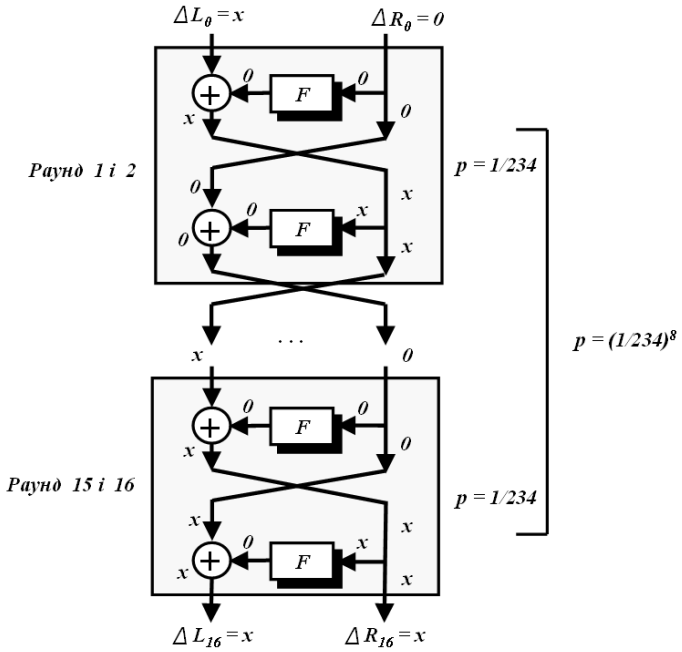


Рис. 5.22. Шестнадцатираундная характеристика для дифференциального криптографического анализа

На этом рис. 5.22 *DES* состоит из восьми секций с двумя раундами. Каждая секция использует характеристики на рис. 5.20, а и рис. 5.20, б. Ясно, что если последние раунды не имеют устройства замены, вход (*x, 0*) создает выход (*0, x*) с вероятностью $(1/234)^8$.

Для примера предположим, что злоумышленник использует характеристику по рис. 5.21, чтобы создать атаку на *DES* с шестнадцатью раундами. Злоумышленник каким-то способом провоцирует отправителя, чтобы зашифровать много исходных данных в форме (*x, 0*), в которой левая половина – *x* (различные значения) и правая половина – *0*. Злоумышленник затем сохраняет все зашифрованные данные, полученные от отправителя, в форме (*0, x*). Обратите внимание, что *0* здесь означает 00000000_{16}.

Окончательная цель злоумышленника в дифференциальном криптографическом анализе состоит в том, чтобы найти ключ шифра. Для этого нужно найти ключи каждого раунда от основания до вершины (k_{16} … k_1).

Если злоумышленник имеет достаточно много пар исходных данных/зашифрованных данных (каждая с различными значениями *x*), он может использовать отношения в последнем раунде, $0 = F(k_{16},x)$ и найти некоторые из битов в k_{16}. Это можно сделать, выбирая самые вероятные значения.

Ключи для других раундов можно найти, используя другие характеристики или применяя атаку "грубой силы".

Известно, что необходимы 2^{47} выборки пар исходных данных/зашифрованных данных, чтобы осуществить атаку на *DES* с *16* раундами. Найти такое огромное число выбранных пар чрезвычайно трудно в ситуациях реальной жизни. Это означает, что *DES* неуязвимы для этого типа атаки.

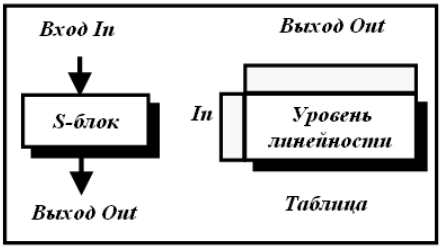
5.9.3. Линейный криптографический анализ DES

Методика линейного криптографического анализа для современных блочных шифров обсуждалась в разделе 3. Линейный криптографический анализ – более новая методика, чем дифференциальный криптографический анализ. *DES* более уязвим к применению линейного криптографического анализа, чем к дифференциальному криптографическому: вероятно потому, что этот тип атак не был известен проектировщикам *DES* и *S*-блоки не являются очень стойкими к линейному криптографическому анализу. Показано, что *DES* может быть взломан с использованием 2^{43} пары известных исходных данных. Однако с практической точки зрения перехват такого количества пар очень маловероятен.

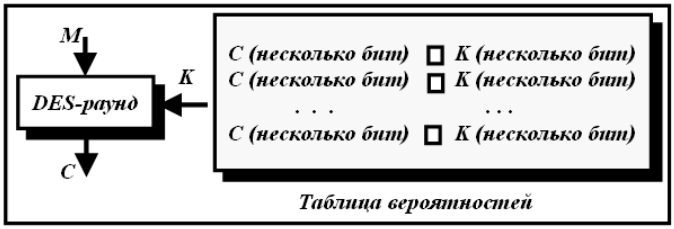
Линейный криптографический анализ для *DES* был разработан *Матцуи* [7, 20]. Это – атака знания исходных данных. Анализ использует распространение конкретного набора битов через устройство шифрования.

Линейный криптографический анализ основан на отношениях линейности. В этом типе криптографический анализ представляют интерес два набора отношений: линейные профайлы и характеристики раунда, как показано на рис. 5.23.

Линейный профайл показывает уровень линейности между входом и выходом *S*-блока. В *S*-блоке каждый бит выхода – функция всех входных битов. Желательное свойство в *S*-блоке достигнуто, если каждый бит выхода – нелинейная функция всех входных битов.



а) линейный профайл

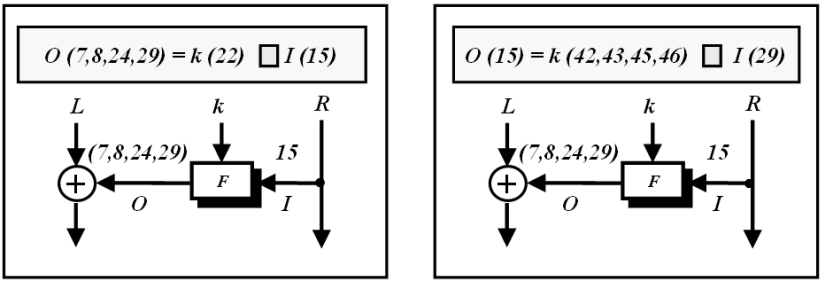


б) характеристика раунда

Рис. 5.23. Линейный профайл и характеристика раунда в *DES*

К сожалению, эта идеальная ситуация не существует в *DES*; некоторые биты выхода – линейная функция некоторых комбинаций входных битов. Другими словами, можно найти, что некоторые комбинации битов "входа-выхода" могут быть отображены между собой, используя линейную функцию. Линейный профайл показывает уровень линейности (или нелинейности) между входом и выходом. Криптографический анализ может создать восемь различных таблиц, по одной для каждого *S*-блока, в которых первый столбец показывает возможные комбинации входов по шесть бит, от 00_{16} до $3F_{16}$. Первая строка показывает возможные комбинации выходов по четыре бита, от 00_{16} до F_{16}. Входы показывают уровень линейности (или нелинейности) данного проекта. Не будем углубляться в детали того, как измеряется уровень линейности, но входы с высокого уровня из линейности интересны для криптографического анализа.

Характеристика раунда в линейном криптографическом анализе показывает комбинации входных битов, битов ключей раунда и битов выхода для того, чтобы определить линейное отношение. Рис. 5.24 изображает две различные характеристики раунда.



а) p = 52/64 *б) p = 42/64*

Рис. 5.24. Некоторые характеристики раунда для линейного криптографического анализа

Система обозначений, используемая для каждого случая, определяет биты, которые складываются по модулю *2*. Например, *O*(*7, 8, 24, 29*) означает операцию *xor 7, 8, 24* и *29*-го битов, выходящих из функции; *k(22)* означает *22*-й бит в ключе раунда; *I(15)* означает *15*-й бит, входящий в функцию.

Ниже показаны отношения для частей рис. 5.24, а и рис. 5.24, б, использующих индивидуальные биты:

- часть *а*: $O(7) \oplus O(8) \oplus O(24) \oplus O(29) = I(15) \oplus k(22)$;
- часть б: $F(15) = I(29) \oplus k(42) \oplus k(43) \oplus k(45) \oplus k(46)$.

После создания и хранения одноraундных характеристик аналитик может комбинировать различные раунды, чтобы создать множественную характеристику раунда. Рис. 5.25 показывает случай трехраундного *DES*, в котором раунды *1* и *3* используют одну и ту же характеристику, как это изображено на рис. 5.24, а, а в раунде *2* использована произвольная характеристика.

Цель линейного криптографического анализа состоит в том, чтобы найти линейное отношение между некоторыми битами в паре “исходные данные – зашифрованные данные” и ключ. Посмотрим, можно ли установить такое отношение для *DES* с тремя раундами, изображенной на рис. 5.25.

Раунд *1*: $R_1(7, 8, 24, 29) = L_0(7, 8, 24, 29) \oplus R_0(15) \oplus k_1(22)$.

Раунд *3*: $L_3(7, 8, 24, 29) = L_2(7, 8,24, 29) \oplus R_2(15) \oplus k_3(22)$.

Но L_2 – тот же самый, что и R_1, и R_2 – тот же самый, что и R_3. После замены L_2 на R_1 и R_2 на R_3 во втором отношении получим:

$$L_3(7, 8, 24, 29) = R_1(7, 8, 24, 29) \oplus R_3(15) \oplus k_3(22).$$

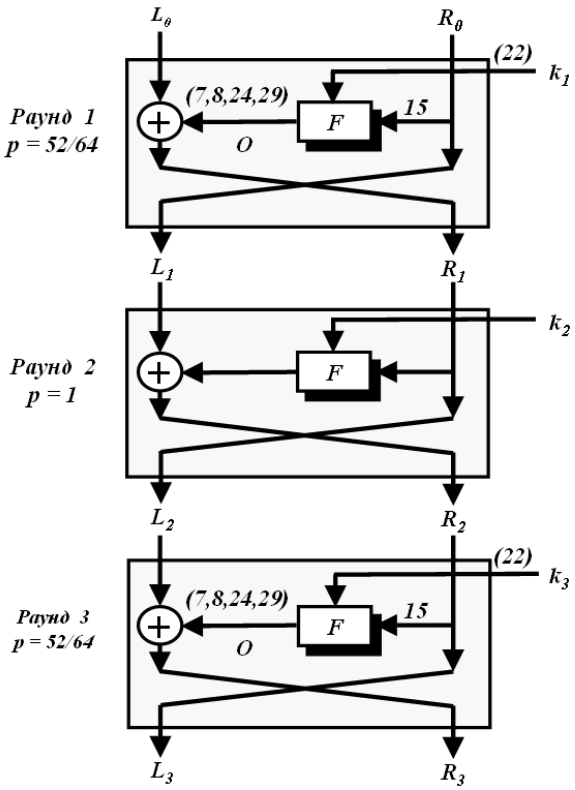


Рис. 5.25. Трехраундные характеристики для линейного криптографического анализа

Можно заменить R_1 на его эквивалентное значение в раунде *1*, в результате будем иметь:

$$L_3(7, 8, 24, 29) = L_0(7, 8, 24, 29) \oplus R_0(15) \oplus$$

$$\oplus k_1(22) \oplus R_3(15) \oplus k_3(22).$$

Это отношения между битами входа и выхода для всей системы из трех раундов после преобразований:

$$L_3(7, 8, 24, 29) \oplus R_3(15) = L_0(7, 8, 24, 29) \oplus$$

$$\oplus R_0(15) \oplus k_1(22) \oplus k_3(22).$$

Другими словами, имеем:

$$C(7, 8, 15, 24, 29) = M(7, 8, 15, 24, 29) \oplus k_1(22) \oplus k_3(22).$$

Один интересный вопрос: как найти вероятность трехраундных (или *n*-раундных) *DES*. *Матцуи* (*Matsui*) показал, что вероятность в этом случае [7, 32]

$$P = 0{,}5 + (2n-1)\prod_i (p_i - 0{,}5),$$

где n – количество раундов; p_i – вероятность каждой характеристики раунда; P – полная вероятность.

Например, полная вероятность для трехраундного анализа на рис. 5.24

$$P = 0{,}5 + (2\ 3 - 1)\ [(52/64 - 0{,}5) \times (1 - 0{,}5) \times (52/64 - 0{,}5)] = 0{,}695.$$

Шестнадцатираундная характеристика может также быть скомпилирована, чтобы обеспечить линейные отношения между некоторыми битами входных данных, некоторыми битами зашифрованных данных и некоторыми битами в ключах раунда:

$$C(\text{некоторые биты}) = M(\text{некоторые биты}) \oplus$$
$$\oplus\ k_1(\text{некоторые биты}) \oplus \ldots \oplus k(\text{некоторые биты}).$$

После нахождения и сохранения многих отношений между некоторыми битами исходных данных, битами зашифрованных данных и битами ключей раунда злоумышленник может обратиться к некоторым парам “исходных данных – зашифрованных данных” (атака знания исходных данных) и использовать соответствующие биты из сохраненных характеристик, чтобы найти биты в ключах раунда.

Известно, что для того чтобы создать атаку на *16*-раундовый *DES*, необходимы 2^{43} известных пар “исходных данных – зашифрованных данных”. Линейный криптографический анализ выглядит более вероятным, чем дифференциальный криптографический анализ, по двум причинам: первая – количество шагов у него меньше; вторая – он более прост для атаки знания исходных данных, чем для атаки с выборкой исходных данных. Однако и такая атака все еще далека от того, чтобы ее серьезно опасаться тем, кто работает с *DES*.

Контрольные вопросы и задания

1. Какой размер блока данных, размер ключа шифра и размер ключей раунда в *DES*?

2. Какое количество раундов в *DES*?

3. Сколько смесителей и устройств замены используется в первом способе зашифрования и обратного расшифрования? Какое их количество используется во втором способе?

4. Какое количество перестановок используется в алгоритме шифра *DES?*

5. Сколько операций *xor* используется в *DES*?

6. Почему в *DES* необходима операция, которая осуществляет перестановку с расширением?

7. Почему генератор ключей раунда *DES* требует удаления проверочных битов?

8. Какая разница между слабым, полуслабым и возможно слабым ключом шифра *DES*?

9. Что такое двухкратный *DES*? Какая атака двухкратного *DES* сделала его бесполезным?

10. Что такое трехкратный *DES*? Что такое трехкратный *DES* с двумя ключами шифра? Что такое трехкратный *DES* с тремя ключами шифра?

11. Значение последовательности входных данных в *DES* равно: $1234567890abcdef_{16}$. Определить значение последовательности на выходе блока *IP*-перестановки.

12. Значение последовательности R_0 в *DES* равно: $R_0 = f0aae8a5_{16}$. Определить значение последовательности на выходе *E*-блока перестановки и расширения.

13. Значение последовательности R_1 в *DES* равно: $R_1 = 116ba133_{16}$. Определить значение последовательности на выходе *E*-блока перестановки и расширения.

14. Значение последовательности на выходе *E*-блока перестановки и расширения в *DES* равно: $R_1^E = 8a2b57d029a6_{16}$. Значение раундового ключа: $k_1 = 4568581abcce_{16}$. Определить значение последовательности на выходе операции *xor* функции *Фейстеля* смесителя.

15. Значение последовательности на выходе *E*-блока перестановки и расширения в *DES* равно: $R_{12}^E = 9f3ca2bffea6_{16}$. Значение раундового ключа: $k_{12} = c2c1e96a4bf3_{16}$. Определить значение

последовательности на выходе операции *xor* функции Фейстеля смесителя.

16. Значение последовательности на выходе операции *xor* смесителя в *DES* равно: $R_{12}^{\oplus} = 5dfd4bd5b555_{16}$. Определить значения на выходе блоков замены и сжатия: а) S_2; б) S_3; в) S_4; г) S_5.

17. Значение последовательности на выходе операции *xor* смесителя в *DES* равно: $R_{13}^{\oplus} = e8848768_{16}$. Определить значение последовательности на выходе *S*-блоков замены и сжатия смесителя.

18. Значение последовательности на выходе *S*-блоков замены и сжатия смесителя в *DES* равно: $R_2^S = d5b80915_{16}$. Определить значение последовательности на выходе *P*-блока прямой перестановки смесителя.

19. Значение последовательности на выходе *P*-блока прямой перестановки смесителя в *DES* равно: $R_3^P = 07d0a03c_{16}$. Значение последовательности: $L_3 = 4f73c3b3_{16}$. Определить значение последовательности на выходе функции *xor* смесителя.

20. Значения последовательностей на входе пятого раунда зашифрования *DES* равно: $L_4 = 07eb4845_{16}$, $R_4 = 48a3638f_{16}$. Значение последовательности на выходе *P*-блока прямой перестановки смесителя: $R_4^P = 46932b6e_{16}$. Определить значения последовательностей *L* и *R* на выходе пятого раунда зашифрования *DES*.

21. Значения последовательностей на входе шестого раунда зашифрования *DES* равны: $L_5 = 48a3638f_{16}$, $R_5 = 4178632b_{16}$. Значение раундового ключа $k_6 = c1948e87475e_{16}$. Определить значения последовательностей L_6 и R_6 на выходе шестого раунда зашифрования *DES*.

22. Значение последовательности ключа шифра *DES* равно: $K = abba08192637cddc_{16}$. Определить значения последовательностей C_0 и D_0.

23. Значение последовательностей C_0 и D_0 шифра *DES* равны: $C_0 = c3c033a_{16}$ і $D_0 = 33f0cfa_{16}$. Определить значения последовательностей: а) C_2 и D_2; б) C_3 и D_3; в) C_7 и D_7; г) C_{12} и D_{12}.

24. Значение последовательностей C_0 и D_0 *DES* равны: $C_0 = c3c033a_{16}$ и $D_0 = 33f0cfa_{16}$. Определить значения раундового ключа k_2.

25. Значение последовательностей C_0 и D_0 *DES* равны: $C_0 = c3c033a_{16}$ и $D_0 = 33f0cfa_{16}$. Определить значения ключей:

а) четвертого раундового ключа;

б) девятого раундового ключа;

в) тринадцатого раундового ключа;

г) шестнадцатого раундового ключа.

26. Значения последовательностей на выходе шестнадцатого раунда зашифрования *DES* равны: $L_{16} = c95320e2_{16}$, $R_{16} = 9b7b3602_{16}$. Определить значение зашифрованных данных.

Раздел 6

РЕЖИМЫ ВЫПОЛНЕНИЯ КРИПТОГРАФИЧЕСКИХ АЛГОРИТМОВ БЛОЧНОГО СИММЕТРИЧНОГО ШИФРОВАНИЯ ДАННЫХ

Вскоре после *DES* в *США* был принят еще один федеральный стандарт, рекомендующий четыре способа эксплуатации (выполнения) алгоритма *DES* для шифрования данных. С тех пор эти режимы стали общепринятыми и применяются с любыми блочными шифрами. Позже был добавлен еще один режим.

Для любого симметричного блочного алгоритма шифрования определено пять режимов их выполнения:

1. *Режим электронной кодовой книги (Electronic Code Book – ECB)*. В этом режиме выполнения каждый блок незашифрованных данных шифруется независимо от остальных блоков, с применением одного и того же ключа шифрования. Типичные приложения – безопасная передача одиночных значений (например, криптографического ключа).

2. *Режим сцепления блоков шифрованных данных (Cipher Block Chaining – CBC)*. В этом режиме выполнения вход криптографического алгоритма является результатом применения операции *xor* к следующему блоку незашифрованных и предыдущему блоку зашифрованных данных. Типичные приложения – общая блочно-ориентированная передача, аутентификация.

3. *Режим обратной связи по зашифрованным данным (Cipher Feedback – CFB)*. В этом режиме выполнения при каждом вызове алгоритма обрабатывается *l* битов входного значения. Предшествующий зашифрованный блок используется в качестве входа в алгоритм; к *l* битам выхода алгоритма и следующему незашифрованному блоку из *l* битов применяется операция *xor*, результатом которой

является следующий зашифрованный блок из *l* битов. Типичные приложения - потокоориентированная передача, аутентификация.

4. *Режим обратной связи по выходу (Output Feed back – OFB).* Этот режим выполнения аналогичен режиму выполнения *CFB*, за исключением того, что на вход алгоритма при шифровании следующего блока подается результат шифрования предыдущего блока; только после этого выполняется операция *xor* с очередными *l* битами незашифрованных данных. Типичные приложения – потокоориентированная передача по зашумленному каналу (например, спутниковая связь).

5. *Режим счетчика* (*Counter Mode – CTR*). Этот режим выполнения очень похож на режим *OFB*, но вместо использования случайных уникальных значений вектора инициализации для генерации значений ключевого потока, этот режим использует счетчик, значение которого добавляется к каждому блоку открытого текста, который нужно зашифровать. Уникальное значение счетчика гарантирует, что каждый блок объединяется с уникальным значением ключевого потока.

6.1. РЕЖИМ ВЫПОЛНЕНИЯ “ЭЛЕКТРОННАЯ КОДОВАЯ КНИГА”

Режим электронной кодовой книги ECB является простейшим среди стандартных способов использования блочных шифров (рис. 6.1).

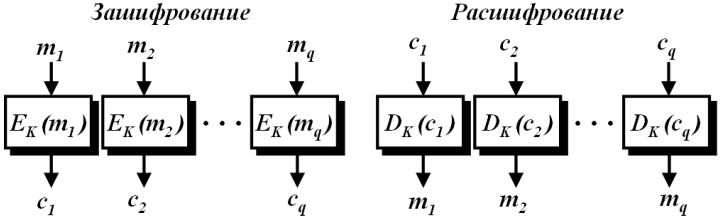


Рис. 6.1. Структура режима выполнения “Электронная кодовая книга”

Данные, которые предстоит зашифровать *M*, делятся на блоки заданной длинны m_i. Последний из них, при необходимости, дополняют. Каждый из этих блоков зашифровывают независимо с использованием одного и того же ключа шифрования

$$c_i = E_K(m_i), \tag{6.1}$$

где c_i – блоки зашифрованных данных; E – функция шифрования; K – ключ шифра.

Процесс расшифрования – простое обращение предыдущей операции (6.1)

$$m_i = D_K(c_i), \tag{6.2}$$

где D – функция расшифрования.

Основное достоинство данного режима – простота реализации. Однако с режимом *ECB* связан ряд проблем.

Первая проблема возникает из-за того, что при равенстве $m_i = m_j$ получаются одинаковые блоки $c_i = c_j$, т.е. одинаковые блоки на входе индуцируют совпадающие блоки на выходе. Это, действительно, проблема, поскольку шаблонные начало и конец сообщения совпадают, что дает криптографическому аналитику некоторую информацию о содержании сообщения.

Вторая проблема связана с тем, что удаление из сообщения какого-либо блока не оставляет следов, и атакующий может таким образом исказить передаваемую информацию.

Третья очень близка ко второй, но связана со вставкой блоков из других сообщений.

Чтобы лучше представить себе эти проблемы, возьмем простейшую модель шифра, в которой блок соответствует слову, и предположим, что открытое сообщение:

“*Плати Алисе сто фунтов, не плати Бобу двести фунтов*”

в зашифрованном виде выглядят так:

“*У кошки четыре ноги, а у человека две ноги*”.

Можно теперь заставить получателя оплатить Алисе две сотни фунтов, вместо одной, отправив ему сообщение

“*У кошки две ноги*”,

которое получено из первого заменой одного из блоков на блок из второго сообщения. Кроме того, можно приостановить выплаты Алисе, поставив блок *A* зашифрованного сообщения в начало первого сообщения. *A* – зашифрованный вариант частицы *не* открытого сообщения. Можно побудить получателя зашифрованного сообще-

ния выплатить *Бобу* двести фунтов, если удалить блок *A* из зашифрованного сообщения.

Таким атакам можно противостоять, добавляя контрольные суммы нескольких блоков открытых данных или используя режим, при котором к каждому блоку зашифрованных данных добавляется "*контекстный идентификатор*".

При зашифровании в режиме *ECB* ошибка в одном бите зашифрованных данных, появляющаяся на стадии передачи сообщения, повлияет на весь блок, в котором она допущена, и он, естественно будет расшифрован неверно, но это не влияет на остальные блоки расшифрованные данные. Однако, если бит зашифрованных данных случайно потерян или добавлен, то все последующие зашифрованные данные будут расшифрованы неправильно, если для выравнивания границ блоков не используется какая-нибудь кадровая структура.

Большинство сообщений точно не делится на *64*-битные (или любого другого размера, например *128* бит) блоки для зашифрования, в конце обычно используется укороченный блок. Режим же *ECB* требует использовать *64*-битные блоки. Способом решения этой проблемы является набивка блоков данных до заданной длинны.

Последний блок дополняется (набивается) некоторым регулярным шаблоном: нулями, единицами, чередующимися нулями и единицами – для получения полного блока. При необходимости удалить набивку после расшифрования в последний байт последнего блока записывается количество байтов (битов) набивки. Например, пусть размер блока – *64* бита и последний блок состоит из трех байт (*24* бита). Для дополнения блока до *64* бит требуется пять байтов. Поэтому необходимо добавить четыре байта набивки (например, нулей) и один байт длины набивки, который должен быть равен *6*. После расшифрования необходимо взять последний байт последнего блока и, определив длину набивки, удалить из последнего блока расшифрованных данных последние пять байтов. Чтобы этот метод работал правильно, каждое сообщение должно быть дополнено (набито), даже если открытые данные содержат целое число блоков. В таком случае придется добавить один полный блок, который будет иметь семь байт набивки и один байт длины набивки.

6.2. РЕЖИМ ВЫПОЛНЕНИЯ “СЦЕПЛЕНИЕ БЛОКОВ ЗАШИФРОВАННЫХ ДАННЫХ”

Одним из путей обхода проблем, возникающих при использовании режима *ECB*, состоит в “*сцеплении*” зашифрованных блоков данных, т.е. в добавлении к каждому зашифрованному блоку данных контекстного идентификатора. Самый простой способ сделать это – применить режим “*сцепления блоков шифрованных данных*” или *CBC* (рис. 6.2).

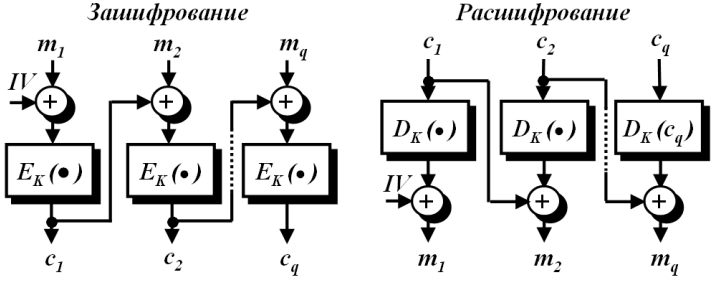


Рис. 6.2. Структура режима выполнения “Сцепление блоков шифрованных данных”

В этом режиме исходные данные для зашифрования, как обычно, разбиваются на серию блоков

$$m_1, m_2, m_3, ..., m_q,$$

где q – количество блоков исходных данных (с учетом набивки), которые подлежат зашифрованию.

Как и в предыдущем режиме, последний блок может потребовать дополнения, чтобы длина исходных данных стала кратна длине блока.

Зашифрование осуществляется согласно выражениям:

$$c_1 = E_K(m_1 \oplus IV), \quad c_i = E_K(m_i \oplus c_{i-1}), \quad i=2,3,...,q. \qquad (6.3)$$

При зашифровании первого блока участвует начальная величина *IV* (*initialization vector – вектор инициализации*), которую следует отнести к заданию функции зашифрования. Величину *IV* привлекают для зашифрования с тем, чтобы зашифрованные версии одинаковых частей исходных данных выглядели по разному.

Естественно величина *IV* участвует и при расшифровании. Этот процесс выглядит следующим образом:

$$m_1 = D_K(c_1) \oplus IV, \quad m_i = D_K(c_i) \oplus c_{i-1}, \quad i=2,3,\ldots,q. \qquad (6.4)$$

Очевидно, что последний *64*-битовый блок зашифрованных данных является функцией секретного ключа *K*, начального значения *IV* и каждого бита исходных данных независимо от его длины. Этот блок зашифрованных данных называют *кодом аутентификации сообщения* (*message authentication code – MAC*).

MAC может быть легко проверен после расшифрования данных получателем, владеющим секретным ключом *K* и начальным значением *IV*, путем повторения процедуры, выполненной отправителем. Посторонний, однако, не может осуществить генерацию *MAC*, который воспринялся бы получателем как подлинный, чтобы добавить его к ложному сообщению, либо отделить *MAC* от истинного сообщения для использования его с измененным или ложным сообщением.

Достоинство данного режима заключается в том, что он не позволяет накапливаться ошибкам при передаче. Как следует из (6.4) блок m_i, после расшифрования, является только функцией c_{i-1} и c_i. Поэтому ошибка при передаче приведет к потере только двух блоков исходных данных m_{i-1} и m_i, причем, если первый блок исходных данных будет потерян полностью, во втором же блоке будет искажен только один бит данных с номером позиции соответствующим номеру позиции искаженного бита.

Как следует из рисунка 6.2 для зашифрования данных используется дополнительно один блок случайных данных *IV*. Этот блок не имеет никакого смыслового значения, он используется только для того, чтобы сделать каждое сообщение уникальным. Хорошим *IV* служит метка времени или случайная последовательность битов.

С использованием *IV* сообщения с идентичными исходными данными при зашифровании переходят в сообщения с различными зашифрованными данными. Следовательно, злоумышленник не сможет предпринять вставку (удаление) или подмену блоков зашифрованных данных.

Требование уникальности *IV* для каждого передаваемого сообщения не является обязательным. Кроме того *IV* не должен хранится в секрете, он может передаваться открыто вместе с зашифрованными данными. Пусть передаваемое сообщение состоит из несколь-

ких блоков $m_1, m_2, \ldots, m_q$. m_1 зашифровывается с *IV*. m_2 зашифровывается с использованием зашифрованных данных m_1 в роли *IV*. m_3 зашифровывается с использованием зашифрованных данных m_2 в роли *IV*, и так далее. Итак, если количество блоков данных равно q, то $q-1$ "векторов инициализации" открыты, даже если первоначальный *IV* хранится в секрете. Поэтому причин хранить в секрете *IV* нет, *IV* – это просто блок-заглушка, можно считать его нулевым блоком сцепления m_0.

Набивка используется так же, как и в режиме *ECB*, но в некоторых приложениях размер зашифрованных данных должен в точности совпадать с размером исходных данных. Например, зашифрованный файл должен занять в точности тот же объем памяти, что и файл открытых данных. В этом случае последний короткий блок придется зашифровать иначе. Пусть последний неполный блок состоит из l битов. Зашифровав последний полный блок (рис. 6.3) необходимо снова произвести шифрование q-1–го блока зашифрованных данных. Затем, выбрав из полученного блока зашифрованных данных старшие (левые) l битов, выполнить для них и короткого блока операцию *xor*, создавая окончательно зашифрованные данные.

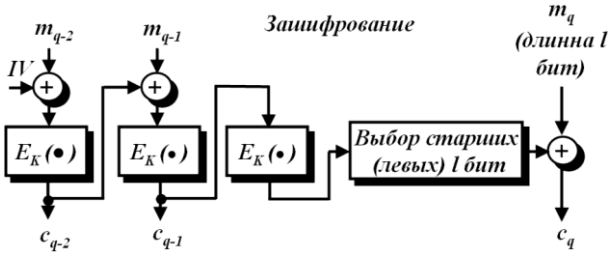


Рис. 6.3. Зашифрование короткого последнего блока в режиме *CBC*

6.3. РЕЖИМ ВЫПОЛНЕНИЯ "ОБРАТНАЯ СВЯЗЬ"

Режимы *ECB* и *CBC* предназначены для зашифрования и расшифрования блоков сообщений. В режиме *ECB* и *CBC* зашифрование данных не начнется, если не будет получен целый блок данных. Это создает проблемы для некоторых сетевых приложений. Например, в безопасной сетевой среде терминал (клиент) должен иметь возможность передать главному компьютеру (серверу) символ сразу, как только он будет введен. Если данные нужно обрабатывать байтами, режимы *ECB* и *CBC* не будут удовлетворять требованиям.

6.3.1. Режим выполнения "Обратная связь по зашифрованным данным"

Решение состоит в том, чтобы применить алгоритмы симметричного блочного шифрования или другие в режиме обратной связи по шифрованным данным (*CFB*). В этом режиме размер блока *n*, но размер блока исходных или зашифрованных данных – *l*, где $l < n$.

Идея состоит в том, что алгоритмы симметричного блочного шифрования используются не для того, чтобы зашифровать исходные или расшифровать зашифрованные данные, а для того, чтобы зашифровать или расшифровать содержание регистра сдвига, размером *n*. Шифрование сделано с применением операции *xor* к *l*-битовому блоку исходных данных с *l*-битовым регистром сдвига (рис. 6.4).

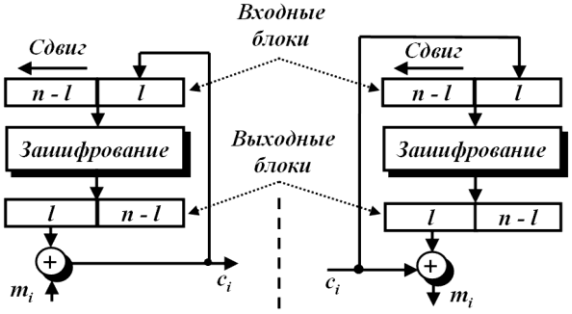


Рис. 6.4. Структура режима выполнения “Обратная связь по зашифрованным данным”

Входной блок (регистр сдвига влево) при зашифровании вначале содержит вектор инициализации, выровненный по правому краю.

Предположим, что в результате разбиения исходных данных на блоки, получено *q* блоков длиной *l* бит каждый. Тогда каждый *i*-й блок зашифрованных данных будет определяться следующим образом:

$$c_i = m_i \oplus g_{i-1}, \quad i=1,2,...,q, \tag{6.5}$$

где g_{i-1} – обозначает *l* старших бит предыдущего выходного блока зашифрованных данных.

Обновление регистра сдвига (входного блока) осуществляется путем сдвига его данных влево на *l* бит и записи c_i на место *l* младших (правых) разрядов.

Расшифрование сделано с применением операции *xor* к *l*-битовому блоку зашифрованных данных с *l*-битовым регистром сдвига. Входной блок (регистр сдвига влево) при расшифровании вначале содержит вектор инициализации, выровненный по правому краю. Обратите внимание, что содержание регистра сдвига для первого блока – это *IV* – не сдвигается.

Тогда каждый *i*-й блок расшифрованных (исходных) данных будет определяться следующим образом:

$$m_i = c_i \oplus g_{i-1}, \quad i=1,2,\ldots,q, \tag{6.6}$$

где g_{i-1} обозначает l старших бит предыдущего выходного блока расшифрованных данных.

Для каждого блока регистр сдвига выполняет сдвиг содержимого (предыдущий регистр сдвига) на l бит влево, заполняя самые правые l бит из c_{i-1}. Содержимое регистра сдвига зашифровывается и только самые левые l битов обрабатываются с помощью *xor* с зашифрованными данными, из блока c_i получая m_i. Обратите внимание, что содержание регистра сдвига для первого блока – это *IV* – не сдвигается.

Интересно, что в этом режиме не требуется дополнение блоков, потому что размер блоков, l, обычно выбирается так, чтобы удовлетворить размеру блока данных, который нужно зашифровать (например, символ). Интересное также другое – система не должна ждать получения большого блока данных (*64* бита или *128* битов) для того, чтобы начать зашифрование. Процесс зашифрования выполняется для маленького блока данных (таких, как символ). Эти два преимущества приводят к двум недостаткам. *CFB* менее эффективен, чем *ECB* или *CBC*, потому что он применяет зашифрование основным блочным шифром маленького блока размером l.

Как и режим *CBC*, режим *CFB* связывает вместе символы открытых данных так, что зашифрованные данные зависят от всех предшествующих исходных данных.

Как и в режиме *CBC IV* должно быть уникальным, кроме того, *IV* должен изменяться для каждого сообщения.

В режиме *CFB* ошибка в открытых данных влияет на все последующие зашифрованные данные, но самоустраняется при расшифровании. Гораздо интереснее ошибка в зашифрованных данных. Первым эффектом сбоя бита зашифрованных данных является сбой одного бита расшифрованных данных. Затем ошибка попадает

в регистр сдвига, и пока сбойный бит не выйдет из регистра, будут формироваться неправильные расшифрованные данные.

6.3.2. Режим выполнения "Обратная связь по выходу"

Режим выполнения *OFB* тоже использует переменный размер блока и регистр сдвига, инициализируемый так же, как в режиме *CFB*, а именно – входной блок вначале содержит начальное значение *IV*, выровненное по правому краю (рис. 6.5).

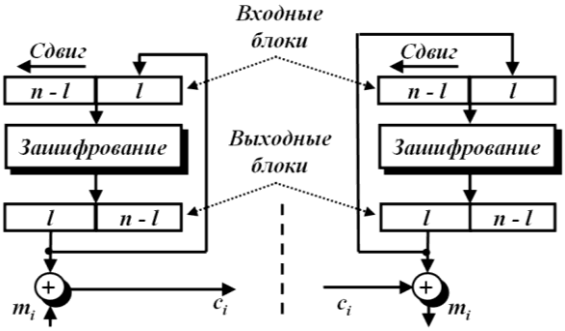


Рис. 6.5. Структура режима выполнения “Обратная связь по выходу”

При этом для каждого сеанса шифрования данных необходимо использовать новое начальное значение состояния регистра, которое должно пересылаться по каналу открытыми данными.

Положим, что открытое сообщение представляется в виде блоков

$$m_1, m_2, \ldots, m_q.$$

Для всех $i = 1,2,3,\ldots,q$

$$c_i = m_i \oplus g_{i-1}. \qquad (6.7)$$

Отличие от режима обратной связи по шифрованным данным состоит в методе обновления регистра сдвига. Это осуществляется путем отбрасывания старших l битов и дописывание справа g_i.

Основное преимущество режима *OFB* заключается в том, что если во время передачи произошла ошибка, то она не распространяется на последующие зашифрованные блоки, и тем самым сохраняется возможность расшифрования последующих блоков. Например, если появляется ошибочный бит в c_i, то это приведет только к не-

возможности расшифрования этого блока и получение m_i. Дальнейшая последовательность блоков будет расшифрована корректно.

Недостаток *OFB* в том, что он более уязвим к атакам модификации потока сообщений, чем *CFB*.

6.4. РЕЖИМ ВЫПОЛНЕНИЯ “СЧЕТЧИК”

В режиме счетчика (*CTR*) нет информации обратной связи. Псевдослучайный ключевой поток достигается с помощью счетчика. Счетчик на n бит инициализируется в заранее определенное значение (*IV*) и увеличивается по основному и заранее определенному правилу ($mod\ 2^n$). Чтобы обеспечивать случайность, величина приращения может зависеть от номера блока. Исходные и зашифрованные данные имеют один и тот же размер блока, как и основной шифр. Блоки размера n исходных данных зашифрованы так, чтобы создать зашифрованные данные с блоком размера n. На рис. 6.6 показано зашифрование данных в режиме счетчика.

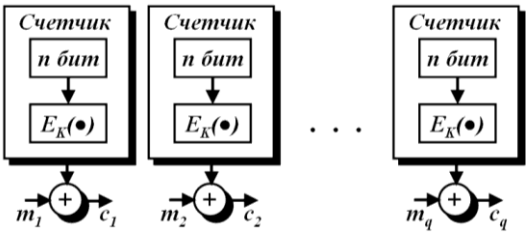


Рис. 6.6. Структура зашифрования данных в режиме выполнения “Счетчик”

Отношение между исходными данными и блоками зашифрованных данных показано ниже.

Зашифрование:

$$c_i = m_i \oplus E_{K_i}(n_i),$$

где n_i – n-битовое значение счетчика.

Расшифрование (рис. 6.7):

$$m_i = c_i \oplus E_{K_i}(n_i).$$

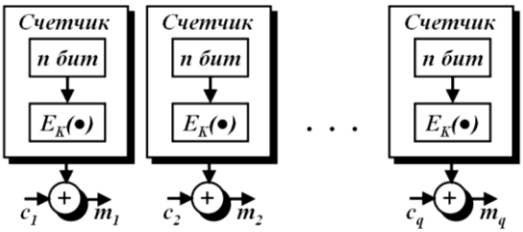


Рис. 6.7. Структура расшифрования данных в режиме выполнения "Счетчик"

Режим *CTR* использует функцию зашифрования основного блокового шифра (E_K) и для зашифрования, и для расшифрования. Достаточно легко доказать, что блок m_i исходных данных может быть восстановлен из зашифрованных данных c_i.

Можно сравнить режим *CTR* с режимами *OFB* и *ECB*. Подобно *OFB*, *CTR* создает ключевой поток, который независим от предыдущего блока зашифрованных данных, но *CTR* не использует информацию обратной связи. Так же как *ECB*, *CTR* создает *n*-битовые зашифрованные данные, блоки которого независимы друг от друга – они зависят только от значений счетчика. Отрицательной стороной этого свойства является то, что режим *CTR*, подобно режиму *ECB*, не может использоваться для обработки в реальном масштабе времени. Алгоритм шифрования ждет перед зашифрованием законченный *n*-разрядный блок данных. Положительная сторона этого свойства та, что режим *CTR*, подобно режиму *ECB*, может использоваться, чтобы зашифровать и расшифровать файлы произвольного доступа, и значение счетчика может быть связано номером записи в файле.

Проблемы безопасности для режима *CTR* те же, что и для режима *OFB*. Единственная ошибка в зашифрованных данных касается только соответствующего бита в расшифрованных данных.

6.5. ДРУГИЕ РЕЖИМЫ ШИФРОВАНИЯ

Существует также ряд дополнительных режимов шифрования данных с помощью блоковых симметричных криптографических алгоритмов [23]:

- *режим распространенного сцепления блоков зашифрованных данных* (*Propagating Cipher Block Chaining – PCBC*);

- *режим сцепления блоков зашифрованных данных с контрольной суммой* (*Cipher Block Chaining with Checksum – CBCC*);

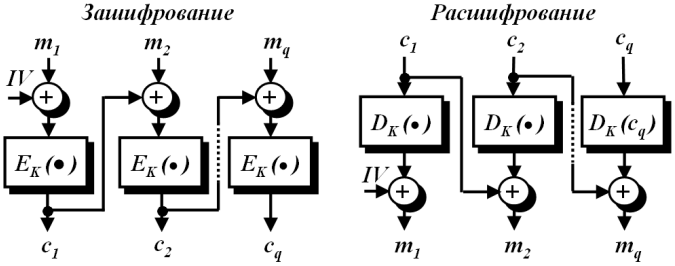
- *режим обратной связи по выходу с нелинейной функцией* (*Output Feedback With a Nonlinear Function – OFBNLF*);

- *режим обратной связи по открытым данным* (*PFB*);

- *режим сцепления блоков открытых данных* (*PBC)* и т.д.

Режим распространенного сцепления блоков зашифрованных данных. Одной из потенциальных проблем режима *CBC* является возможность внесения контролируемых изменений в следующий расшифрованный блок открытых данных. Например, если злоумышленник изменит один бит в блоке, то весь блок будет расшифровано неверно, но в следующем блоке появится ошибка в соответствующей позиции. Есть ситуации, когда это нежелательно. Для борьбы с этой угрозой открытые данные должны содержать определенную избыточность.

Режим *PCBC* подобный режиму *CBC*, за исключением того, что как предыдущий блок открытых данных, так и предыдущий блок зашифрованных данных подвергаются операции *xor* с текущим блоком открытых данных перед зашифрованием (рис. 6.8, а) или после расшифрования (рис. 6.8, б)



а) б)

Рис. 6.8. Структура шифрования данных в режиме *PCBC*:
а) зашифрования; б) расшифрования

В этом случае зашифрования и расшифрования данных осуществляется с использованием соотношений:

$$c_i = E_k\,(m_i \oplus c_{i-1} \oplus m_{i-1}),\ i=1,2,\ldots,q,\ c_0=IV;$$

$$m_i = c_{i-1} \oplus m_{i-1} \oplus D_k\,(c_i),\ i=1,2,\ldots,q,\ c_0=IV.$$

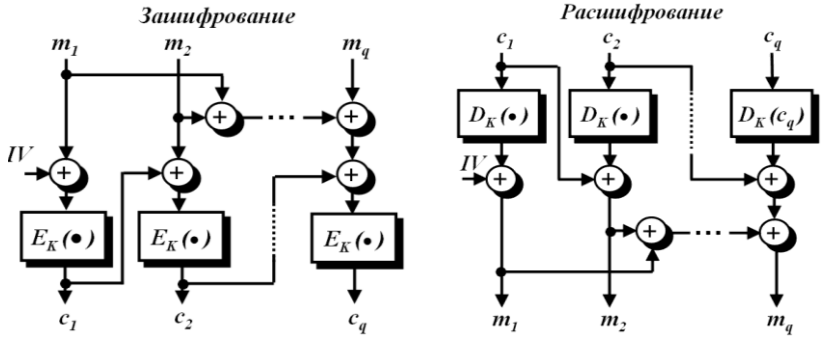
Режим *PCBC* используется в протоколе *Kerberos* для выполнения за один проход и шифрования, и проверки целостности данных. В режиме *PCBC* ошибка в зашифрованных данных тянет некорректное расшифрование всех последующих блоков.

Это означает, что проверка стандартного блока в конце сообщения используется для проверки целостности сообщений.

Недостаток этого режима так же заключается в том, что перестановка двух блоков зашифрованных данных приводит к некорректному расшифрованию двух соответствующих блоков открытых данных. Однако благодаря природе операции *xor* над открытыми и зашифрованными данными, дальнейшие ошибки компенсируются. Поэтому если проверка целостности охватывает лишь несколько последних блоков расшифрованных (открытых) данных, можно получить частично искаженное сообщение.

Это подозрительное свойство заставило разработчиков отказаться от данного режима в пользу *CBC* в следующей версии протокола *Kerberos*.

Режим сцепления блоков зашифрованных данных с контрольной суммой отличается от режима *CBC* только тем, что к последнему блоку исходных данных перед зашифрованием прибавляется сумма по модулю *2* для всех предыдущих блоков исходных данных (рис. 6.9).



а) б)

Рис. 6.9. Структура шифрования данных в режиме сцепления блоков зашифрованных данных с контрольной суммой: а) зашифрование; б) расшифрование

Это дает возможность проконтролировать целостность передаваемых данных с небольшими дополнительными накладными расходами.

В этом случае зашифрование и расшифрование данных осуществляется с использованием соотношений:

$$s = (m_1 \oplus m_{i+1} \oplus \ldots \oplus m_{q-1});$$

$$c_i = E_k (m_i \oplus c_{i-1}),\ i=1,2,\ldots,q-1,\ c_0=IV;$$

$$c_q = E_k (m_q \oplus s \oplus c_{q-1}),$$

а расшифрование – использованием соотношений:

$$m_i = D_k (c_i) \oplus c_{i-1},\ i=1,2,\ldots,q-1,\ c_0=IV;$$

$$s = (m_1 \oplus m_{i+1} \oplus \ldots \oplus m_{q-1});$$

$$m_q = E_k (c_q) \oplus c_{q-1} \oplus s).$$

Единичная ошибка в зашифрованных данных распространяется только на один блок открытых данных, однако нужна поддержка синхронизации.

Скорость обработки информации определяется не только скоростью шифрования базовым алгоритмом, но и скоростью обновления текущего значения ключа.

Режим сцепления блоков открытых данных (*Plaintext Block Chaining - PBC*) подобный *CBC* за исключением того, что операция *xor* выполняется над текущим и предыдущим блоком открытых данных, а не над блоком зашифрованных данных.

Режим с обратной связью по открытым данным (*Plaintext Feedback – PFB*) подобный режим *CFB*, однако для обратной связи используются не зашифрованные, а открытые данные. Оба режима допускают взлом с использованием специально подобранных открытых данных. С целью повышения устойчивости к взлому оба режима допускают использование открытых данных.

Контрольные вопросы и задания

1. Поясните, почему необходимы режимы работы, если для шифрования используются современные блочные шифры.

2. Перечислите основные режимы работы симметричных блочных шифров.

3. Определите режим *ECB* и перечислите его преимущества и недостатки.

4. Определите режим *CBC* и перечислите его преимущества и недостатки.

5. Определите режим *CFB* и перечислите его преимущества и недостатки.

6. Определите режим *OFB* и перечислите его преимущества и недостатки.

7. Определите режим *CTR* и перечислите его преимущества и недостатки.

8. Разделите основные режимы работы на две группы: те, которые используют функции зашифрования и расшифрования, – основные шифры (например, *DES*), и те, которые используют только функцию зашифрования.

9. Разделите основные режимы работы на две группы: те, которые требуют дополнения данных, и те, которые не требуют этого.

10. Разделите основные режимы работы на две группы: те, которые используют тот же ключ для шифрования всех блоков, и те, которые используют ключевой поток для шифрования блоков.

11. Перечислите режимы работы, которые могут быть ускорены параллельной обработкой.

12. Перечислите режимы работы, которые могут использоваться для зашифрования файлов произвольного доступа.

13. Покажите, почему режим *CFB* создает несинхронный шифр потока, а режим *OFB* – синхронный.

14. Сколько блоков затрагивает единственный бит ошибки при передаче в режиме *CFB*?

15. В режиме *ECB* ($n = 64$) в первом блока зашифрованных данных искажен один (*5-й*) бит в течение передачи. Определите возможные искажения бит в расшифрованных данных.

16. В режиме *CBC* ($n = 64$) в первом блока зашифрованных данных искажен один (*5-й*) бит в течение передачи. Определите возможные искажения бит в расшифрованных данных.

17. В режиме *CFB* ($l = 8$, $n = 64$) второй бит первого блока открытых данных искажен. Определите возможные искажения бит в расшифрованных данных.

18. В режиме *CFB* (*l = 8, n = 64*) второй бит первого блока зашифрованных данных искажен. Определите возможные искажения бит в расшифрованных данных.

19. В режиме *OFB* (*l = 8, n = 64*) второй бит первого блока открытых данных искажен. Определите возможные искажения бит в расшифрованных данных.

20. В режиме *CTR* (*n = 64*) второй бит второго блока зашифрованных данных искажен. Определите возможные искажения бит в расшифрованных данных.

21. Докажите, что исходные данные, используемые отправителем, могут быть восстановлены получателем в режиме *CFB*.

22. Докажите, что исходные данные, используемые отправителем, могут быть восстановлены получателем в режиме *OFB*.

23. Докажите, что исходные данные, используемые отправителем, могут быть восстановлены получателем в режиме *CTR*.

Раздел 7

СТАНДАРТ КРИПТОГРАФИЧЕСКОГО ПРЕОБРАЗОВАНИЯ ДАННЫХ ГОСТ 28147-89

7.1. ИСТОРИЯ РАЗРАБОТКИ ГОСТ 28147-89

Важной задачей в обеспечении гарантированной безопасности информации в информационных системах является разработка и использования стандартных алгоритмов шифрования данных. Первым среди подобных стандартов, как отмечалось в разд. 5, стал американский *DES*, представляющий собой последовательное использование замен и перестановок. В настоящее время все чаще говорят о неоправданной сложности и невысокой криптографической стойкости *DES*. На практике приходится использовать его модификации.

Более эффективным является отечественный стандарт криптографического преобразования данных *ГОСТ 28147-89* [12, 29 36].

Он рекомендован к использованию для защиты любых данных, представленных в виде двоичного кода, хотя не исключаются и другие методы шифрования. Данный стандарт формировался с учетом мирового опыта, и в частности, были приняты во внимание недостатки и нереализованные возможности алгоритма *DES*, поэтому использование стандарта *ГОСТ* предпочтительнее. Алгоритм достаточно сложен и ниже будет описана в основном его концепция.

Этот стандарт закреплен *ГОСТ 28147-89*, принятом, как показано из его обозначения, еще в 1989 году в *СССР* и введен в действие с 1 июля 1990 года [36]. Однако, без сомнения, история этого шифра гораздо более давняя. Стандарт родился предположительно в недрах восьмого главного управления *КГБ СССР*, преобразованного ныне в *ФАПСИ* (*Федеральное агентство правительственной связи и информации при Президенте РФ*). Еще в 70-х годах XX века

он имел гриф "*Совершенно секретно*", позже гриф был изменен на "*секретно*", затем снят совсем. К сожалению, в отличие от самого стандарта, история его разработки и критерии проектирования шифра до сих пор остаются "*тайной за семью печатями*" [2].

Описание стандарта шифрования *ГОСТ 28147-89* содержится в очень интересном документе, озаглавленном "*Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования ГОСТ 28147-89*" [36]. 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.

Стандарт криптографического преобразования данных *ГОСТ 28147-89* оперирует с данными длиной *64* бита. Зашифрование и расшифрование данных осуществляется с использованием либо *32* либо *16* раундов (циклов). Ключ шифра *256* бит, из которого формируются восемь подключей длиной по *32* бита (из этих восьми подключей формируются *32* (*16*) раундовых (цикловых) подключей).

Стандарт предусматривает четыре режима работы:

- шифрование данных в режиме простой замены;
- шифрование данных в режиме гаммирования;
- шифрование данных в режиме гаммирования с обратной связью;
- шифрование с целью выработки имитовставки.

7.2. МАТЕМАТИЧЕСКИЕ ПРЕДПОСЫЛКИ ГОСТ 28147-89

На различных шагах алгоритмов *ГОСТ 28147-89* данные, которыми они оперируют, интерпретируются и используются различным образом. В некоторых случаях элементы данных обрабатываются как массивы независимых битов, в других случаях – как целое число без знака, в третьих – как имеющий структуру сложный элемент, состоящий из нескольких более простых элементов. Поэтому

во избежание путаницы следует договориться об используемых обозначениях.

Элементы данных обозначаются заглавными латинскими буквами с наклонным начертанием (например, X). Через $|X|$ обозначается размер элемента данных X в битах. Таким образом, если интерпретировать элемент данных X как целое неотрицательное число, можно записать следующее неравенство: $0 \le X \le 2^{|X|}$.

Если элемент данных состоит из нескольких элементов меньшего размера, то этот факт обозначается следующим образом:

$$X = (X_0, X_1, \ldots, X_{n-1}) = X_0 \,||\, X_1 \,||\ldots||\, X_{n-1}\,.$$

Процедура объединения нескольких элементов данных в один называется *конкатенацией* данных и обозначается символом "||". Естественно, для размеров элементов данных должно выполняться следующее соотношение:

$$|X| = |X_0| + |X_1| + \ldots + |X_{n-1}|.$$

При задании сложных элементов данных и операции конкатенации составляющие элементы данных перечисляются в порядке возрастания старшинства. Иными словами, если интерпретировать составной элемент и все входящие в него элементы данных как целые числа без знака, то можно записать следующее равенство:

$$X = (X_0, X_1, \ldots, X_{n-1}) = X_0 \,||\, X_1 \,||\ldots||\, X_{n-1} =$$

$$= X_0 + 2^{|X_0|}\cdot(X_1 + 2^{|X_1|}\cdot(X_2 + 2^{|X_2|}\cdot(\ldots 2^{|X_{n-2}|}\cdot X_{n-1})\ldots)).$$

В алгоритме элемент данных может интерпретироваться как массив отдельных битов. В этом случае биты обозначаются той же самой буквой, что и массив, но в строчном варианте, как показано на следующем примере:

$$X = (x_0, x_1, \ldots, x_{n-1}) = x_0\cdot 2^0 + x_1\cdot 2^1 + \ldots x_{n-1}\cdot 2^{n-1}.$$

Таким образом, для *ГОСТ*а принята так называемая "*little-endian*" нумерация разрядов, т.е. внутри многоразрядных слов данных отдельные двоичные разряды и их группы с меньшими номерами являются менее значимыми. Об этом прямо говорится: "*При сложении и циклическом сдвиге двоичных векторов старшими разрядами считаются разряды накопителей с большими номерами*"

[36]. Алгоритм *ГОСТ 28147-89* предписывает начинать заполнение данными регистров-накопителей виртуального шифрующего устройства с младших, т.е. менее значимых разрядов. Точно такой же порядок нумерации принят в микропроцессорной архитектуре *Intel x86*, именно поэтому при программной реализации шифра на данной архитектуре никаких дополнительных перестановок разрядов внутри слов данных не требуется.

Если над элементами данных выполняется некоторая операция, имеющая логический смысл, то предполагается, что данная операция выполняется над соответствующими битами элементов. Иными словами,

$$A \bullet B = (a_0 \bullet b_0,\ a_1 \bullet b_1,\ a_{n-1} \bullet b_{n-1}),$$

где $n = |A| = |B|$, а символом "$\bullet$" обозначается произвольная бинарная логическая операция; как правило, имеется в виду операция *xor*, она же – операция суммирования по модулю *2*:

$$a \oplus b = (a + b) \bmod 2.$$

Кроме того, в алгоритме используется операция сложения двух *32*-разрядных целых положительных чисел по модулю 2^{32} (обозначается $\boxplus$) и $2^{32}-1$ ($\boxplus$).

Два целых числа a и b, где $0 \le a < 2^{32}$, $0 \le b < 2^{32}$:

$$a = (a_{32}, a_{31}, ..., a_2, a_1) \quad \text{i} \quad b = (b_{32}, b_{31}, ..., b_2, b_1),$$

представленные в двоичном виде, т.е.

$$a = a_{32} \cdot 2^{31} + a_{31} \cdot 2^{30} + ... + a_2 \cdot 2^1 + a_1 \cdot 2^0;$$

$$b = b_{32} \cdot 2^{31} + b_{31} \cdot 2^{30} + ... + b_2 \cdot 2^1 + b_1 \cdot 2^0,$$

суммируются по модулю 2^{32} (операция $\boxplus$) по следующему правилу:

$$a \boxplus b = a + b, \text{ если } a + b < 2^{32};$$

$$a \boxplus b = a + b - 2^{32}, \text{ если } a + b \ge 2^{32}.$$

Два целых числа a и b, где $0 \le a < 2^{32}-1$, $0 \le b < 2^{32}-1$, представленные аналогично, суммируются по модулю 2^{32} (операция $\boxplus$) по следующему правилу:

$$a \boxplus b = a + b, \text{ если } a + b < 2^{32}\text{-}1,$$

$$a \boxplus b = a + b - (2^{32}\text{-}1), \text{ если } a + b \geq 2^{32}\text{-}1.$$

7.3. ШИФРОВАНИЕ ДАННЫХ ГОСТ 28147-89 В РЕЖИМЕ ПРОСТОЙ ЗАМЕНЫ

7.3.1. Структура криптографической системы шифрования данных ГОСТ 28147-89 в режиме простой замены

Для реализации алгоритма шифрования в режиме *простой замены* (аналогичный режим – "*Электронная кодовая книга*") используется только часть блоков общей криптографической системы (рис. 7.1).

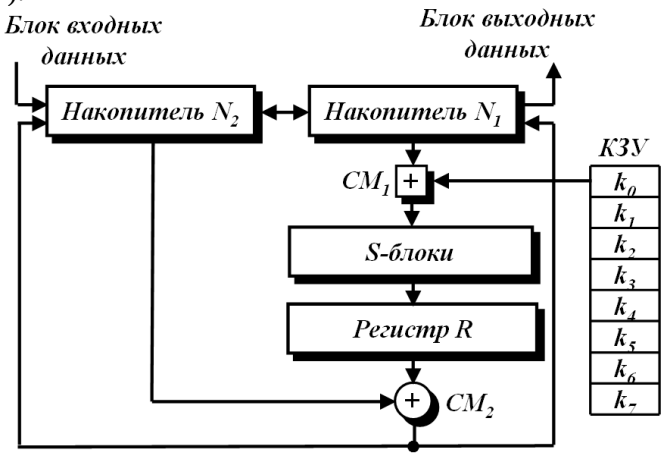


Рис. 7.1. Структура реализации режима простой замены стандарта *ГОСТ 28147-89*

Обозначения на схеме:

- N_1 и N_2 - *32*-разрядные накопители (регистры сдвига);
- CM_1 - *32*-разрядный сумматор по модулю 2^{32} ($\boxplus$);
- CM_2 - *32*-разрядный сумматор по модулю 2 ($\oplus$);
- *R* – *32*-разрядный регистр циклического сдвига данных на *11* разрядов влево (в сторону старших разрядов);
- *КЗУ* – ключевое запоминающее устройство на *256* бит, состоящее из восьми *32*-разрядных накопителей k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7;
- *S*-блок замены, состоящий из восьми узлов замены: S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

7.3.2. Зашифрование данных ГОСТ 28147-89 в режиме простой замены

Открытые данные, подлежащие зашифрованию, разбивают на *64*-разрядные блоки. Процедура зашифрования *64*-разрядного блока T_O в режиме простой замены включает *32* цикла (раунда). В *ключевое запоминающее устройство* (*КЗУ*) вводят *256* бит ключа шифра *K* в виде восьми *32*-разрядных подключей (чисел) k_i:

$$K = (k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7).$$

Последовательность битов блока

$$T_O = (a_1(0), a_2(0), \ldots, a_{32}(0), b_1(0), b_2(0), \ldots, b_{32}(0))$$

разбивают на две последовательности по *32* бита (взятые в обратном порядке):

$$b(0) = b_{32}(0), b_{31}(0), \ldots, b_1(0) \text{ и } a(0) = a_{32}(0), a_{33}(0), \ldots, a_1(0),$$

где *b(0)* – левые или старшие, а *a(0)* – правые или младшие биты.

При обозначении последовательностей *b(0)* и *a(0)* индекс в нижней части определяет номер бита.

Эти последовательности вводят в накопители N_1 и N_2 перед началом первого цикла зашифрования. В результате начальное заполнение накопителя N_1

$$N_1 = a(0) = (a_{32}(0), a_{31}(0), \ldots, a_1(0)),$$

начальное заполнение накопителя N_2

$$N_2 = b(0) = (b_{32}(0), b_{31}(0), \ldots, b_1(0)).$$

Первый цикл процедуры зашифрования *64*-разрядного блока открытых данных можно описать уравнениями:

$$\begin{cases} a(1) = f(a(0) \boxplus k_0) \oplus b(0); \\ b(1) = a(0), \end{cases}$$

где *a(1)* – заполнение N_1, после первого цикла зашифрования; *b(1)* – заполнение N_2 после первого цикла зашифрования; *f(●)* – функция зашифрования.

Аргументом функции $f(\bullet)$ является сумма по модулю 2^{32} числа $a(0)$ (начального заполнения накопителя N_1) и числа k_0 – подключа, считываемого из накопителя *КЗУ*. Каждое из этих чисел равно *32* битам.

Например, $a(0) = 1010\ 0100\ 0100\ 1101\ 1100\ 1011\ 0001\ 0101_2$, а $k_0 = 1001\ 0100\ 1000\ 1001\ 1000\ 0101\ 0010\ 1001_2$.

После выполнения операции суммирования по модулю 2^{32} результат на выходе сумматора CM_1 будет равен

$$a(0) \boxplus k_0 = 0011\ 1000\ 1101\ 0111\ 0101\ 0000\ 0011\ 1110_2.$$

Функция $f(\bullet)$ включает две операции над полученной *32*-разрядной суммой $a(0) \boxplus k_0$.

Первая операция называется подстановкой (заменой) и выполняется *S*-блоком подстановки. *S*-блок подстановки, как было выше сказано, состоит из восьми узлов замены с памятью *64* бита каждый. Поступающий из CM_1, на *S*-блок подстановки *32*-разрядный вектор разбивают на восемь последовательно идущих *4*-разрядных векторов, каждый из которых преобразуется в четырехразрядный вектор соответствующим узлом замены.

Каждый узел замены можно представить в виде таблицы-перестановки шестнадцати четырехразрядных двоичных чисел в диапазоне: $0000, 0001, \ldots, 1111_2$. Входной вектор указывает адрес строки в таблице, а число в этой строке является выходным вектором. Затем четырехразрядные выходные векторы последовательно объединяют в *32*-разрядный вектор. Узлы замены (таблицы-перестановки) представляют собой ключевые элементы, которые являются общими для сети *ЭВМ* и редко изменяются. Эти узлы замены должны сохраняться в секрете. Узлы замены определены документом *RFC 4357*. В табл. 7.1 приведен пример узла замены с помощью *S*-блоков. Этот узел замены определен *ГОСТ Р 34.11-94* для целей тестирования.

Например, если после выполнения операции суммирования по модулю 2^{32} результат на выходе сумматора CM_1 равен

$$a(0) \boxplus k_0 = 0000\ 0001\ 0010\ 0011\ 0100\ 0101\ 0110\ 0111_2,$$

то в таком случае, с учетом табл. 7.1, будет сформировано значение

$$S(a(0) \boxplus k_0) = 0001\ 1011\ 1010\ 0001\ 0000\ 0011\ 1111\ 1110_2.$$

Таблица 7.1

Замена данных с помощью *S*-блоков

		Значения четырехразрядных слов															
		00	*01*	*02*	*03*	*04*	*05*	*06*	*07*	*08*	*09*	*10*	*11*	*12*	*13*	*14*	*15*
Номера S-блоков замены	*00*	*04*	*10*	*09*	*02*	*13*	*08*	*00*	*14*	*06*	*11*	*01*	*12*	*07*	*15*	*05*	*03*
	01	*14*	*11*	*04*	*12*	*06*	*13*	*15*	*10*	*02*	*03*	*08*	*01*	*00*	*07*	*05*	*09*
	02	*05*	*08*	*01*	*13*	*10*	*03*	*04*	*02*	*14*	*15*	*12*	*07*	*06*	*00*	*09*	*11*
	03	*07*	*13*	*10*	*01*	*00*	*08*	*09*	*15*	*14*	*04*	*06*	*12*	*11*	*02*	*05*	*03*
	04	*06*	*12*	*07*	*01*	*05*	*15*	*13*	*08*	*04*	*10*	*09*	*14*	*00*	*03*	*11*	*02*
	05	*04*	*11*	*10*	*00*	*07*	*02*	*01*	*13*	*03*	*06*	*08*	*05*	*09*	*12*	*15*	*14*
	06	*13*	*11*	*04*	*01*	*03*	*15*	*05*	*09*	*00*	*10*	*14*	*07*	*06*	*08*	*02*	*12*
	07	*01*	*15*	*13*	*00*	*05*	*07*	*10*	*04*	*09*	*02*	*03*	*14*	*06*	*11*	*08*	*12*

Вторая операция – циклический сдвиг влево (на *11* разрядов) *32*-разрядного вектора, полученного с выхода *S*-блока замены. Циклический сдвиг выполняется регистром сдвига *R*.

Далее результат работы функции зашифрования $f(\bullet)$ суммируют поразрядно по модулю *2* в сумматоре CM_2 с *32*-разрядным начальным заполнением $b(0)$ накопителя N_2. Затем значение N_1 переписывают в накопитель N_2 (значение $b(1) = a(0)$), а полученный на выходе CM_2 результат (значение $a(1) = f(a(0) \boxplus k_0) \oplus b(0)$ записывают в накопитель N_1. После этого первый цикл завершен.

Последующие циклы осуществляются аналогично, при этом во втором цикле из *КЗУ* считывают подключ k_1, в третьем цикле – k_2, и т.д., в восьмом цикле – k_7. В циклах с *9*-го по *16*-й, а также в циклах с *17*-го по *24*-й подключи из *КЗУ* считываются в том же порядке:

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7.$$

В последних восьми циклах (с *25*-го по *32*-й) порядок считывания подключей из *КЗУ* обратный: $k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0$.

Таким образом, при шифровании в *32* циклах осуществляется следующий порядок выборки из *КЗУ* подключей:

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7,$$

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0.$$

После завершения *32*-го цикла результат из сумматора CM_2 вводится в накопитель N_2, а в накопителе N_1, сохраняется прежнее за-

полнение. Полученные после *32*-го цикла зашифрования заполнения накопителей N_2 и N_1 являются блоком зашифрованных данных $T_{Ш}$, соответствующим блоку открытых данных T_O.

Уравнения зашифрования в режиме простой замены имеют вид:

$$\begin{cases} a(j) = f(a(j-1) \boxplus k_{(j-1) \bmod 8}) \oplus b(j-1), \ j = 1, 2, \ldots, 24; \\ b(j) = a(j-1), \ j = 1, 2, \ldots, 24, \end{cases} \tag{7.1}$$

$$\begin{cases} a(j) = f(a(j-1) \boxplus k_{32-j}) \oplus b(j-1), \ j = 25, 26, \ldots, 31; \\ b(j) = a(j-1), \ j = 25, 26, \ldots, 31, \end{cases} \tag{7.2}$$

$$\begin{cases} b(j) = f(a(j-1) \boxplus k_{32-j}) \oplus b(j-1), \ j = 32; \\ a(j) = a(j-1), \ j = 32, \end{cases} \tag{7.3}$$

где $a(j) = (a_{32}(j), a_{31}(j), \ldots, a_1(j))$ – заполнение N_1 после j-го цикла зашифрования; $b(j) = (b_{32}(j), b_{31}(j), \ldots, b_1(j))$ – заполнение N_2 после j-го цикла зашифрования, $j = 1,2, \ldots, 32$.

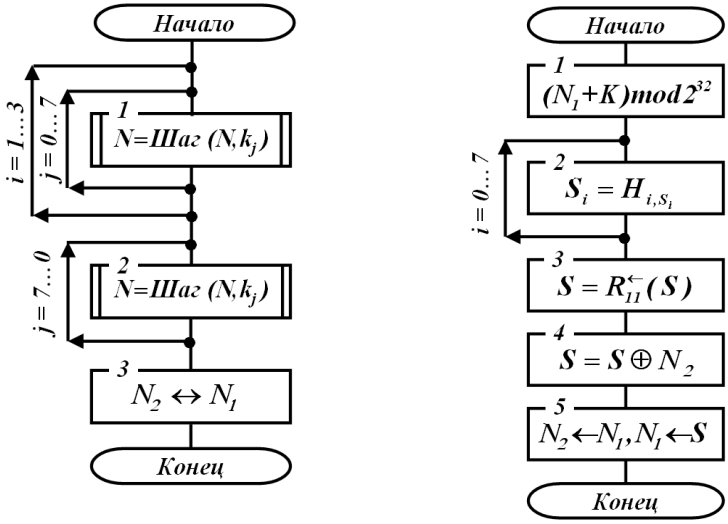
Блок зашифрованных данных $T_{Ш}$ (*64* разряда) выводится из накопителей N_2 и N_1 в следующем порядке: из разрядов *1...32* накопителя N_1, затем из разрядов *1...32* накопителя N_2, т.е. начиная с младших разрядов:

$$T_{Ш} = (a_1(32), a_2(32), \ldots, a_{32}(32), b_1(32), b_2(32), \ldots, b_{32}(32)).$$

Остальные блоки открытых данных зашифровываются в режиме простой замены аналогично.

Структурная схема алгоритма зашифрования данных в режиме простой замены приведена на рис. 7.2. На рис. 7.2 операция циклического сдвига на одиннадцать разрядов влево обозначена $R_{11}^{\leftarrow}$.

Структурная схема основного шага криптографического преобразования стандарта ($N = Шаг(N,k_j)$) приведена на рис. 7.3.



а) б)

Рис. 7.2. Структурная схема алгоритма зашифрования данных в режиме простой замены: а) *32*-х циклов; б) одного цикла

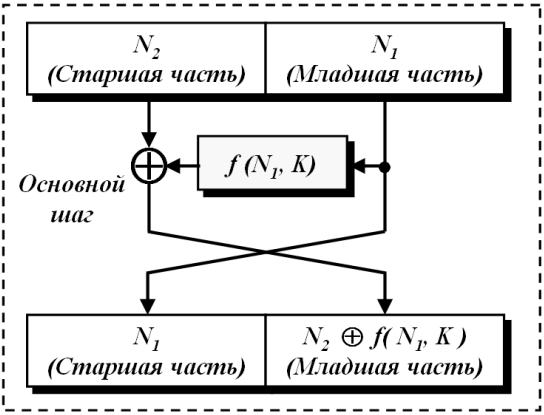


Рис. 7.3. Структурная схема основного шага криптографического преобразования стандарта

7.3.3. Расшифрование данных ГОСТ 28147-89 в режиме простой замены

Криптографическая схема, реализующая алгоритм расшифрования данных в режиме простой замены, имеет тот же вид, что и при зашифровании (см. рис. 7.1).

Расшифрование данных в режиме простой замены осуществляется по тому же алгоритму, что и зашифрование, с тем изменением, что заполнения накопителей считываются из *КЗУ* в циклах расшифрования в следующем порядке (в порядке – обратным зашифрованию)

$$k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0,$$

$$k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0, k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0 .$$

В соответствие с этим уравнения (7.1)…(7.3) для зашифрования данных в режиме простой замены могут быть представлены для расшифрования в таком виде:

$$\begin{cases} a(32-j) = f(a(33-j) \boxplus k_{(j-1)}) \oplus b(33-j), \ j = 1, 2, \ldots, 8; \\ b(32-j) = a(33-j), \ j = 1, 2, \ldots, 8, \end{cases} \tag{7.4}$$

$$\begin{cases} a(32-j) = f(a(33-j) \boxplus k_{(32-j) \bmod 8}) \oplus b(33-j), \ j = 9, 10, \ldots, 31; \\ b(32-j) = a(33-j), \ j = 9, 10, \ldots, 31, \end{cases} \tag{7.5}$$

$$\begin{cases} b(32-j) = f(a(33-j) \boxplus k_{32-j}) \oplus b(33-j), \ j = 32; \\ a(32-j) = a(33-j), \ j = 32. \end{cases} \tag{7.6}$$

Полученные после *32* циклов работы заполнения накопителей N_1 и N_2 образуют блок расшифрованных (открытых) данных

$$T_O = (a_1(0), a_2(0), \ldots, a_{32}(0), b_1(0), b_2(0), \ldots, b_{32}(0)).$$

При этом состояние накопителя N_1:

$$N_1 = a(0) = (a_{32}(0), a_{31}(0), \ldots, a_1(0)),$$

а состояние накопителя N_2

$$N_2 = b(0) = (b_{32}(0), b_{31}(0), \ldots, b_1(0)).$$

Аналогично расшифровываются остальные блоки зашифрованных данных.

Если алгоритм зашифрования данных в режиме простой замены *64*-битового блока T_O обозначить через *A*, то

$$A(T_O) = A(a(0),b(0)) = (a(32), b(32)) = T_{Ш}, \qquad (7.7)$$

а

$$A^{-1}(T_{Ш}) = A^{-1}(a(32),b(32)) = (a(0), b(0)) = T_O. \qquad (7.8)$$

Следует иметь в виду, что режим простой замены допустимо использовать для шифрования данных только в ограниченных случаях – при выработке ключа шифра и шифровании его с обеспечением имитозащиты для передачи по каналам связи или для хранения в памяти *ЭВМ*.

В стандарте *ГОСТ 28147-89* зашифрование и расшифрование данных обозначают "*Цикл 32-З*" ($Ц_{32\text{-}З}$) и "*Цикл 32-Р*" ($Ц_{32\text{-}Р}$) соответственно.

7.4. ШИФРОВАНИЕ ДАННЫХ ГОСТ 28147-89 В РЕЖИМЕ ГАММИРОВАНИЯ

7.4.1. Структура криптографической системы шифрования данных ГОСТ 28147-89 в режиме гаммирования

Для реализации алгоритма шифрования данных в режиме гаммирования используются полностью все блоки общей криптографической системы (рис. 7.4).

Обозначения на схеме:

- N_1–N_6 – *32*-разрядные накопители;
- $СМ_1$ и $СМ_3$ – *32*-разрядные сумматоры по модулю 2^{32} ($\boxplus$);
- $СМ_2$ и $СМ_5$ – *32*-разрядные сумматоры по модулю *2* ($\oplus$);
- $СМ_4$ – *32*-разрядный сумматор по модулю $2^{32}–1$ ($\boxplus$);
- *R* – *32*-разрядный регистр циклического сдвига данных на *11* разрядов влево (в сторону старших разрядов);
- *КЗУ* – ключевое запоминающее устройство на *256* бит, состоящее из восьми *32*-разрядных накопителей k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7;
- *S*-блок замены, состоящий из восьми узлов замены: S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

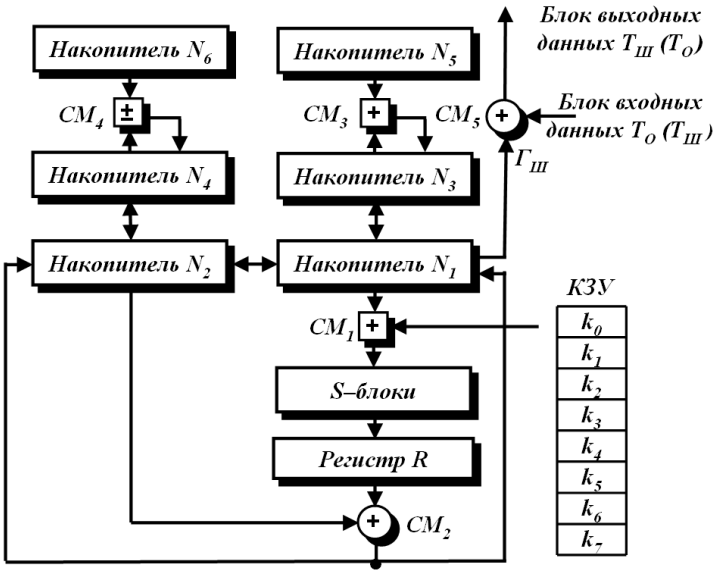


Рис. 7.4. Структура реализации режима гаммирования стандарта

Структурная схема алгоритма зашифрования данных приведена на рис. 7.5.

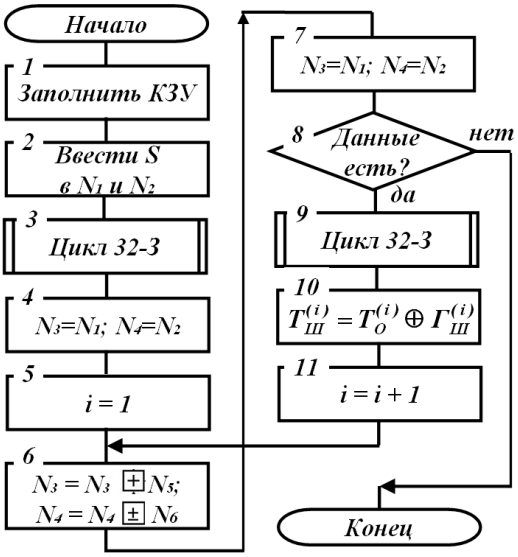


Рис. 7.5. Алгоритм зашифрования данных в режиме гаммирования

7.4.2. Зашифрование данных ГОСТ 28147-89 в режиме гаммирования

Открытые данные разбивают на *64*-разрядные блоки

$$T_O = (T_O^{(1)}, T_O^{(2)}, ..., T_O^{(q)}),$$

где $T_O^{(i)}$ – *i*-й *64*-разрядный блок открытых данных (*i = 1, 2, ..., q*, а *q* определяется объемом шифруемых данных).

Эти блоки поочередно зашифровываются в режиме гаммирования путем поразрядного сложения по модулю *2* в сумматоре CM_5 с гаммой шифра $Г_Ш$, которая вырабатывается блоками по *64* бита, т.е.

$$Г_Ш = (Г_Ш^{(1)}, Г_Ш^{(2)}, ..., Г_Ш^{(q)}),$$

где $Г_Ш^{(i)}$ – *i*-й *64*-разрядный блок гаммы (*i = 1, 2, ..., q*, а *q* определяется объемом шифруемых данных).

Число двоичных разрядов в блоке $T_O^{(q)}$ может быть меньше *64*, при этом неиспользованная для зашифрования часть гаммы шифра из блока $Г_Ш^{(q)}$ отбрасывается.

Уравнение шифрования данных в режиме гаммирования имеет вид

$$T_Ш^{(i)} = T_O^{(i)} \oplus Г_Ш^{(i)},$$

где $Г_Ш^{(i)} = A(Y_{i-1} \boxplus C_2, Z_{i-1} \boxplus C_1)$, *i = 1,2,...,q*; $T_Ш^{(i)}$ – *i*-й *64*-разрядный блок зашифрованных данных; *A(●)* – функция зашифрования в режиме простой замены; C_1 и C_2 – *32*-разрядные двоичные константы; Y_i и Z_i – *32*-разрядные двоичные последовательности, которые определяются итерационно по мере формирования гаммы $Г_Ш^{(i)}$.

Вначале формируются исходные значения Y_i и Z_i следующим образом:

$$(Y_0, Z_0) = A(\tilde{S}),$$

где $\tilde{S}$ – *64*-разрядная двоичная последовательность (синхропосылка).

Затем итерационно

$$(Y_i, Z_i) = (Y_{i-1} \boxplus C_2, Z_{i-1} \boxplus C_1),\ i = 1, 2, ..., q.$$

Рассмотрим реализацию процедуры зашифрования в режиме гаммирования. В накопители N_6 и N_5 заранее записаны *32*-разрядные двоичные константы C_1 и C_2, имеющие следующие значения (в шестнадцатеричной и двоичной форме):

$C_1 = 01010101_{16} = 0000\ 0001\ 0000\ 0001\ 0000\ 0001\ 0000\ 0001_2$;

$C_2 = 01010104_{16} = 0000\ 0001\ 0000\ 0001\ 0000\ 0001\ 0000\ 0100_2$.

В *КЗУ* вводится *256* бит ключа; в накопители N_1 и N_2 – *64*-разрядная двоичная последовательность (синхропосылка)

$$\tilde{S} = (S_1, S_2, \ldots, S_{64}),$$

где S_i – *i*-й двоичный разряд синхропосылки.

Синхропосылка $\tilde{S}$ является исходным заполнением накопителей N_1 и N_2 для последовательной выработки *q* блоков гаммы шифра. Исходное заполнение накопителя N_1: $N_1 = (S_{32}, S_{31}, \ldots, S_1)$, а исходное заполнение накопителя N_2: $N_2 = (S_{64}, S_{63}, \ldots, S_{33})$.

Заполнение накопителя N_4 суммируют по модулю 2^{32}-*1* в сумматоре CM_4 с *32*-разрядной константой C_1 из накопителя N_6. Результат записывается в N_4. Заполнение накопителя N_3 суммируется по модулю 2^{32}-*1* в сумматоре CM_3 с *32*-х разрядной константой C_2 из накопителя N_5. Сложение по модулю 2^{32}–*1*, по сути, является аддитивной инверсией результата сложения. Результат записывается в N_3. Заполнение N_3 переписывают в N_1, а заполнение N_4 - в N_2, при этом заполнения N_3 и N_4 сохраняются. Заполнения накопителей N_1 и N_2 шифруются в режиме простой замены.

Полученное в результате шифрования заполнения накопителей N_1 и N_2 образуют первый *64*-разрядный блок гаммы шифра

$$\Gamma_{Ш}^{(1)} = (\gamma_1^{(1)}, \gamma_2^{(1)}, \gamma_3^{(1)}, \ldots, \gamma_{63}^{(1)}, \gamma_{64}^{(1)}),$$

где γ_i^1, *i = 1, 2, ..., 64* – двоичный разряд последовательности гаммы.

Этот блок гаммы ($\Gamma_{Ш}^{(1)}$) суммируют поразрядно по модулю *2* в сумматоре CM_5 с первым *64*-разрядным блоком открытых данных

$$T_O^{(1)} = (t_1^{(1)}, t_2^{(1)}, t_3^{(1)}, \ldots, t_{63}^{(1)}, t_{64}^{(1)}),$$

где t_i^1, *i = 1, 2, ..., 64* – двоичный разряд первого блока открытых данных.

В результате суммирования по модулю *2* значений $T_O^{(1)}$ и $\Gamma_{Ш}^{(1)}$ получают первый *64*-разрядный блок зашифрованных данных:

$$T_{Ш}^{(1)} = T_O^{(1)} \oplus \Gamma_{Ш}^{(1)} = (\tau_1^{(1)}, \tau_2^{(1)}, \tau_3^{(1)}, ..., \tau_{63}^{(1)}, \tau_{64}^{(1)})$$

где $\tau_i^{(1)} = t_i^{(1)} \oplus \gamma_i^{(1)}$, *i = 1, 2, ..., 64* – двоичный разряд первого блока зашифрованных данных.

Для получения следующего *64*-разрядного блока гаммы шифра $\Gamma_{Ш}^{(2)}$ заполнение N_4 суммируют по модулю 2^{32}-*1* в сумматоре CM_4 с *32*-разрядной константой C_1 из накопителя N_6. Результат записывается в N_4. Заполнение накопителя N_3 суммируется по модулю 2^{32} в сумматоре CM_3 с *32*-разрядной константой C_2 из накопителя N_5. Результат записывается в N_3. Новое заполнение N_3 переписывают в N_1, а новое заполнение N_4 – в N_2, при этом заполнения N_3 и N_4 сохраняются. Заполнения накопителей N_1 и N_2 зашифровуються в режиме простой замены.

Полученное в результате зашифрования заполнения накопителей N_1 и N_2 образуют второй *64*-разрядный блок гаммы шифра $\Gamma_{Ш}^{(2)}$, который суммируется поразрядно по модулю *2* в сумматоре CM_5 со вторым блоком открытых данных $T_O^{(2)}$:

$$T_{Ш}^{(2)} = T_O^{(2)} \oplus \Gamma_{Ш}^{(2)} = (\tau_1^{(2)}, \tau_2^{(2)}, \tau_3^{(2)}, ..., \tau_{63}^{(2)}, \tau_{64}^{(2)}).$$

Аналогично вырабатываются блоки гаммы шифра

$$\Gamma_{Ш}^{(3)}, \Gamma_{Ш}^{(4)}, ..., \Gamma_{Ш}^{(q)}$$

и зашифровуються блоки открытых данных $T_O^{(3)}, T_O^{(4)}, ..., T_O^{(q)}$.

В канал связи или память *ЭВМ* передаются синхропосылка $\tilde{S}$ и блоки зашифрованных данных $T_{Ш}^{(1)}, T_{Ш}^{(2)}, ..., T_{Ш}^{(q)}$.

7.4.3. Расшифрование данных ГОСТ 28147-89 в режиме гаммирования

При расшифровании данных криптографическая схема имеет тот же вид, что и при зашифровании (см. рис. 7.4), а структурная схема алгоритма расшифрования данных приведена на рис. 7.6.

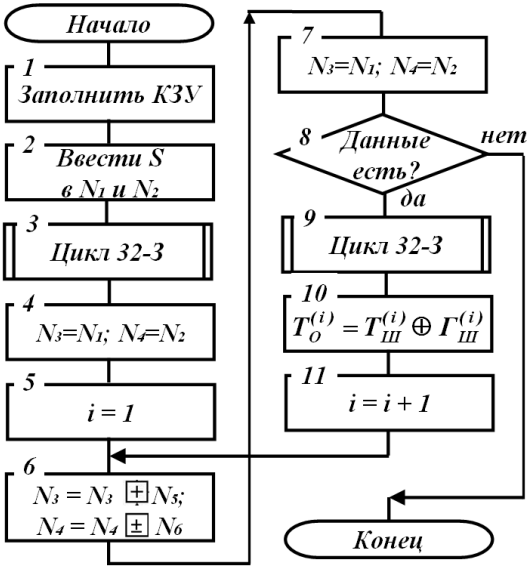


Рис. 7.6. Алгоритм расшифрования данных в режиме гаммирования стандарта

Уравнение расшифрования:

$$T_O^{(i)} = T_{Ш}^{(i)} \oplus Г_{Ш}^{(i)} = T_{Ш}^{(i)} \oplus A(Y_{i-1} \boxplus C_2, Z_{i-1} \boxplus C_1),\ i = 1, 2, \ldots, q.$$

Следует отметить, что расшифрование данных возможно только при наличии синхропосылки, которая не является секретным элементом шифра и может храниться в памяти *ЭВМ* или передаваться по каналам связи вместе с зашифрованными данными.

Рассмотрим реализацию процедуры расшифрования. В *КЗУ* вводят *256* бит ключа, с помощью которого осуществляется шифрование данных $T_O^{(1)}, T_O^{(2)}, \ldots, T_O^{(q)}$. В накопители N_1 и N_2 вводится син-

хропосылка, и осуществляется процесс выработки q блоков гаммы шифра

$$\Gamma_{Ш}^{(1)},\Gamma_{Ш}^{(2)},...,\Gamma_{Ш}^{(q)}.$$

Блоки зашифрованных данных $T_{Ш}^{(1)}$, $T_{Ш}^{(2)},...,T_{Ш}^{(q)}$ суммируются поразрядно по модулю *2* в сумматоре $СМ_5$ с блоками гаммы шифра $\Gamma_{Ш}^{(1)},\Gamma_{Ш}^{(2)},...,\Gamma_{Ш}^{(q)}$. В результате получаются блоки открытых данных $T_O=(T_O^{(1)}, T_O^{(2)},..., T_O^{(q)})$ при этом, $T_O^{(q)}$ может содержать меньше *64* разрядов.

7.5. ШИФРОВАНИЕ ДАННЫХ ГОСТ 28147-89 В РЕЖИМЕ ГАММИРОВАНИЯ С ОБРАТНОЙ СВЯЗЬЮ

7.5.1. Структура криптографической системы шифрования данных ГОСТ 28147-89 в режиме гаммирования с обратной связью

Для реализации алгоритма шифрования данных в режиме гаммирования с обратной связью используется тоже только часть блоков общей криптографической системы (рис. 7.7).

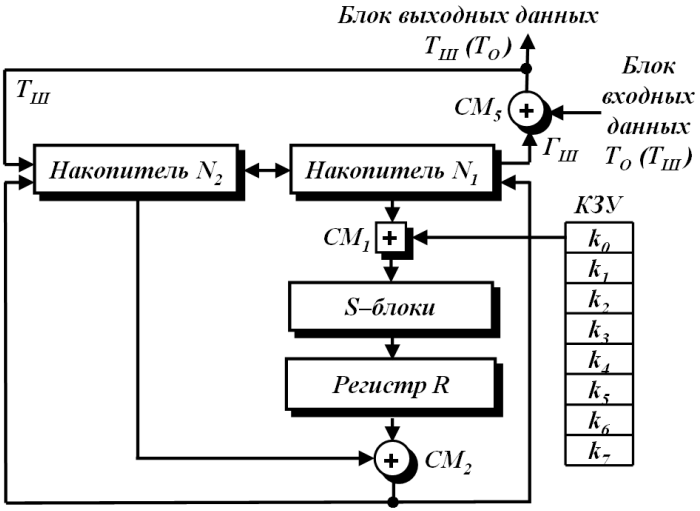


Рис. 7.7. Схема реализации режима гаммирования с обратной связью стандарта

Обозначения на схеме:

- N_1, N_2 – *32*-разрядные накопители;
- CM_1 – *32*-разрядный сумматор по модулю 2^{32} ($\boxplus$);
- CM_2 и CM_5 – *32*-разрядные сумматоры по модулю *2* ($\oplus$);
- *R* – *32*-разрядный регистр циклического сдвига данных на *11* разрядов влево (в сторону старших разрядов);
- *КЗУ* – ключевое запоминающее устройство на *256* бит, состоящее из восьми *32*-х разрядных накопителей k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7;
- *S*-блок замены, состоящий из восьми узлов замены: S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7.

7.5.2. Зашифрование данных ГОСТ 28147-89 в режиме гаммирования с обратной связью

Криптографическая схема, реализующая алгоритм зашифрования в режиме гаммирования с обратной связью, показана на рис. 7.7, а структурная схема алгоритма шифрования данных приведена на рис. 7.8.

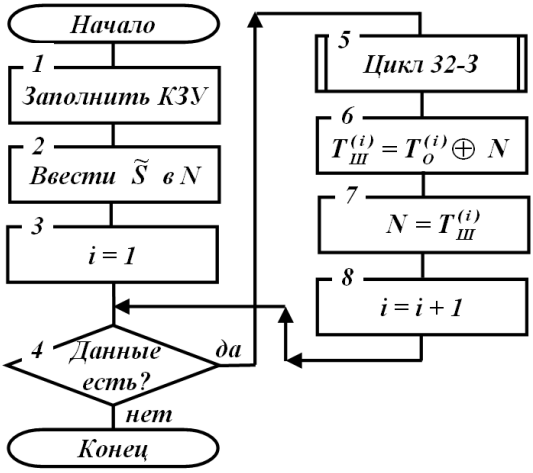


Рис. 7.8. Алгоритм зашифрования данных в режиме гаммирования с обратной связью стандарта

Открытые данные, разбитые на *64*-разрядные блоки $T_O = (T_O^{(1)}, T_O^{(2)}, ..., T_O^{(q)})$, зашифровываются в режиме гаммирования

с обратной связью путем поразрядного сложения по модулю *2* с гаммой шифра $\Gamma_{Ш}$, которая вырабатывается блоками по *64* бита

$$\Gamma_{Ш} = (\Gamma_{Ш}^{(1)}, \Gamma_{Ш}^{(2)}, ..., \Gamma_{Ш}^{(q)}).$$

Число двоичных разрядов в блоке $T_O^{(q)}$ может быть меньше *64*, при этом неиспользованная для шифрования часть гаммы шифра из блока $\Gamma_{Ш}^{(q)}$ отбрасывается.

Уравнения зашифрования в режиме гаммирования с обратной связью имеют вид:

$$T_{Ш}^{(1)} = \Gamma_{Ш}^{(1)} \oplus T_O^{(1)} = A(\tilde{S}) \oplus T_O^{(1)};$$

$$T_{Ш}^{(i)} = A(T_{Ш}^{(i-1)}) \oplus T_O^{(i)},\ i = 2, 3, 4, ..., q,$$

где $T_{Ш}^{(i)}$ – *i*-й *64*-разрядный блок шифрованных данных; *A(●)* – функция зашифрования в режиме простой замены.

Аргументом функции *A(●)* на первом шаге итеративного алгоритма является *64*-разрядная синхропосылка $\tilde{S}$, а на всех последующих шагах – предыдущий блок зашифрованных данных $T_{Ш}^{(i-1)}$.

Процедура зашифрования данных в режиме гаммирования с обратной связью реализуется следующим образом. В *КЗУ* вводятся *256* бит ключа. В накопители N_1 и N_2 вводится синхропосылка $\tilde{S} = (S_1, S_2, ..., S_{64})$ состоящая из *64* бит.

Исходное заполнение накопителей N_1 и N_2 (синхропосылка) зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение накопителей N_1 и N_2 образует первый *64*-разрядный блок гаммы шифра

$$\Gamma_{Ш}^{(1)} = A(\tilde{S}),$$

который суммируется поразрядно по модулю *2* в сумматоре CM_5 с первым *64*-разрядным блоком открытых данных

$$T_O^{(1)} = (t_1^{(1)}, t_2^{(1)}, \quad t_3^{(1)}, ..., t_{63}^{(1)}, t_{64}^{(1)}).$$

В результате получают первый *64*-разрядный блок зашифрованных данных:

$$T_{Ш}^{(1)} = T_{O}^{(1)} \oplus \Gamma_{Ш}^{(1)} = (\tau_1^{(1)}, \tau_2^{(1)}, \tau_3^{(1)}, ..., \tau_{63}^{(1)}, \tau_{64}^{(1)}).$$

Блок зашифрованных данных $T_{Ш}^{(1)}$ одновременно является также исходным состоянием накопителей N_1 и N_2 для выработки второго блока гаммы шифра $\Gamma_{Ш}^{(2)}$, и поэтому по обратной связи $T_{Ш}^{(1)}$ записывается в указанные накопители N_1 и N_2.

Заполнение накопителя N_1:

$$N_1 = (\tau_{32}^{(1)}, \tau_{31}^{(1)}, \tau_{30}^{(1)}, ..., \tau_1^{(1)}, \tau_2^{(1)}),$$

а заполнение накопителя N_2:

$$N_2 = (\tau_{64}^{(1)}, \tau_{63}^{(1)}, \tau_{62}^{(1)}, ..., \tau_{34}^{(1)}, \tau_{33}^{(1)}).$$

Заполнение накопителей N_1 и N_2 зашифровывается в режиме простой замены. Полученное в результате зашифрования заполнение накопителей N_1 и N_2 образует второй *64*-х разрядный блок гаммы шифра $\Gamma_{Ш}^{(2)}$ который суммируется поразрядно по модулю *2* в сумматоре CM_5 со вторым блоком открытых данных $T_{O}^{(2)}$

$$T_{Ш}^{(2)} = T_{O}^{(2)} \oplus \Gamma_{Ш}^{(2)} = A(T_{Ш}^{(1)}) \oplus T_{O}^{(2)}).$$

Формирование последующих блоков гаммы шифра $\Gamma_{Ш}^{(i)}$ и зашифрование соответствующих блоков открытых данных $T_{O}^{(i)}$ производится аналогично.

Если длина последнего q-го блока открытых данных $T_{O}^{(q)}$ меньше *64*-разрядов, то из $\Gamma_{Ш}^{(i)}$ используется только соответствующее число разрядов гаммы шифра, остальные разряды отбрасываются.

В канал связи или память *ЭВМ* передаются синхропосылка $\tilde{S}$ и блоки зашифрованных данных

$$T_{Ш} = (T_{Ш}^{(1)}, T_{Ш}^{(2)}, T_{Ш}^{(3)}, ..., T_{Ш}^{(q-1)}, T_{Ш}^{(q)}).$$

7.5.3. Расшифрование данных ГОСТ 28147-89 в режиме гаммирования с обратной связью

При расшифровании данных в режиме гаммирования с обратной связью криптографическая схема *ГОСТ 28147-89* имеет тот же вид, что и при зашифровании (см. рис. 7.7). Структурная схема алгоритма расшифрования данных приведена на рис. 7.9.

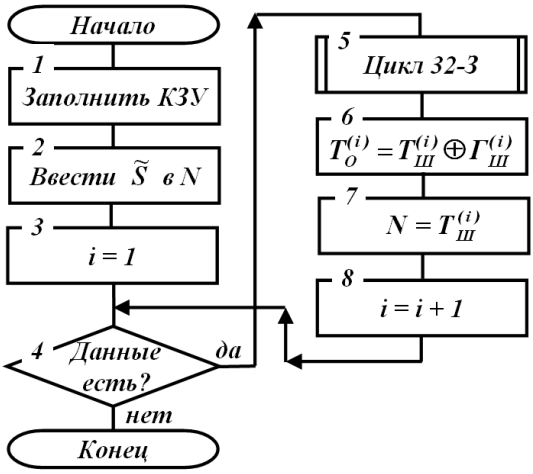


Рис. 7.9. Алгоритм расшифрования данных в режиме гаммирования с обратной связью стандарта

Уравнение расшифрования:

$$T_O^{(1)} = A(\tilde{S}) \oplus T_{Ш}^{(1)} = \Gamma_{Ш}^{(1)} \oplus T_{Ш}^{(1)};$$

$$T_O^{(i)} = A(T_{Ш}^{(i-1)}) \oplus T_{Ш}^{(i)} = \Gamma_{Ш}^{(i)} \oplus T_{Ш}^{(i)},\ i = 2, 3, \ldots, q.$$

Как следует из приведенных соотношений, что расшифрование данных возможно только при наличии синхропосылки $\tilde{S}$, которая не является секретным элементом шифра и может храниться в памяти *ЭВМ* или передаваться по каналам связи вместе с зашифрованными данными.

Рассмотрим реализацию процедуры расшифрования. В *КЗУ* вводят *256* бит ключа, с помощью которого осуществляется зашифрование данных $T_O^{(1)}, T_O^{(2)}, \ldots, T_O^{(q)}$. В накопители N_1 и N_2 вводится

синхропосылка $\tilde{S}$. Исходное заполнение накопителей N_1 и N_2 (синхропосылка $\tilde{S}$) шифруются в режиме простой замены.

Полученное в результате шифрования заполнение N_1 и N_2 образует первый блок гаммы шифра

$$Г_{Ш}^{(1)} = A(\tilde{S}),$$

который суммируется поразрядно по модулю *2* в сумматоре $СМ_5$ с блоком зашифрованных данных $T_{Ш}^{(1)}$. В результате получается первый блок открытых данных

$$T_{О}^{(1)} = Г_{Ш}^{(1)} \oplus T_{Ш}^{(1)}.$$

Блок зашифрованных данных $T_{Ш}^{(1)}$ является исходным заполнением накопителей N_1 и N_2 для выработки второго блока гаммы шифра $Г_{Ш}^{(2)}$

$$Г_{Ш}^{(2)} = A(T_{Ш}^{(1)}).$$

Полученное заполнение накопителей N_1 и N_2 зашифровуются в режиме простой замены. Образованный в результате зашифрования блок $Г_{Ш}^{(2)}$ суммируется поразрядно по модулю *2* в сумматоре $СМ_5$ со вторым блоком шифрованных данных $T_{Ш}^{(2)}$. В результате получают второй блок открытых данных $T_{О}^{(2)}$. Аналогично в N_1 и N_2 последовательно записывают блоки шифрованных данных

$$T_{Ш}^{(2)}, T_{Ш}^{(3)}, T_{Ш}^{(4)}, ..., T_{Ш}^{(q-1)}, T_{Ш}^{(q)},$$

из которых в режиме простой замены вырабатываются блоки гаммы шифра $Г_{Ш}^{(3)}, Г_{Ш}^{(4)}, ..., Г_{Ш}^{(q)}$.

Блоки гаммы шифра суммируются поразрядно по модулю *2* в сумматоре $СМ_5$ с блоками зашифрованных данных

$$T_{Ш}^{(3)}, T_{Ш}^{(4)}, ..., T_{Ш}^{(q-1)}, T_{Ш}^{(q)}.$$

В результате получают блоки открытых данных $T_{О}^{(3)}$, $T_{О}^{(4)}$,…, $T_{О}^{(q)}$, при этом последний блок открытых данных $T_{О}^{(q)}$ может содержать меньше *64* разрядов.

7.6. КРИПТОГРАФИЧЕСКИЕ ПРЕОБРАЗОВАНИЯ ДАННЫХ ГОСТ 28147-89 В РЕЖИМЕ ВЫРАБОТКИ ИМИТОВСТАВКИ

Имитовставка – это блок из L бит, который вырабатывают по определенному правилу из открытых данных с использованием ключа и затем добавляется к зашифрованным данным для обеспечения их *имитозащиты*.

Имитозащита – это защита системы шифрованной связи от навязывания ложных данных.

В стандарте *ГОСТ 28147-89* определяется процесс выработки имитовставки, который единообразен для любого из режимов шифрования данных. Имитовставка $И_L$ вырабатывается из блоков открытых данных либо перед зашифрованием всего сообщения, либо параллельно с зашифрованием по блокам. Первые блоки открытых данных, которые участвуют в выработке имитовставки, могут содержать служебную информацию (например, адресную часть, время, синхропосылку) и не зашифровываются.

Значение параметра L (число двоичных разрядов в имитовставке) определяется криптографическими требованиями с учетом того, что вероятность навязывания ложных помех равна $P_{ЛП} = 2^{-L}$.

Для реализации алгоритма криптографического преобразования в режиме выработки имитовставки используется тоже только часть блоков общей криптосистемы (рис. 7.10).

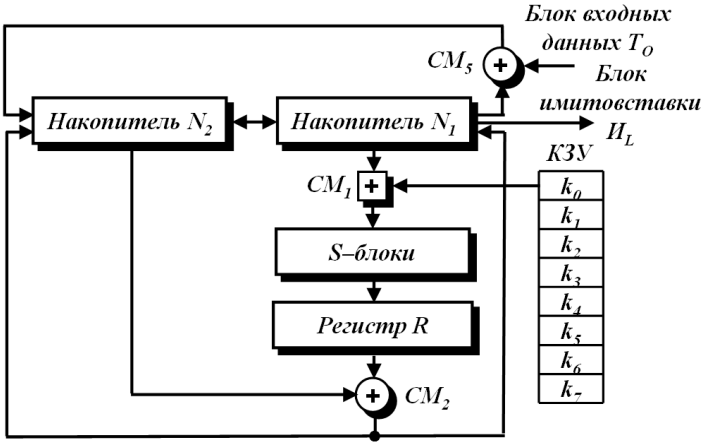


Рис. 7.10. Схема реализации режима выработки имитовставки стандарта

Этот режим не является в общепринятом смысле режимом шифрования. При работе в режиме выработки имитовставки создаётся некоторый дополнительный блок, зависящий от всех открытых и ключевых данных. Данный блок используется для проверки того, что в зашифрованные данные случайно или преднамеренно не были внесены искажения. Это особенно важно для зашифрования в режиме гаммирования, где злоумышленник может изменить конкретные биты, даже не зная ключа; однако и при работе в других режимах вероятные искажения нельзя обнаружить, если в передаваемых данных нет избыточной информации.

Имитовставка вырабатывается только в том случае, если блоков открытых данных по 64 бита не меньше двух.

Структурная схема алгоритма криптографического преобразования данных *ГОСТ 28147-89* в режиме выработки имитовставки приведена на рис. 7.11.

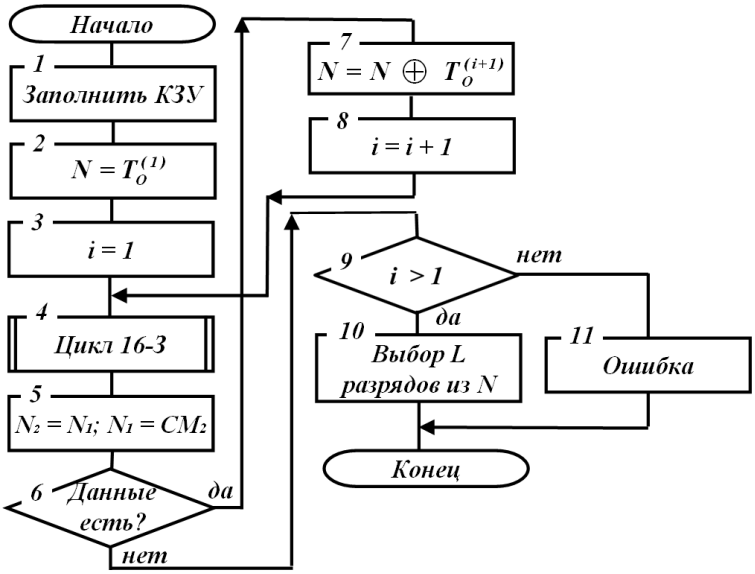


Рис. 7.11. Структурная схема алгоритма криптографических преобразований данных *ГОСТ 28147-89* в режиме выработки имитовставки

Для выработки имитовставки открытые данные представляют в виде последовательности *64*-разрядных блоков $T_O^{(i)}$, $i = 1,2,...,q$.

В ключевое запоминающее устройство (*КЗУ*) вводят *256* бит ключа *K*.

Первый блок открытых данных $T_O^{(1)}$ записывается в регистры N_1 и N_2, после чего подвергается преобразованию $\tilde{A}(\bullet)$, соответствующему первым *16* циклам алгоритма зашифрования в режиме простой замены.

Полученное после *16* циклов *64*-разрядное число $\tilde{A}(T_O^{(1)})$ суммируют по модулю *2* со вторым блоком открытых данных $T_O^{(2)}$. Результат суммирования $(\tilde{A}(T_O^{(1)}) \oplus T_O^{(2)})$ снова подвергают преобразованию $\tilde{A}(\bullet)$.

Полученное *64*-разрядное число $\tilde{A}(\tilde{A}(T_O^{(1)}) \oplus T_O^{(2)}))$ суммируют по модулю 2 с третьим блоком $T_O^{(3)}$ и снова подвергают преобразованию $\tilde{A}(\bullet)$, получая *64*-разрядное число

$$\tilde{A}(\tilde{A}(\tilde{A}(T_O^{(1)}) \oplus T_O^{(2)}) \oplus T_O^{(3)}),$$

и т.д.

Последний блок $T_O^{(q)}$ (при необходимости дополненный нулями до полного *64*-разрядного блока) суммируют по модулю *2* с результатом вычислений на шаге *q–1*, после чего зашифровывается в режиме простой замены, используя преобразование $\tilde{A}(\bullet)$.

Из полученного *64*-разрядного числа выбирают непрерывный отрезок $И_L$ (имитовставку) длиной *L* старших (левых) бит (не менее одного и не более *64* бит)

$$И_L = (a_{32-L+1}^{(q)}(16), a_{32-L+2}^{(q)}(16), \ldots, a_{32}^{(q)}(16)),$$

где $a_i^{(q)}(16)$ – *i*-й бит *64*- разрядного числа, полученного после *16*-го цикла последнего преобразования $\tilde{A}(\bullet)$, $32-L+1 \le i \le 32$.

Имитовставка $И_P$ передается по каналу связи или в память *ЭВМ* в конце зашифрованных данных, т.е.

$$T_Ш^{(1)}, T_Ш^{(2)}, T_Ш^{(3)}, \ldots, T_Ш^{(q-1)}, T_Ш^{(q)}, И_L.$$

Поступившие к получателю зашифрованные данные

$$T_{Ш}^{(1)},T_{Ш}^{(2)},T_{Ш}^{(3)},...,T_{Ш}^{(q-1)},T_{Ш}^{(q)}$$

расшифровываются, и из полученных блоков открытых данных $T_{O}^{(1)},T_{O}^{(2)},T_{O}^{(3)},...,T_{O}^{(q-1)},T_{O}^{(q)}$ аналогичным образом вырабатывается имитовставка $И^*_L$. Эта имитовставка $И^*_L$ сравнивается с имитовставкой $И_L$, полученной вместе с зашифрованными данными из канала связи или из памяти *ЭВМ*. В случае несовпадения имитовставок полученные при расшифровании блоки открытых данных $T_{O}^{(1)},T_{O}^{(2)},T_{O}^{(3)},...,T_{O}^{(q-1)},T_{O}^{(q)}$ считают ложными.

Следует отметить, что выработка имитовставки может проводиться параллельно шифрованию с использованием одного из описанных выше режимов работы.

7.7. БЕЗОПАСНОСТЬ ГОСТ 28147-89

7.7.1. Криптографическая стойкость ГОСТ 28147-89

При выборе криптографического алгоритма для использования в конкретной разработке одним из определяющих факторов является его *стойкость*, то есть устойчивость к попыткам противника его раскрыть. Вопрос о стойкости шифра при ближайшем рассмотрении сводится к двум взаимосвязанным вопросам:

- можно ли вообще раскрыть данный шифр;
- если да, то насколько это трудно сделать практически.

Шифры, которые вообще невозможно раскрыть, называются *абсолютно* или *теоретически стойкими*. Существование подобных шифров доказывается теоремой *Шеннона* [45], однако ценой этой стойкости является необходимость использования для зашифрования каждого сообщения ключа, не меньшего по размеру самого сообщения. Во всех случаях за исключением ряда особых эта цена чрезмерна, поэтому на практике в основном используются шифры, не обладающие абсолютной стойкостью. Таким образом, наиболее употребительные схемы шифрования могут быть раскрыты за конечное время или, что точнее, за конечное число шагов, каждый из которых является некоторой операцией над числами. Для них наиважнейшее значение имеет понятие практической стойкости, выражающее практическую трудность их раскрытия. Количественной мерой этой трудности может служить число элементарных арифме-

тических и логических операций, которые необходимо выполнить, чтобы раскрыть шифр, то есть, чтобы для заданного зашифрованного сообщения с вероятностью, не меньшей заданной величины, определить соответствующее открытое сообщение. При этом в дополнении к дешифруемому массиву данных криптографический аналитик может располагать блоками открытых данных и соответствующих им зашифрованных данных или даже возможностью получить для любых выбранных им открытых данных соответствующие зашифрованные данные. В зависимости от перечисленных и многих других неуказанных условий различают отдельные виды криптографического анализа.

Все современные криптографические системы построены по принципу *Керкгоффса*, то есть секретность зашифрованных сообщений определяется секретностью ключа. Это значит, что даже если сам алгоритм шифрования известен криптографическому аналитику, тот, тем не менее, не в состоянии расшифровать сообщение, если не располагает соответствующим ключом. Шифр считается хорошо спроектированным, если нет способа вскрыть его более эффективным способом, чем полным перебором по всему ключевому пространству, т.е. по всем возможным значениям ключа. *ГОСТ 28147-89*, вероятно, соответствует этому принципу – за годы интенсивных исследований не было предложено ни одного результативного существенного способа его криптографического анализа. В плане стойкости он на много порядков превосходит прежний американский стандарт шифрования *DES*.

В *ГОСТ 28147-89* используется *256*-битовый ключ и объем ключевого пространства составляет 2^{256}. Ни на одном из существующих в настоящее время или предполагаемых к реализации в недалеком будущем электронном устройстве нельзя подобрать ключ за время, меньшее многих сотен лет. Эта величина стала фактическим стандартом размера ключа для симметричных криптографических алгоритмов в наши дни, поэтому, новый стандарт шифрования *США* также его поддерживает. Прежний же американский стандарт, *DES* с его реальным размером ключа в *56* бит и объемом ключевого пространства всего 2^{56} уже не является достаточно стойким в свете возможностей современных вычислительных средств. Это было продемонстрировано в конце 90-х годов XX столетия несколькими успешными попытками взлома *DES* переборным путем. Кроме того, *DES* оказался подвержен специальным способам криптографического анализа, таким как дифференциальный и линейный. В этой

связи *DES* может представлять скорее исследовательский или научный, чем практический интерес. В 1998 году его криптографическая слабость была признана официально: Национальный институт стандартов *США* рекомендовал использовать троекратное шифрование по *DES*. А в конце 2001 года был официально утвержден новый стандарт шифрования *США*, *AES*, построенный на иных принципах и свободный от недостатков своего предшественника [38].

7.7.2. Замечания по архитектуре ГОСТ 28147-89

Общеизвестно, что стандарт шифрования *ГОСТ 28147-89* является представителем целого семейства шифров, построенных на одних и тех же принципах. Самым известным его "*родственником*" является прежний американский стандарт шифрования *DES*. Все эти шифры, подобно *ГОСТ 28147-89*, содержат алгоритмы трех уровней. В основе всегда лежит некий "*основной шаг*", на базе которого сходным образом строятся "*базовые циклы*", и уже на их основе построены практические процедуры шифрования и выработки имитовставки. Таким образом, специфика каждого из шифров этого семейства заключена именно в его "*основном шаге*", точнее даже в его части.

Алгоритмы "*основных шагов криптопреобразования*" для шифров, подобных *ГОСТ 28147-89*, построены идентичным образом, и эта архитектура называется *сбалансированная сеть Фейстеля* (*Balanced Feistel Network*) по имени человека, впервые предложившего ее [4, 29]. Схема преобразования данных на одном цикле, или, как его принято называть, раунде, приведена на рис. 7.3.

На вход основного шага подается блок четного размера, старшая и младшая половины которого обрабатываются отдельно друг от друга. В ходе преобразования младшая половина блока помещается на место старшей, а старшая, скомбинированная с помощью операции побитового *xor* с результатом вычисления некоторой функции на место младшей. Эта функция, принимающая в качестве аргумента младшую половину блока и элемент ключевой информации (X), является содержательной частью шифра и называется его *функцией шифрования*. По разным соображениям, оказалось, выгодно разделить шифруемый блок на две одинаковые по размеру части: $|N_1| = |N_2|$ – именно этот факт отражает слово "*сбалансированная*" в названии архитектуры. Впрочем, шифрующие несбалансированные сети также используются время от времени, хотя и не

так часто, как сбалансированные. Кроме того, соображения стойкости шифра требуют, чтобы размер ключевого элемента не был меньше размера половины блока: $|N_i| \le |X|$ в *ГОСТ 28147-89* все три размера равны *32* битам.

Если применить сказанное к схеме основного шага алгоритма *ГОСТ 28147-89*, станет очевидным, что блоки 1, 2, 3 алгоритма (см. рис. 7.2, б) определяют вычисление его функции шифрования, а блоки 4 и 5 задают формирование выходного блока основного шага исходя из содержимого входного блока и значения функции шифрования.

Ранее было проведено сравнение *DES* и *ГОСТ 28147-89* по стойкости, теперь сравним их по функциональному содержанию и удобству реализации. В циклах шифрования *ГОСТ 28147-89* основной шаг повторяется *32* раза, для *DES* эта величина равна *16*. Однако сама функция шифрования *ГОСТ 28147-89* существенно проще аналогичной функции *DES*, в которой присутствует множество нерегулярных битовых перестановок. Эти операции чрезвычайно неэффективно реализуются на современных неспециализированных процессорах. *ГОСТ 28147-89* не содержит подобных операций, поэтому он значительно удобней для программной реализации.

Ни одна из реализаций *DES* для платформы *Intel x86* не достигает даже половины производительности предложенной реализации *ГОСТ 28147-89* несмотря на вдвое более короткий цикл. Все сказанное выше свидетельствует о том, что разработчики *ГОСТ 28147-89* учли как положительные, так и отрицательные стороны *DES*, а также более реально оценили текущие и перспективные возможности криптографического анализа. Впрочем, брать *DES* за основу при сравнении быстродействия реализаций шифров уже не актуально. У нового стандарта шифрования *США* дела с эффективностью обстоят гораздо лучше: при таком же, как у *ГОСТ 28147-89*, размере ключа в *256* бит *AES* работает быстрее него примерно на *14%* – это если сравнивать по числу элементарных операций. Кроме того, *ГОСТ 28147-89* практически не удается распараллелить, а у *AES* возможностей в этом плане намного больше. На некоторых архитектурах это преимущество *AES* может быть меньше, на других – больше. Так, на процессоре *Intel Pentium* оно достигает *28%*. Подробности можно найти в [1, 7, 13, 38, 47].

7.7.3. Требования к качеству ключевой информации ГОСТ 28147-89 и источники ключей

Не все ключи шифра и таблицы замен обеспечивают максимальную стойкость шифра. Для каждого алгоритма шифрования существуют свои критерии оценки ключевой информации. Так, для алгоритма *DES* известно существование так называемых *слабых ключей*, при использовании которых связь между открытыми и зашифрованными данными не маскируется достаточным образом, и шифр сравнительно просто вскрывается.

Исчерпывающий ответ на вопрос о критериях качества ключей и таблиц *ГОСТ 28147-89* если и можно вообще где-либо получить, то только у разработчиков алгоритма. Соответствующие данные не были опубликованы в открытой печати. Однако согласно установленному порядку, для шифрования информации, имеющей гриф, должны быть использованы ключевые данные, полученные от уполномоченной организации. Косвенным образом это может свидетельствовать о наличии методик проверки ключевых данных на "*вшивость*". Если наличие слабых ключей в *ГОСТ 28147-89* – дискуссионный вопрос, то наличие слабых узлов замены не вызывает сомнения. Очевидно, что "*тривиальная*" таблица замен, по которой любое значение заменяется им же самим, является настолько слабой, что при ее использовании шифр взламывается элементарно, каков бы ни был ключ.

Как уже было отмечено выше, критерии оценки ключевой информации недоступны, однако на их счет все же можно высказать некоторые общие соображения.

Ключ

Ключ должен являться массивом статистически независимых битов, принимающих с равной вероятностью значения *0* и *1*. Нельзя полностью исключить при этом, что 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. По крайней мере, интенсивные исследования шифра до сих пор не выявили ни одного такого ключа ни для одной из известных (т.е. предложенных *ФАПСИ*) таблиц замен. Поэтому ключи, выработанные с помощью некоторо-

го датчика истинно случайных чисел, будут качественными с вероятностью, отличающейся от единицы на ничтожно малую величину. Если же ключи вырабатываются с помощью генератора псевдослучайных чисел, то используемый генератор должен обеспечивать указанные выше статистические характеристики, и, кроме того, обладать высокой криптографической стойкостью, – не меньшей, чем у самого *ГОСТ 28147-89*. Иными словами, задача определения отсутствующих членов вырабатываемой генератором последовательности элементов не должна быть проще, чем задача вскрытия шифра. Кроме того, для отбраковки ключей с плохими статистическими характеристиками могут быть использованы различные статистические критерии. На практике обычно хватает двух критериев: для проверки равновероятного распределения битов ключа между значениями *0* и *1* обычно используется *критерий Пирсона* (“*хи квадрат*”), а для проверки независимости битов ключа – *критерий серий*.

Наилучшим подходом для выработки ключей было бы использование аппаратных датчиков случайных чисел, однако это не всегда приемлемо по экономическим соображениям. При генерации небольшого по объему массива ключевой информации разумной альтернативой использованию такого датчика является и широко используется на практике *метод* “*электронной рулетки*”, когда очередная вырабатываемая порция случайных битов зависит от момента времени нажатия оператором некоторой клавиши на клавиатуре компьютера. В этой схеме источником случайных данных является пользователь компьютера, точнее – временные характеристики его реакции. За одно нажатие клавиши при этом может быть выработано всего несколько битов случайных данных, поэтому общая скорость выработки ключевой информации при этом невелика – до нескольких бит в секунду. Очевидно, данный подход не годится для получения больших массивов ключей.

В случае же, когда необходимо выработать большой по объему массив ключевой информации, возможно и очень широко распространено использование различных программных датчиков псевдослучайных чисел. Поскольку от подобного датчика требуются высокие показатели криптографической стойкости, естественным является использование в качестве него генератора гаммы самого шифра – просто “*нарезаем*” вырабатываемую шифром гамму на “*куски*” нужного размера, для *ГОСТ 28147-89* – по *32* байта. Конечно, для такого подхода нам потребуется *мастер-ключ*, который можно получить описанным выше методом “*электронной рулетки*”,

а с его помощью, используя шифр в режиме генератора гаммы, получаем массив ключевой информации нужного нам объема. Так эти два способа выработки ключей, – *ручной* и а*лгоритмический*, – работают в тандеме, дополняя друг друга. Схемы генерации ключей в *малобюджетных* системах криптографической защиты информации практически всегда построены по такому принципу.

Таблица замен

Таблица замен является долговременным ключевым элементом, то есть действует в течение гораздо более длительного срока, чем *отдельный (сеансовый) ключ*. Предполагается, что она является общей для всех узлов шифрования в рамках одной системы криптографической защиты. Даже при нарушении конфиденциальности таблицы замен стойкость шифра остается чрезвычайно высокой и не снижается ниже допустимого предела. Поэтому нет особой нужды держать в отдельных случаях таблицу в секрете, и в большинстве коммерческих применений *ГОСТ 28147-89* так оно и делается. С другой стороны, таблица замен является критически важным элементом для обеспечения стойкости всего шифра. Выбор ненадлежащей таблицы может привести к тому, что шифр будет легко вскрываться известными методами криптографического анализа. Критерии выработки узлов замен – *тайна за семью печатями* и *ФАПСИ* вряд ли ей поделится с общественностью в ближайшем обозримом будущем. В конечном итоге, для того, чтобы сказать, является ли данная конкретная таблица замен хорошей или плохой, необходимо провести огромный объем работ – многие тысячи человеко- и машино-часов. Единожды выбранная и используемая таблица подлежит замене в том и только в том случае, если шифр с ее использованием оказался уязвимым к тому или иному виду криптографического анализа. Поэтому лучшим выбором для рядового пользователя шифра будет взять одну из нескольких таблиц, ставших достоянием гласности. Например, из стандарта на хеш-функцию, она же "*центробанковская*" (узлы замены определенные документом *RFC 4357*). Сведения об этих таблицах можно найти в открытой печати и даже в интернете.

Для тех же, кто не привык идти легкими путями, ниже приведена общая схема получения качественных таблиц:

1. С помощью той или иной методики сформировать комплект из восьми узлов замен с гарантированными характеристиками не-

линейности. Таких методик существует несколько, одна из них – использование так называемых *бент-функций* [44, 47].

2. Проверить выполнение простейших критериев качества, например, тех, что опубликованы для узлов замены *DES*. Вот еще несколько общих соображений на этот счет: каждый узел замен может быть описан четырьмя логическими функциями от четырех логических аргументов. Если эти функции, записанные в *минимальной форме* (т.е. с минимально возможной длиной выражения) окажутся недостаточно сложными, такой узел замены отвергается. Кроме того, отдельные функции в пределах всей таблицы замен должны отличаться друг от друга в достаточной степени. На этом этапе отсеиваются многие заведомо некачественные таблицы.

3. Для шифра с выбранными таблицами необходимо построить различные модели раунда, соответствующие разным видам криптографического анализа, и измерить соответствующие *профильные* характеристики. Так, для линейного криптографического анализа строится линейный статистический аналог раунда шифрования и вычисляется профильная характеристика – показатель нелинейности. Если она оказывается недостаточной, таблица замен отвергается.

4. Наконец, используя результаты предыдущего пункта, необходимо тщательно исследовать шифр с выбранной таблицей (попыткой криптографического анализа всеми известными методами). Именно этот этап является наиболее сложным и трудоемким. Но если он сделан качественно, то с высокой степенью вероятности можно констатировать, что шифр с выбранными таблицами не будет вскрыт даже спецслужбам.

Можно, однако, поступить гораздо проще. Все дело в том, что чем больше в шифре раундов, тем меньшее влияние на стойкость всего шифра имеют характеристики стойкости одного раунда. В *ГОСТ 28147-89 – 32* раунда, это больше, чем практически во всех шифрах с аналогичной архитектурой. Поэтому для большинства бытовых и коммерческих применений бывает достаточно получить узлы замен как независимые случайные перестановки чисел от *0* до *15*. Это может быть практически реализовано, например, с помощью перемешивания колоды из шестнадцати карт, за каждой из которых закреплено одно из значений указанного диапазона.

Относительно таблицы замен необходимо отметить еще один интересный факт. Для обратимости циклов зашифрования “*32-З*” и расшифрования “*32-Р*” не требуется, чтобы узлы замен были перестановками чисел от *0* до *15*. Все работает даже в том случае, ес-

ли в узле замен есть повторяющиеся элементы, и замена, определяемая таким узлом, необратима, – однако в этом случае снижается стойкость шифра. Для этого достаточно попытаться сначала зашифровать, а затем расшифровать блок данных, используя такую "*неполноценную*" таблицу замен, узлы которой содержат повторяющиеся значения.

Использование ГОСТ 28147-89

Очень часто для использования в системе криптографической защиты данных требуется алгоритм с большим, чем *ГОСТ 28147-89* быстродействием реализации, и при этом не требуется такая высокая криптографическая стойкость. Типичным примером подобных задач являются различного рода электронные биржевые торговые системы, управляющие торговыми сессиями в реальном времени. Здесь от использованных алгоритмов шифрования требуется, чтобы было невозможно расшифровать оперативные данные системы в течение сессии (данные о выставленных заявках, о заключенных сделках и т.п.), по ее истечении же эти данные, как правило, уже бесполезны для злоумышленников. Другими словами, требуется гарантированная стойкость всего на несколько часов – такова типичная продолжительность торговой сессии. Ясно, что использование полновесного *ГОСТ 28147-89* в этой ситуации было бы нецелесообразным.

Как поступить в этом и аналогичном ему случаях, чтобы увеличить быстродействие шифрования? Ответ лежит на поверхности – использовать модификацию шифра с меньшим количеством основных шагов (раундов) в базовых циклах. Во сколько раз мы уменьшаем число раундов шифрования, во столько же раз возрастает быстродействие. Указанного изменения можно достигнуть двумя путями: уменьшением длины ключа и уменьшением числа "*циклов просмотра*" ключа. Известно, что количество основных шагов в базовых циклах шифрования равно $N=n\cdot m$, где n – число 32-битовых элементов в ключе, m – число циклов использования ключевых элементов, в стандарте $n = 8$, $m = 4$. Можно уменьшить любое из этих чисел, но простейший вариант – уменьшать длину ключа, не трогая схемы его использования.

Понятно, что платой за ускорение работы будет снижение стойкости шифра. Основная трудность заключается в том, что достаточно сложно более или менее точно оценить величину этого сниже-

ния. Очевидно, единственно возможный способ сделать это – провести исследование вариантов шифра с редуцированными циклами криптографического преобразования "*по полной программе*". Понятно, что, во-первых, это требует использования закрытой информации, которой владеют только разработчики *ГОСТ 28147-89*, и, во-вторых, очень трудоемко. Поэтому попытаемся дать оценку, исходя лишь из общих закономерностей.

Что касается устойчивости шифра к взлому "*экстенсивными*" методами, то есть к атаке "*грубой силы*", тот тут все более или менее ясно: ключ размером *64* бита находится где-то на грани доступности этому виду атаки, шифр с ключом *96* бит и выше (ключ должен содержать целое число *32*-битовых элементов) вполне устойчив против него. Действительно, несколько лет назад прежний стандарт шифрования *США*, *DES*, был неоднократно взломан путем перебора ключа: сначала его взломала вычислительная сеть, организованная на базе глобальной сети *Интернет*, а затем – специализированная, т.е. сконструированная специально для этого вычислительная машина. Примем, что стандартный вариант *ГОСТ 28147-89* при программной реализации на современных процессорах работает вчетверо быстрее *DES*. Тогда *8*-раундовый "*редуцированный ГОСТ*" будет работать в *16* раз быстрее *DES*. Примем также, что за прошедшее с момента взлома *DES* время, производительность вычислительной техники согласно *закону Мура* возросла вчетверо. Получаем в итоге, что сейчас проверка одного *64*-битового ключа для "*редуцированного ГОСТа*" с восемью циклами осуществляется в *64* раза быстрее, чем в свое время выполнялась проверка одного ключа *DES*. Таким образом, преимущество такого варианта *ГОСТ 28147-89* перед *DES* по трудоемкости атаки "*грубой силы*" сокращается с $2^{64-56} = 2^8 = 256$ до $256/64 = 4$ раз. Это весьма иллюзорное различие, почти что ничего.

Гораздо сложнее оценить устойчивость ослабленных модификаций *ГОСТ 28147-89* к *интенсивным* способам криптографического анализа. Однако общую закономерность можно проследить и здесь. Дело в том, что "*профильные*" характеристики многих наиболее сильных на сегодняшний момент видов криптографического анализа зависят экспоненциально от числа раундов шифрования. Так, для линейного криптографического анализа это будет характеристика линейности L:

$$L = C \cdot e^{-\alpha \cdot r},$$

где C и α – константы; r – количество раундов.

Аналогичная зависимость существует и для дифференциального криптографического анализа. По своему *физическому смыслу* все характеристики такого рода – вероятностные. Обычно объем необходимых для криптографического анализа исходных данных и его трудоемкость обратно пропорциональны подобным характеристикам. Отсюда следует, что эти показатели трудоемкости растут экспоненциально с ростом числа основных шагов шифрования. Поэтому при снижении количества раундов в несколько раз трудоемкость наиболее известных видов анализа изменится (очень приблизительно и грубо) как корень этой степени из первоначального количества. Это очень большое падение стойкости.

С другой стороны, *ГОСТ 28147-89* проектировался с большим запасом прочности и на сегодняшний день устойчив ко всем известным видам криптографического анализа, включая дифференциальный и линейный. Применительно к линейному криптографическому анализу это означает, что для его успешного проведения требуется больше пар "*открытый блок – зашифрованный блок*", чем существует, то есть более 2^{64}. С учетом сказанного выше это означает, что для успешного линейного криптографического анализа *16*-раундового *ГОСТ 28147-89* потребуется не менее $\sqrt{2^{64}} = 2^{32}$ блоков или 2^{35} байтов или *32* Гбайта данных, а для *8*-раундового – не менее $\sqrt[4]{2^{64}} = 2^{16}$ блоков или 2^{19} байтов или *0,5* Мбайт.

Выводы из всего, сказанного выше, приведены в табл. 7.2, обобщающей характеристики редуцированных вариантов *ГОСТ 28147-89*.

Примечание. Размер ключа при неизменной схеме его использования, предполагающей сначала просмотр ключа *3* раза в прямом направлении, затем один раз в обратном. Индекс быстродействия показывает, во сколько раз возрастет быстродействие шифрования по сравнению со стандартным *32*-раундовым вариантом.

Два последних варианта, с *12* и *8* раундами, способны обеспечить весьма и весьма ограниченную во времени защиту. Их использование оправдано лишь в задачах, где требуется лишь краткосрочная секретность закрываемых данных, порядка нескольких часов. Возможная область применения этих слабых вариантов шифра – закрытие *UDP*-трафика электронных биржевых торговых систем. В этом случае каждый пакет данных (*datagram*, средняя *D* из аббревиатуры *UDP*) шифруется на отдельном *64*-битовом ключе, а сам

ключ шифруется на сеансовом ключе (ключе, область действия которого – один сеанс связи между двумя компьютерами) и передается вместе с данными.

Таблица 7.2

Обобщающие характеристики сокращенных вариантов *ГОСТ 28147-89*

Число раундов	*Размер ключа, бит*	*Индекс быстродействия*	*Вероятные характеристики шифра(очень грубая оценка)*
24	*192*	*1,33*	*Устойчив к большинству известных видов криптографического анализа, или находится на грани устойчивости. Практическая реализация криптографического анализа невозможна из-за высоких требований к исходным данным и трудоемкости.*
16	*128*	*2*	*Теоретически неустойчив к некоторым видам криптографического анализа, однако их практическая реализация в большинстве случаев затруднена из-за высоких требований к исходным данным и трудоемкости.*
12	*95*	*2,67*	*Неустойчив к некоторым известным видам криптографического анализа, однако годится для обеспечения секретности небольших объемов данных (до десятков-сотен Кбайт) на короткий срок.*
8	*64*	*4*	*Неустойчив к некоторым известным видам криптографического анализа, однако годится для обеспечения секретности небольших объемов данных (до десятков Кбайт) на короткий срок.*

У рассматриваемого вопроса есть и обратная сторона. Что если скорость шифрования некритична, а требования к стойкости весьма жестки? Повысить стойкость *ГОСТ 28147-89* можно двумя путями – условно назовем их “э*кстенсивный*” и “*интенсивный*”.

Первый из них – это не что иное, как простое увеличение числа раундов шифрования. Однако стандарт *ГОСТ 28147-89* и без этого обеспечивает необходимую устойчивость. Указанный подход позволяет реально увеличить стойкость шифра: если раньше криптографический анализ был просто невозможным, то теперь он вообще невозможен.

Более хитрым и интересным является вопрос, а можно ли увеличить стойкость шифра, не меняя количества и структуры основных шагов шифрования. Как ни удивительно, ответ на этот него положительный. Дело в том, что в *ГОСТ 28147-89* на основном шаге преобразования предполагается выполнение замены *4* на *4* бит, а на практике все программные реализации выполняют замену побайтно, т.е. *8* на *8* бит – так делается по соображениям эффективности. Если сразу спроектировать такую замену как 8-битовую, то мы существенно улучшим характеристики одного раунда.

Во-первых, увеличится "*диффузионная*" характеристика или показатель "*лавинности*" – один бит исходных данных и/или ключа будет влиять на большее количество бит результата.

Во-вторых, для больших по размеру узлов замены можно получить более низкие дифференциальную и линейную характеристики, уменьшив тем самым подверженность шифра одноименным видам криптографического анализа. Особенно актуально это для редуцированных (сокращенных) циклов *ГОСТ 28147-89*, а для *8*- и *12*-раундовых вариантов такой шаг просто необходим. Это несколько скомпенсирует потерю стойкости в них от уменьшения количества раундов. Затрудняет использование этого приема – то, что конструировать подобные *увеличенные* узлы замены придется самостоятельно. А также то, что более крупные узлы вообще конструировать заметно труднее, чем меньшие по размеру.

7.7.4. Нестандартное использование ГОСТ 28147-89

Безусловно, основное назначение криптографических алгоритмов *ГОСТ 28147-89* – это шифрование и имитозащита данных. Однако им можно найти и другие применения, связанные, естественно, с защитой информации. Коротко расскажем о них:

1. Для шифрования в режиме гаммирования *ГОСТ 28147-89* предусматривает выработку криптографической гаммы – последовательности бит с хорошими статистическими характеристиками, обладающей высокой криптографической стойкостью. Далее эта

гамма используется для модификации открытых данных, в результате чего получаются зашифрованные данные. Однако, это не единственное возможное применение криптографической гаммы. Дело в том, что алгоритм ее выработки – это *генератор последовательности псевдослучайных чисел* с великолепными характеристиками. Конечно, использовать такой *ГППСЧ* там, где требуются только получение статистических характеристик вырабатываемой последовательности, а криптографическая стойкость не нужна, не очень разумно – для этих случаев имеются более эффективные генераторы. Но для разных применений, связанных с защитой информации, такой источник будет кстати.

Как уже отмечалось выше, гамму можно использовать как "*сырье*" для выработки ключей. Для этого нужно лишь получить отрезок гаммы нужной длины – *32* байта. Таким способом ключи можно изготавливать по мере необходимости и их не надо будет хранить, – если такой ключ понадобится повторно, будет достаточно легко его выработать снова. Надо только будет вспомнить, на каком ключе он был выработан исходно, какая использовалась синхропосылка, и с какого байта выработанной гаммы начинался ключ. Вся информация, кроме использованного ключа, несекретна. Данный подход позволит легко контролировать достаточно сложную и разветвленную систему ключей, используя всего лишь один *мастер-ключ*.

Аналогично предыдущему, гамму можно использовать в качестве исходного "*сырья*" для выработки паролей. Тут может возникнуть вопрос, зачем вообще нужно их генерировать, не проще ли по мере надобности их просто выдумывать. Несостоятельность такого подхода была наглядно продемонстрирована серией инцидентов в компьютерных сетях, самым крупным из которых был суточный паралич интернета в ноябре 1988 года, вызванный "*червем Моррис*а". Одним из способов проникновения злоумышленной программы на компьютер был подбор паролей: программа пыталась войти в систему, последовательно перебирая пароли из своего внутреннего списка в несколько сотен, причем в значительной доле случаев ей это удавалось сделать. Фантазия человека по выдумыванию паролей оказалась весьма бедной. Именно поэтому в тех организациях, где безопасности уделяется должное внимание, пароли генерирует и раздает пользователям системный администратор по безопасности. Выработка паролей чуть сложнее, чем выработка ключей, так как при этом "*сырую*" двоичную гамму необходимо преобразовать к символьному виду, а не просто "*нарезать*" на куски.

Кроме того, отдельные значения, возможно, придется отбросить, чтобы обеспечить равную вероятность появления всех символов алфавита в пароле.

Еще один способ использования криптографической гаммы – гарантированное затирание данных на магнитных носителях. Дело в том, что даже при перезаписи информации на магнитном носителе остаются следы предыдущих данных, которые может восстановить соответствующая экспертиза. Для уничтожения этих следов такую перезапись надо выполнить многократно. Оказалось, что потребуется перезаписывать информацию на носитель меньшее количество раз, если при такой процедуре использовать случайные или псевдослучайные данные, которые останутся неизвестными экспертам, пытающимся восстановить затертую информацию. Гамма шифра здесь будет как нельзя кстати.

2. Не только криптографическая гамма, но и само криптографическое преобразование, может быть использовано для нужд, непосредственно не связанных с шифрованием.

Известно, что один из таких вариантов использования *ГОСТ*а – выработка имитовставки для массивов данных. Однако на базе любого блокового шифра, и *ГОСТ*а в том числе, достаточно легко построить схему вычисления односторонней хэш-функции, называемой также в литературе *MDC*, что в разных источниках расшифровывается как *код обнаружения изменений*/*манипуляций* (*Modification/Manipulation Detection Code*) или *дайджест сообщения* (*Message Digest Code*).

MDC может непосредственно использоваться в системах имитозащиты в качестве аналога имитовставки, не зависящего, однако, от секретного ключа. Кроме того, *MDC* широко используется в схемах *электронной цифровой подписи* (*ЭЦП*). Для удобства большинство таких схем сконструированы таким способом, что можно подписывать блок данных фиксированного размера. Как известно, на базе обсуждаемого стандарта *ГОСТ 28147-89* построен стандарт Российской Федерации по вычислению односторонней хэш-функции *ГОСТ Р 34.11-94* [12].

Контрольные вопросы и задания

1. Для каких целей может использоваться алгоритм криптографического преобразования данных *ГОСТ 28147-89*?

2. Перечислите основные параметры алгоритма симметричного шифрования *ГОСТ 28147-89*.

3. Какие операции используются в блочном алгоритме шифрования *ГОСТ 28147-89*?

4. Чем отличается последний цикл (*32*-й) зашифрования и расшифрования данных от всех предыдущих?

5. Для чего используется таблица замен в алгоритме *ГОСТ 28147-89*?

6. Какие основные отличия алгоритма шифрования *ГОСТ 28147-89* от алгоритма *DES*?

7. Какая информация, кроме секретного ключа, при условии использования стандартных алгоритмов *ГОСТ 28147-89* необходимая для расшифрования сообщения?

8. В каких режимах может проводиться шифрование данных с помощью алгоритма криптографического преобразования *ГОСТ 28147-89*?

9. Чем отличаются режимы работы гаммирования и гаммирование с обратной связью в алгоритме *ГОСТ 28147-89*?

10. Что такое имитовставка? С какой целью может быть использована имитовставка?

11. В регистре N_1 алгоритма *ГОСТ 28147-89* находятся данные, которые в шестнадцатеричном исчислении имеют вид: N_1 – *191a2ab8*, а в N_2 – *434665b2*. Ключ для шифрования в шестнадцатеричном исчислении имеет вид: k_0=*eb8a7159*, k_1=*7ce5d63d*, k_2=*4ac1d6e0*, k_3=*bafe4731*, k_4=*a3deb025*, k_5=*8bb389ac*, k_6=*10d3b61a*, k_7=*e9ac340f*. Цикл зашифрования – *25*. Что будет находится в N_1 и N_2 после завершения цикла. В качестве блоков замены использовать табл. 7.1.

12. В регистре N_1 алгоритма *ГОСТ 28147-89* находятся данные, которые в шестнадцатеричном исчислении имеют вид: N_1 – *434665b2*, а в N_2 – *191a2ab8*. Ключ для шифрования в шестнадцатеричном исчислении имеет вид: k_0=*7ce5d63d*, k_1=*4ac1d6e0*, k_2=*bafe4731*, k_3=*a3deb025*, k_4=*8bb389ac*, k_5=*9ea2923b*, k_6=*9a62e045*, k_7=*e9ac340f*. Цикл расшифрования – *9*. Что будет находится в N_1 и N_2 после завершения цикла. В качестве блоков замены использовать табл. 7.1.

13. В регистре N_1 алгоритма *ГОСТ 28147-89* находятся данные, которые в шестнадцатеричном исчислении имеют вид: N_1 – *10d3b61a*, а в N_2 — *eb8a7159*. Ключ для шифрования в шестнадцатеричном исчислении имеет вид: k_0=*e9ac340f*, k_1=*a3deb025*, k_2=*8bb389ac*, k_3=*9ea2923b*, k_4=*9a62e045*, k_5=*bafe4731*, k_6=*7ce5d63d*, k_7=*4ac1d6e0*. Цикл зашифрования – *32*. Что будет находится в N_1 и N_2 после завершения цикла. В качестве блоков замены использовать табл. 7.1.

14. В регистре N_1 алгоритма *ГОСТ 28147-89* находятся данные, которые в шестнадцатеричном исчислении имеют вид: N_1 – *0d3b61a5*, а в N_2 — *eb8a7159*. Ключ для шифрования в шестнадцатеричном исчислении имеет вид: k_0=*e9ac340f*, k_1=*9a62e045*, k_2=*10d3b61a*, k_3=*8bb389ac*, k_4=*9ea2923b*, k_5=*7ce5d63d*, k_6= *eb8a7159*, k_7=*bafe4731*. Цикл расшифрования – *32*. Что будет находится в N_1 и N_2 после завершения цикла. В качестве блоков замены использовать табл. 7.1.

15. В алгоритме *ГОСТ 28147-89* вероятность навязывания ложных помех должна быть не менее $P_{ЛП} = 2{,}4 \cdot 10^{-10}$. Определить длину имитовставки в битах.

16. В алгоритме *ГОСТ 28147-89* формируется имитовставка длиной *24* бита. Определить вероятность навязывания ложных помех.

Раздел 8

БЛОЧНЫЙ СИММЕТРИЧНЫЙ КРИПТОГРАФИЧЕСКИЙ АЛГОРИТМ RIJNDAEL И СТАНДАРТ AES

8.1. ИСТОРИЯ РАЗРАБОТКИ СТАНДАРТА AES

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 криптографическому алгоритму в мире [38]. *DES* считается устаревшим по многим параметрам: длине ключа, удобству реализации на современных процессорах, быстродействию и др., за исключением самого главного – стойкости. За *23* года интенсивного криптографического анализа не было найдено методов вскрытия этого шифра, существенно отличающихся по эффективности от полного перебора по ключевому пространству.

Требования к кандидатам были следующие:

- криптографический алгоритм должен быть открыто опубликован;
- криптографический алгоритм должен быть симметричным блочным шифром, допускающим размеры ключей в *128, 192* и *256* бит;
- криптографический алгоритм должен быть предназначен как для аппаратной, так и для программной реализации;
- криптографический алгоритм должен быть доступен для открытого использования в любых продуктах, а значит, не может быть запатентован, в противном случае патентные права должны быть аннулированы;
- криптографический алгоритм должен подвергаться изучению по следующим параметрам: стойкости, стоимости, гибкости, реализуемости в *smart*-картах.

Стойкость. Это самый важный критерий в оценке алгоритма. Оценивались: способность шифра противостоять различным методам криптографического анализа; статистическая безопасность и относительная защищенность по сравнению с другими кандидатами.

Стоимость. Не менее важный критерий, учитывая одну из основных целей *NIST,* – широкая область использования и доступность *AES*. Стоимость зависит от вычислительной эффективности (в первую очередь быстродействия) на различных платформах, удобства программной и аппаратной реализации, низких требований к памяти, простоты (простые алгоритмы легче реализовывать, они более прозрачны для анализа).

Гибкость. Гибкость включает способность алгоритма обрабатывать ключи больше оговоренного минимума (*128* бит), надежность и эффективность выполнения в разных средах, возможность реализации других криптографических функций: комбинированного шифрования, хеширования и т.д.

Другими словами, *AES* должен быть существенно более эффективным с точки зрения практической реализации (в первую очередь скорости шифрования и формирования ключей), иметь больший запас прочности, чем *TripleDES*, при этом не уступая ему в стойкости.

Реализуемость в smart-картах. Важная область использования *AES* в будущем – s*mart*-карты, при этом главной проблемой является небольшой объем доступной памяти. *NIST* исходил из допущения, что некоторые дешевые карты могут иметь всего *256* байт *RAM* (для вычисляемых данных) и *2000 ROM* (для хранения алгоритмов и констант). Существует два основных метода формирования раундовых ключей:

- вычисление на начальном этапе работы криптографического алгоритма и хранение в памяти;

- вычисление раундовых ключей "*на лету*".

Ясно, что второй вариант уменьшает затраты *RAM*, и поэтому наличие такой возможности в криптографическом алгоритме является его несомненным достоинством.

На открытый конкурс были приняты *15* алгоритмов, разработанных криптографами *12* стран – *Австралии, Бельгии, Великобритании, Германии, Израиля, Канады, Коста-Рики, Норвегии, США, Франции, Южной Кореи* и *Японии*.

В финал конкурса вышли следующие алгоритмы: *Mars*, *Twofish* и *RC6* (*США*), *Rijndael* (*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; *Rijndael* и *Serpent* являются классическими *SP*-сетями. *Mars* и *Twofish* имеют самую сложную конструкцию, *Rijndael* и *RC6* – самую простую.

Финалисты будут описаны по единой схеме, данной в документе *NIST*. Сначала описываются обнаруженные "*слабости*" алгоритма (если таковые имеются), затем преимущества и, наконец, недостатки.

Mars выставлен на конкурс фирмой *IBM*, одним из авторов шифра является *Д. Копперсмит* (англ. *Don Coppersmith*), участник разработки *DES*. В алгоритме не обнаружено слабостей в защите.

Преимущества:

- высокий уровень защищенности;
- высокая эффективность на *32*-разрядных платформах, особенно поддерживающих операции умножения и циклического сдвига;
- потенциально поддерживает размер ключа больше *256* бит.

Недостатки:

- сложность алгоритма затрудняет анализ его надежности;
- снижение эффективности на платформах без необходимых операций;
- сложность защиты от временного анализа и анализа мощности.

RC6 предложен фирмой *RSA Lab*, одним из его авторов является *Р. Ривест*. В алгоритме не обнаружено слабостей в защите.

Преимущества:

- высокая эффективность на *32*-битовых платформах, особенно поддерживающих операции умножения и циклических сдвигов;
- простая структура алгоритма упрощает анализ его надежности;
- наличие хорошо изученного предшественника – *RC5*;
- быстрая процедура формирования ключа;
- потенциально поддерживает размер ключа больше *256* бит;
- длина ключа и число раундов могут быть переменными.

Недостатки:

- относительно низкий уровень защищенности;
- снижение эффективности на платформах, не имеющих необходимых операций;

- сложность защиты от временного анализа и анализа мощности;

- невозможность генерации раундовых ключей "*на лету*".

Rijndael большинством участников конкурса назван как лучший выбор, если будет отвергнут их собственный шифр. Основан на шифре *Square* тех же авторов. В алгоритме не обнаружено слабостей в защите.

Преимущества:

- высокая эффективность на любых платформах;

- высокий уровень защищенности;

- хорошо подходит для реализации в *smart-картах* из-за низких требований к памяти;

- быстрая процедура формирования ключа;

- хорошая поддержка параллелизма на уровне инструкций;

- поддержка разных длин ключа с шагом *32* бита.

Недостатки:

- уязвим к анализу мощности.

Serpent – разработка профессиональных криптографических аналитиков *Р. Андерсона, Э. Бихама и Л. Кнудсена*. Создав шифр, успешно противостоящий всем известным на сегодня атакам, разработчики затем удвоили количество его раундов. В алгоритме не обнаружено слабостей в защите.

Преимущества:

- высокий уровень защищенности;

- хорошо подходит для реализации в *smart-картах* из-за низких требований к памяти.

Недостатки:

- самый медленный алгоритм среди финалистов;

- уязвим к анализу мощности.

Twofish основан на широко используемом шифре *Blowfish*; один из авторов разработки – *Б. Шнайер*. Главная особенность шифра – изменяющиеся в зависимости от секретного ключа таблицы замен. В алгоритме не обнаружено слабостей в защите.

Преимущества:

- высокий уровень защищенности;

- хорошо подходит для реализации в *smart-картах* из-за низких требований к памяти;

- высокая эффективность на любых платформах, в том числе на ожидаемых в будущем *64*-разрядных архитектурах фирм *Intel* и *Motorola*;

- поддерживает вычисление раундовых ключей "*на лету*";
- поддерживает распараллеливание на уровне инструкций;
- допускает произвольную длину ключа до *256* бит.

Недостатки:

- особенности алгоритма затрудняют его анализ;
- высокая сложность алгоритма;
- применение операции сложения делает алгоритм уязвимым к анализу мощности и временному анализу.

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 Раймен* (*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. [38].

8.2. МАТЕМАТИЧЕСКИЕ ПРЕДПОСЫЛКИ AES

Материалы данного раздела в значительной степени основываются на авторском описании алгоритма *Rijndael* [38] и документе *FIPS-197*. Везде в дальнейшем изложении, где стандарт *AES* совпадает с алгоритмом *Rijndael*, упоминается только последний. Все отличия принятого стандарта от криптографического алгоритма *Rijndael* оговариваются особо.

Поскольку эта криптографическая система, относясь к блочным алгоритмам, имеет много общего с *DES*, хотя и не является непосредственным обобщением шифра *Фейстеля*. Для обеспечения

криптографической стойкости алгоритм *Rijndael* включает в себя повторяющиеся раунды, каждый из которых состоит из замен, перестановок и сложения с ключом. Кроме того, *Rijndael* использует сильную математическую структуру: большинство его операций основаны на арифметике поля $GF(2^8)$. Однако, в отличие от *DES*, зашифрование и расшифрование в этом алгоритме – процедуры разные.

Алгоритм *Rijndael* оперирует байтами, которые рассматриваются как элементы конечного поля $GF(2^8)$. Арифметические операции в поле соответствуют операциям над двоичными многочленами из $GF(2)$ по модулю неприводимого полинома

Элементами поля являются многочлены степени не более 7, которые могут быть заданы строкой своих коэффициентов. Если представить байт в виде

$$\{ a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0 \},\ a_i \in \{ 0,1 \},\ i = 0,1,\ldots,7,$$

то элемент поля описывается многочленом

$$a_7 \cdot x^7 + a_6 \cdot x^6 + a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 .$$

Например, байту $\{ 11001011 \}$ (или $\{ cb \}$ в шестнадцатеричной форме) соответствует многочлен $x^7 + x^6 + x^3 + x + 1$.

Для элементов конечного поля определены аддитивные и мультипликативные операции.

8.2.1 Аддитивные операции

Сложение элементов конечного поля – суть операция поразрядного *xor* и поэтому обозначается как $\oplus$. Пример выполнения операции сложения:

- в виде многочленов

$$(x^6 + x^4 + x^2 + x + 1) \oplus (x^7 + x + 1) = x^7 + x^6 + x^4 + x^2;$$

- в виде двоичного представления

$$\{ 01010111 \} \oplus \{ 10000011 \} = \{ 11010100 \};$$

- в виде шестнадцатеричного представления

$$\{ 57 \} \oplus \{ 83 \} = \{ d4 \}.$$

В конечном поле для любого ненулевого элемента α существует обратный элемент $-\alpha$, при этом $\alpha + (-\alpha) = 0$, где нулевой элемент – это $\{00\}$. В $GF(2^8)$ справедливо $\alpha + \alpha = 0$, т.е. каждый ненулевой элемент является своей собственной аддитивной инверсией.

Сложение двух многочленов с коэффициентами из $GF(2^8)$ – суть операция сложения многочленов с приведением подобных членов в поле $GF(2^8)$ т.е.

$$a(x) + b(x) = (a_7 \oplus b_7)\cdot x^7 + (a_6 \oplus b_6)\cdot x^6 + (a_5 \oplus b_5)\cdot x^5 +$$
$$+ (a_4 \oplus b_4)\, x^4 + (a_3 \oplus b_3)\cdot x^3 + (a_2 \oplus b_2)\cdot x^2 +$$
$$+ (a_1 \oplus b_1)\cdot x + (a_0 \oplus b_0).$$

Таким образом, сложение двух многочленов – суть операция поразрядного *xor* этих коэффициентов полиномов с приведением подобных членов в поле $GF(2^8)$.

8.2.2 Мультипликативные операции

Умножение элементов конечного поля, обозначаемое далее как $\bullet$, более сложная операция. Умножение в поле $GF(2^8)$ – это операция умножения многочленов со взятием результата по модулю неприводимого многочлена $p(x)$ восьмой степени и с использованием операции *xor* при приведении подобных членов. В *Rijndael* выбран $p(x) = x^8 + x^4 + x^3 + x + 1$, или в шестнадцатеричной форме $1\{1b\}$. Запись $1\{1b\}$ обозначает, что присутствует "*лишний*" девятый бит.

Многочлен $p(x) = x^8 + x^4 + x^3 + x + 1$, используемый для построения поля $GF(2^8)$, является первым неприводимым многочленом восьмой степени, упоминающимся в большинстве справочников. То есть его выбор достаточно произволен.

Пример операции умножения:

$$\{57\} \bullet \{83\} = \{c1\},$$

так как

$$(x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod p(x) = x^7 + x^6 + 1,$$

где

$$x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1 =$$
$$= (x^5 + x^3)\cdot(x^8 + x^4 + x^3 + x + 1) \oplus (x7 + x^6 + 1),$$

а шестнадцатеричное представление полинома $x^7 + x^6 + 1$ соответствует значению *{ c1 }*.

Для любого ненулевого элемента α справедливо $\alpha \cdot 1 = \alpha$. Мультипликативной единицей в $GF(2^8)$ является элемент *{ 01 }*.

Операция умножения может быть описана и реализована по-другому. Умножая произвольный многочлен седьмой степени

$$a_7 \cdot x^7 + a_6 \cdot x^6 + a_5 \cdot x^5 + a_4 \cdot x^4 + a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

на x, получим

$$a_7 \cdot x^8 + a_6 \cdot x^7 + a_5 \cdot x^6 + a_4 \cdot x^5 + a_3 \cdot x^4 + a_2 \cdot x^3 + a_1 \cdot x^2 + a_0 \cdot x.$$

Приводя полученный многочлен по модулю $p(x) = 1$ *{ 1b }*, получим результат произведения в конечном поле $GF(2^8)$. Для этого при $a_7 = 1$ достаточно вычесть (применить операцию поразрядного *xor*) $p(x)$ из полученного выше многочлена. Если же $a_7 = 0$, то результат уже получается приведенным. Тогда умножение на x (т.е. *{ 00000010 }* в двоичной или *{ 02 }* в шестнадцатеричной форме) получается сдвигом влево и возможно последующим применением *xor* с многочленом *1 { 1b }*.

Пусть функция *xtime()* осуществляет операцию умножения на x вышеописанным способом. Применяя функцию *xtime()* n раз можно получить результат умножения на x^n, а суммируя различные степени x, можно получить любой элемент поля. Например:

$$\{ 57 \} \bullet \{ 13 \} = \{ fe \},$$

так как

$$\{ 57 \} \bullet \{ 02 \} = xtime(\{ 57 \}) = \{ ae \};$$

$$\{ 57 \} \bullet \{ 04 \} = xtime(\{ ae \}) = \{ 47 \};$$

$$\{ 57 \} \bullet \{ 08 \} = xtime(\{ 47 \}) = \{ 8e \};$$

$$\{ 57 \} \bullet \{ 10 \} = xtime(\{ 8e \}) = \{ 07 \}.$$

Откуда

$$\{ 57 \} \bullet \{ 13 \} = \{ 57 \} \bullet (\{ 01 \} \oplus \{ 02 \} \oplus \{ 10 \}) =$$

$$= (\{ 57 \} \bullet \{ 01 \}) \oplus (\{ 57 \} \bullet \{ 02 \}) \oplus (\{ 57 \} \bullet \{ 10 \}) =$$

$$= \{ 57 \} \oplus \{ ae \} \oplus \{ 07 \} = \{ fe \}.$$

Следовательно, произведение в поле $GF(2^8)$ – это обычная операция умножения многочленов со взятием результата по модулю некоторого неприводимого многочленов и с использованием операции *xor* для приведения подобных членов [1].

Раундовые преобразования *Rijndael* оперируют *32*-разрядными словами. В алгоритме *32*-битовые слова отождествляются с многочленами степени *3* из поля $GF(2^8)$ [38]. Отождествление делается в формате "*перевертыш*", т.е. старший (наиболее значимый) бит соответствует младшему коэффициенту многочлена. Так, например, слово

$$a_0||a_1||a_2||a_3$$

соответствует многочлену

$$a(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0 .$$

Арифметика в алгоритме *Rijndael* совпадает с арифметическими действиями в кольце многочленов $GF(2^8)$ по модулю многочлена $m(x) = x^4 + 1$. Заметим, что многочлен $m(x) = x^4 + 1 = (x + 1)^4$ приводим, и, следовательно, арифметические действия в алгоритме отличны от операций поля $GF(2^8)$, в частности, бывают пары ненулевых элементов, произведение которых равно *0* [38].

Умножение двух многочленов с коэффициентами из поля $GF(2^8)$ – более сложная операция. Предположим, перемножаются два многочлена

$$a(x) = a_3 \cdot x^3 + a_2 \cdot x^2 + a_1 \cdot x + a_0$$

и

$$b(x) = b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0 .$$

Результатом умножения

$$c(x) = a(x) \otimes b(x)$$

будет многочлен

$$c(x) = c_6 \cdot x^6 + c_5 \cdot x^5 + c_4 \cdot x^4 + c_3 \cdot x^3 + c_2 \cdot x^2 + c_1 \cdot x + c_0,$$

где

$$c_0 = a_0 \cdot b_0;$$

$$c_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1;$$
$$c_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2;$$

$$c_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3;$$

$$c_4 = a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3;$$

$$c_5 = a_3 \cdot b_2 \oplus a_2 \cdot b_3;$$

$$c_6 = a_3 \cdot b_3.$$

Для того чтобы результат умножения мог быть представлен *4*-байтовым словом, необходимо взять результат по модулю многочлена степени не более *4*. Авторы шифра выбрали многочлен

$$m(x) = x^4 + 1$$

для которого справедливо [30]

$$x^i \bmod (x^4 + 1) = x^{i \bmod 4}.$$

Таким образом, результатом $d(x)$ умножения $\otimes$ двух многочленов

$$d(x) = a(x) \otimes b(x)$$

по модулю $m(x) = x^4 + 1$ будет многочлен

$$d(x) = d_3 \cdot x^3 + d_2 \cdot x^2 + d_1 \cdot x + d_0,$$

где $d_0 = a_0 \cdot b_0 \oplus a_3 \cdot b_1 \oplus a_2 \cdot b_2 \oplus a_1 \cdot b_3;$

$d_1 = a_1 \cdot b_0 \oplus a_0 \cdot b_1 \oplus a_3 \cdot b_2 \oplus a_2 \cdot b_3;$

$d_2 = a_2 \cdot b_0 \oplus a_1 \cdot b_1 \oplus a_0 \cdot b_2 \oplus a_0 \cdot b_3;$

$d_3 = a_3 \cdot b_0 \oplus a_2 \cdot b_1 \oplus a_1 \cdot b_2 \oplus a_0 \cdot b_3.$

В матричной форме это может быть записано следующим образом

$$\begin{vmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \end{vmatrix} = \begin{vmatrix} a_0 & a_3 & a_2 & a_1 \\ a_1 & a_0 & a_3 & a_2 \\ a_2 & a_1 & a_0 & a_3 \\ a_3 & a_2 & a_1 & a_0 \end{vmatrix} \cdot \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{vmatrix}.$$

Пусть $b(x) = b_3 \cdot x^3 + b_2 \cdot x^2 + b_1 \cdot x + b_0$.

Умножению на x многочлена $b(x)$ с коэффициентами из поля $GF(2^4)$ по модулю $x^4 + 1$, учитывая свойства последнего, соответствует циклический сдвиг байтов в пределах слова в сторону старшего байта, так как

$$x \otimes b(x) = b_2 \cdot x^3 + b_1 \cdot x^2 + b_0 \cdot x + b_3.$$

8.2.3. Операции нахождения мультипликативных и аддитивных обратных величин

В конечном поле для любого ненулевого элемента $a(x)$ существует обратный (инверсный) аддитивный элемент – $a(x)$, при этом

$$a(x)+(-a(x)) \bmod p(x) = 0.$$

В поле $GF(2^8)$ справедливо $a(x) + a(x) = 0$, где нулевой элемент – это $\{ 00 \}$, то есть каждый ненулевой элемент является своей собственной аддитивной инверсией.

В шифре *Rijndael* также используется процедура нахождения элемента *мультипликативно обратного (инверсного) к данному*, то есть такого, для которого выполняется равенство

$$a(x) \bullet a^{-1}(x) \bmod p(x) = 1, \qquad (8.1)$$

где $a(x)$ – некоторый элемент поля $GF(2^8)$; $a^{-1}(x)$ – его мультипликативно обратный элемент.

Произведение (8.1) в поле $GF(2^8)$ просто выполнять, если рассматривать ненулевые элементы поля степени некоторого примитивного элемента ω. Для этого определим структуру устройства (генератора элементов поля), который позволяет поставить в соответствие каждого ненулевого элемента поля соответствующую степень примитивного элемента.

Имеем

$$\begin{aligned} p(x+1) &= (x+1)^8 + (x+1)^4 + (x+1)^3 + (x+1) + 1 = \\ &= (x^8 + 1) + (x^4 + 1) + (x^3 + x^2 + x + 1) + (x + 1) + 1 = \\ &= x^8 + x^4 + x^3 + x^2 + 1 = \tilde{p}(x). \end{aligned} \qquad (8.2)$$

Полином $\tilde{p}(x)$ примитивный, а потому, соответствие между различными формами представления элементов в поле $GF(2^8)$ можно получить, если модулировать работу устройства, приведенного на рис. 8.1.

00000001	*- { 01 }*	$(\omega^0 = 1)$
00000011	*- { 03 }*	(ω^1)
00000101	*- { 05 }*	(ω^2)
00001111	*- { 0f }*	(ω^3)
00010001	*- { 11 }*	(ω^4)
00110011	*- { 33 }*	(ω^5)
01010101	*- { 55 }*	(ω^6)
11111111	*- { ff }*	(ω^7)
00011010	*- { 1a }*	(ω^8)
00101110	*- { 2e }*	(ω^9)
. . .	. . .	. . .
11110111	*- { f7 }*	(ω^{25})
00000010	*- { 02 }*	(ω^{26})
00000110	*- { 06 }*	(ω^{27})
. . .	. . .	. . .
11000111	*- { b4 }*	(ω^{251})
11000111	*- { c7 }*	(ω^{252})
01010010	*- { 52 }*	(ω^{253})
11110110	*- { f6 }*	(ω^{254})
00000001	*- { 01 }*	(ω^{255})

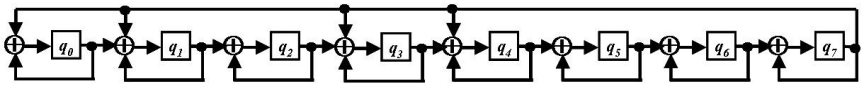


Рис. 8.1. Генератор элементов поля $GF(2^8)$ с примитивным многочленом $\tilde{p}(x) = x^8 + x^4 + x^3 + x + 1$

Например,

$$(x^6 + x^4 + x^2 + x + 1) \bullet (x^7 + x + 1) \bmod p(x) =$$

$$= (x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1) \bmod p(x) =$$

$$= x^7 + x^6 + 1$$

или

$$\omega^{98} \cdot \omega^{80} = \omega^{178},$$

и поэтому

$$\{57\} \bullet \{83\} = \{c1\}.$$

В поле $GF(2^8)$ справедливо $\omega^{255} = 1$, а потому, ненулевые элементы ω^i и ω^j $(i \neq j)$ является мультипликативно обратными тогда и только тогда, если $i + j = 255$ [13, 38].

Например, $\omega^4 \cdot \omega^{251} = 1$, то есть $\{11\} \bullet \{b4\} = 1$ или

$$(x^4 + 1) \cdot (x^7 + x^5 + x^4 + x^2) \bmod p(x) = 1.$$

Здесь i и j – количество тактов, необходимых для перевода генератора элементов поля $GF(2^8)$ в состояние со значением ω^i і ω^j соответственно.

Определение мультипликативно обратных величин в поле $GF(2^8)$ возможно с использованием также приведенного ниже примера.

Пример 8.1. Пусть некоторый байт матрицы состояния на входе функции равен $\{2d\}$. В полиномиальном представлении это $a(x) = x^5 + x^3 + x^2 + 1$. Для нахождения мультипликативно обратного элемента $a^{-1}(x)$ к $a(x)$ в поле $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 + x^4 + x^3 + x + 1$ будем проводить деление многочлена $p(x)$ до тех пор, пока максимальная степень остатка от деления не будет меньше максимальной степени делителя. Если максимальная степень остатка от деления больше 1, деление продолжается, только в качестве делимого берется делитель, а в качестве делителя – остатка от предшествующего деления.

1.

$$\begin{array}{llllll|llll} & x^8 & +x^4 & +x^3 & +x & +1 & x^5 & +x^3 & +x^2 & +1 \\ \oplus & x^8 & +x^6 & +x^5 & +x^3 & & \hline & x^3 & +x & +1 \\ \hline & x^6 & +x^5 & +x^4 & +x & +1 & & & & \\ \oplus & x^6 & +x^4 & +x^3 & +x & & & & & \\ \hline & x^5 & +x^3 & +1 & & & & & & \\ \oplus & x^5 & +x^3 & +x^2 & +1 & & & & & \\ \hline & & & & x^2 & & & & & \end{array}$$

Результат проведенного деления можно представить в виде:

$$p(x) = x^8 + x^4 + x^3 + x + 1 = (x^5 + x^3 + x^2 + 1) \times \\ \times (x^3 + x + 1) + x^2 = a(x) \cdot (x^3 + x + 1) + x^2. \qquad (8.3)$$

В связи с тем, что максимальная степень остатка от деления (x^2) меньше максимальной степени делителя (x^5), процесс текущего де-

ления заканчивается. Так как остаток от деления x^2 больше *1*, то деление продолжается, только в качестве делимого берется делитель ($x^5 + x^3 + x^2 + 1$), а в качестве делителя – остаток от предшествующего деления (x^2).

2.

$$
\begin{array}{llll|l}
 & x^5 & +x^3 \quad +x^2 & +1 & x^2 \\
\oplus & x^5 & & & \hline
 & x^3 & +x^2 \quad +1 & & x^3 \quad +x \quad +1 \\
\oplus & x^3 & & & \\
\hline
 & x^2 & +1 & & \\
\oplus & x^2 & & & \\
\hline
 & 1 & & &
\end{array}
$$

В результате данного деления получился остаток *1*, поэтому процесс деления на этом прекращается.

Результат проведенного деления можно представить в виде:

$$a(x) = x^5 + x^3 + x^2 + 1 = (x^3 + x + 1)\cdot x^2 + 1. \qquad (8.4)$$

Представим выражение (8.4) в таком виде

$$a(x) + (x^3 + x + 1)\cdot x^2 = 1, \qquad (8.5)$$

а выражение (8.3)

$$p(x) + a(x)\cdot(x^3 + x + 1) = x^2. \qquad (8.6)$$

Подставляя (8.5) в (8.6) и производя преобразования получим

$$\begin{aligned} & a(x) + (x^3 + x + 1)\cdot(p(x) + a(x)\cdot(x^3 + x + 1)) = \\ & = a(x) + (x^3 + x + 1)\cdot p(x) + s(x)\cdot(x^3 + x + 1)^2 = \\ & = a(x)\cdot(1 + (x^3 + x + 1)^2) + (x^3 + x + 1)\cdot p(x) = 1. \end{aligned} \qquad (8.7)$$

Вычисляя от левой и правой части (8.7) модульное значение в $GF(2^8)$ с неприводимым полиномом $p(x) = x^8 + x^4 + x^3 + x + 1$ получим:

$$\begin{aligned} & (a(x)\cdot(1 + (x^3 + x + 1)^2) + (x^3 + x + 1)\cdot p(x)) \bmod p(x) = \\ & \quad = a(x)\cdot(1 + (x^3 + x + 1)^2) \bmod p(x) + \\ & \quad + ((x^3 + x + 1)\cdot p(x)) \bmod p(x) = \\ & = a(x)\cdot(1 + (x^3 + x + 1)^2 \bmod p(x) = 1, \end{aligned} \qquad (8.8)$$

так как $((x^3 + x + 1)\cdot p(x)) \bmod p(x) = 0$.

Из выражения (8.8) следует, что

$$a(x)\cdot(1 + (x^3 + x + 1)^2) \bmod p(x) = 1. \qquad (8.9)$$

Сопоставляя (8.9) и (8.1) получим

$$a^{-1}(x) = (1 + (x^3 + x + 1)^2). \qquad (8.10)$$

Производя преобразования в (8.10), окончательно получим

$$a^{-1}(x) = x^6 + x^2. \qquad (8.11)$$

В шестнадцатеричной системе счисление значение байта $a^{-1}(x)$ будет равно *{ 44 }*.

Проверим правильность нахождения мультипликативно обратного элемента $a^{-1}(x)$ в поле $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 + x^4 + x^3 + x + 1$ к элементу $a(x) = x^5 + x^3 + x^2 + 1$. Для этого подставляя значения элементов $a(x)$ и $a^{-1}(x)$ в выражение (8.1) получим:

$$\begin{aligned} a(x)\cdot a^{-1}(x) \bmod p(x) &= (x^5 + x^3 + x^2 + 1)\cdot(x^6 + x^2) = \\ &= (x^{11} + x^9 + x^8 + x^7 + x^6 + x^5 + x^4 + x^2) \bmod p(x) = 1. \end{aligned} \qquad (8.12)$$

Как следует из (8.12) элементы $a(x)$ и $a^{-1}(x)$ являются мультипликативно обратными в поле $GF(2^8)$ с неприводимым полиномом $p(x)$.

8.3. ФОРМАТ ДАННЫХ AES

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.

AES использует пять единиц для представления данных: *биты*, *байты*, *слова*, *блоки* и *массивы состояний*. *Бит* – наименьшая и элементарная единица; другие единицы могут быть выражены в терминах меньших единиц.

В *AES* бит – двоичная цифра со значением *0* или *1*. Для обозначения битов будем использовать строчные буквы.

Рис. 8.2 показывает единицу (формат) данных – байт.

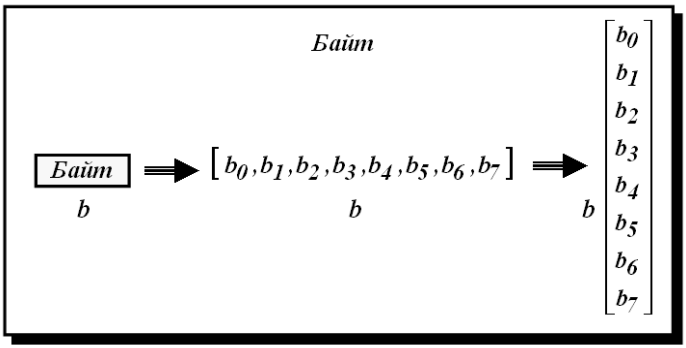


Рис. 8.2. Единица (формат) данных – байт

Байт – группа из восьми битов, которая может быть обработана как единый объект: матрица из одной строки (1×8) восьми битов или столбец матрицы (8×1) из восьми битов. Когда информация байта обрабатываются как матрица строки, то биты вставляются в матрицу слева направо. Когда байт обрабатывается как матрица столбца, биты вставляются в матрицу сверху вниз. Будем использовать строчную букву (b) для обозначения байта.

Рис. 8.3 показывает единицу (формат) данных – слово.

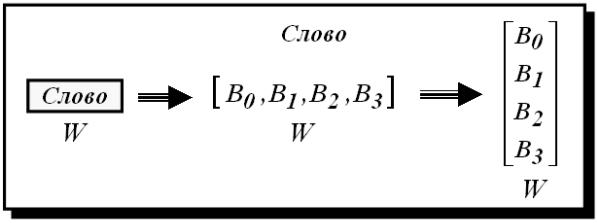


Рис. 8.3. Единица (формат) данных – слово

Слово – группа из *32* битов, которая может быть обработана как единый объект. Это матрица из строки в четыре байта или столбец матрицы из четырех байтов. Когда слово обрабатывается как матрица-строка, байты вставляются слева направо. Когда слово представляется матрицей-столбцом, байты вставляются сверху вниз. Будем использовать прописную букву *W* для обозначения слова.

Рис. 8.4 показывает единицу (формат) данных – блок.

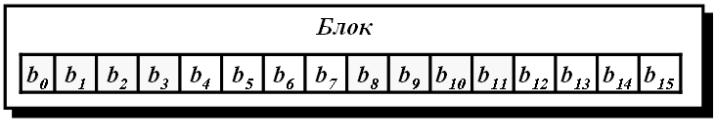


Рис. 8.4. Единица (формат) данных – блок

AES зашифровывает и расшифровывает блоки данных. *Блок* в *AES* – группа из *128* битов. Однако блок может быть представлен как матрица-строка из *16* байтов.

AES использует несколько раундов, каждый раунд состоит из несколько каскадов. Блок данных преобразовывается от одного каскада к другому. В начале и в конце шифра *AES* применяется термин *блок данных*; до и после каждого каскада блок данных называется *матрицей состояния*. Для обозначения этой матрицы будем использовать прописную букву *S*. Хотя матрица состояния на различных каскадах обычно обозначается *S*, иногда будем применять прописную букву *T*, чтобы обозначить временную матрицу состояния.

Рис. 8.5 показывает единицу (формат) данных – матрицу состояния.

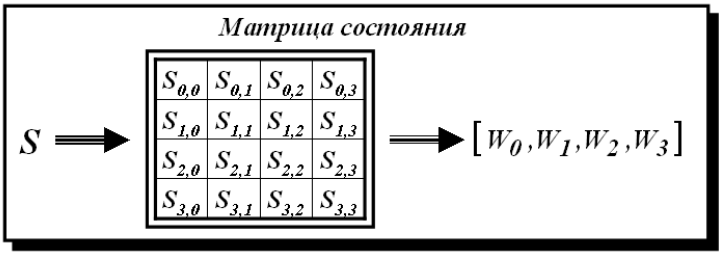


Рис. 8.5. Единица (формат) данных – матрица состояния

Матрицы состояний, подобно блокам, состоят из *16* байтов, но обычно обрабатываются как матрицы 4×4 байтов. В этом случае каждый элемент матрицы состояний обозначается как $S_{r,c}$, где r (от *0* до *3*) определяет строку и c (от *0* до *3*) определяет столбец. Иногда матрица состояния обрабатывается как матрица-строка слов (1×4). Это имеет смысл, если представлять слово как матрицу-столбец. В начале шифра байты в блоке данных вставляются в матрицу состояния столбец за столбцом, в каждом столбце – сверху вниз. В конце шифра байты в матрице состояния извлекаются, как это показано на рис. 8.6.

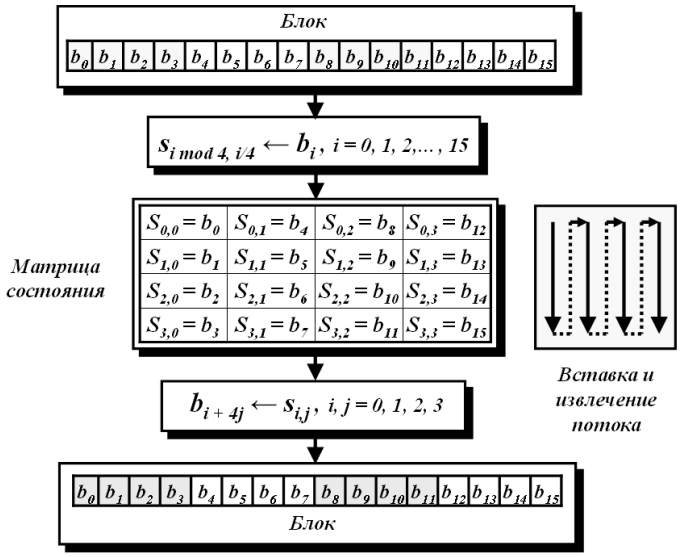


Рис. 8.6. Превращение блока в матрицу состояния и матрицу состояния в блок

Пример 8.2. Рассмотрим, как можно изобразить блок с *16* символами в виде матрицы *4×4*. Предположим, что текстовый блок – "*AES uses a matrix*". Добавим два фиктивных символа в конце и получим "*AESUSESAMATRIXZZ*". Теперь заменим каждый символ целым числом между *00* и *25* (см. рис. 2.1). Представим каждый байт как целое число с двумя шестнадцатеричными цифрами. Например, символ *S* сначала поменяем на *18*, а затем запишем в шестнадцатеричном изображении как *12*. Матрица состояния тогда заполняется столбец за столбцом, как это показано на рис. 8.7.

Текст	*A*	*E*	*S*	*U*	*S*	*E*	*S*	*A*	*M*	*A*	*T*	*R*	*I*	*X*	*Z*	*Z*
Десятичное значение	*00*	*04*	*18*	*20*	*18*	*04*	*18*	*00*	*12*	*00*	*19*	*17*	*08*	*23*	*25*	*25*
Шестнадцатеричное значение	*00*	*04*	*12*	*14*	*12*	*04*	*12*	*00*	*0c*	*00*	*13*	*11*	*08*	*17*	*19*	*19*

Матрица состояния

00	*12*	*0c*	*08*
04	*04*	*00*	*23*
12	*12*	*13*	*19*
14	*00*	*11*	*19*

Рис. 8.7. Переход исходного текста в матрицу состояния

8.4. СТРУКТУРА АЛГОРИТМА И РАУНДОВ AES

8.4.1 Структура алгоритма

AES – шифр не-*Фейстеля*, который зашифровывает и расшифровывает блок данных *128* битов, используя *10, 12* или *14* раундов. Размер ключа может быть *128, 192* или *256* битов и зависит от номера раунда. Рис. 8.8 показывает общую схему: алгоритм зашифрования (называемого шифром); алгоритм расшифрования (называемый обратным шифром), для которого применяются те же ключи, но в обратном порядке.

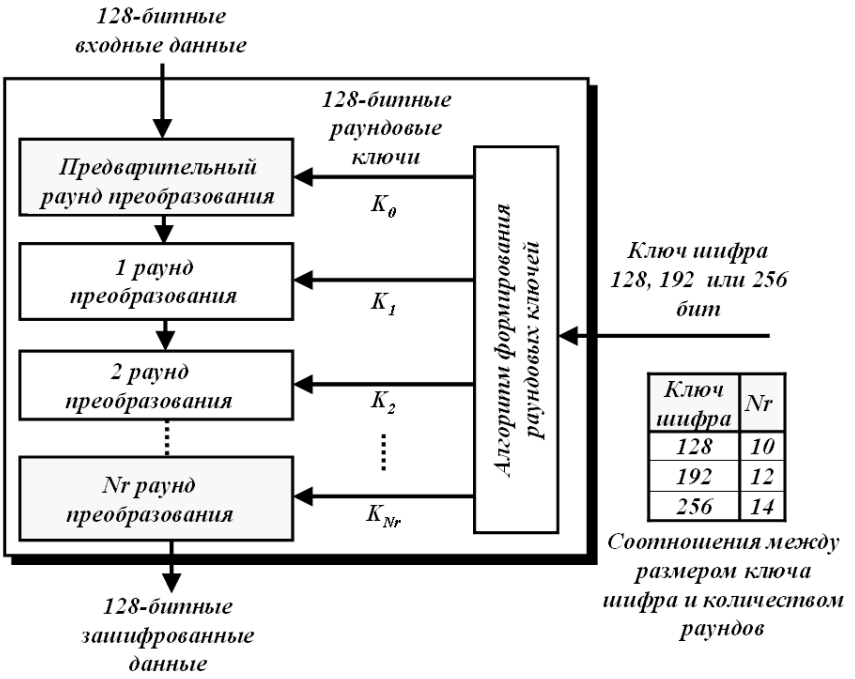


Рис. 8.8. Общее построение схемы шифрования стандарта AES

На рис. 8.8 *Nr* определяет номер раундов. Рисунок также показывает отношение между количеством раундов и размером ключа. Это означает, что существует три различных версии *AES*; они обозначаются как *AES-128, AES-192* и *AES-256*. Однако ключи раунда, которые созданы алгоритмом расширения ключей всегда *128* бит,

имеют тот же самый размер, что и блоки исходных или зашифрованных данных.

Число ключей раунда, сгенерированных алгоритмом расширения ключей, всегда на один больше, чем число раундов. Другими словами, имеем количество раундовых ключей $Nr + 1$. Обозначим раундовые ключом как: $K_0, K_1, K_2, ..., K_{Nr}$.

8.4.2 Структура раундов алгоритма

Рис. 8.9 показывает структуру каждого раунда на стороне зашифрования.

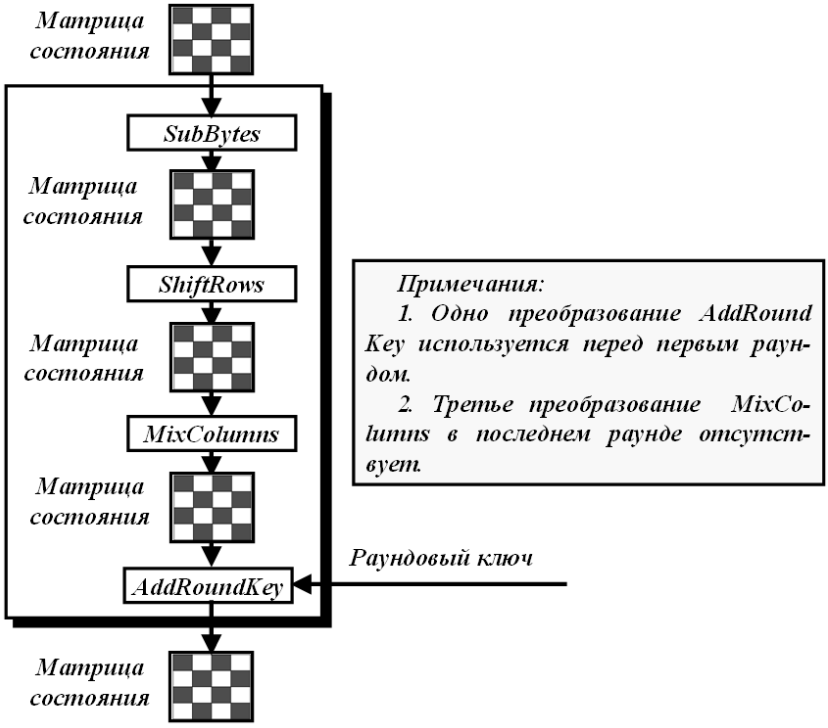


Рис. 8.9. Структура каждого раунда стандарта *AES* на стороне зашифрования

Каждый раунд, кроме последнего, использует четыре преобразования, которые являются обратимыми. Последний раунд имеет только три преобразования.

Как показывает рис. 8.9, каждое преобразование принимает матрицу состояния и создает другую матрицу состояния, которая применяется для следующего преобразования или следующего раунда. Секция, предваряющая раунды, использует только одно преобразование (*AddRoundKey*); последний раунд использует только три преобразования (*MixColumns* – преобразование отсутствует).

8.4.3. Количество раундов алгоритма

Для выполнения преобразований в блоке используется раундовый ключ *W*, получаемый из секретного ключа *K*. Раундовый ключ в стандарте *AES* состоит из блоков (по *128* бит). Также в стандарте *AES* (*Rijndael*) определено соответствие между размером ключа, размером блока данных и количеством раундов шифрования, как показано в в табл. 8.1 и 8.2.

Таблица 8.1

Количество раундов шифрования *Nr* как функція размера ключа *Nk* и размера блока данных *Nb*

Nr	*Nb = 4*	*Nb = 6*	*Nb = 8*
Nk = 4	*10*	*12*	*14*
Nk = 6	*12*	*12*	*14*
Nk = 8	*14*	*14*	*14*

Таблица 8.2

Соответствие между размером ключа *Nk*, размером блока данных *Nb* и количеством раундов шифрования *Nr* в стандарте *AES*

Стандарт	*Nk*	*Nb*	*Nr*
AES-128	*4*	*4*	*10*
AES-192	*6*	*4*	*12*
AES-256	*8*	*4*	*14*

Авторы шифра определили количество раундов, учитывая максимальное количество раундов, для которых еще возможна атака, более эффективная, чем полный перебор по всему ключевому пространству (так называемая сокращенная атака), и прибавили запас в виде дополнительных раундов.

Для упрощенной версии шифра из *6* раундов со *128*-битовыми блоками данных и ключом не было выявлено ни одной эффектив-

ной сокращенной атаки. Добавление еще *4* раундов с учетом следующих соображений является более чем достаточным запасом прочности: два раунда шифра обеспечивают полное рассеивание в том смысле, что каждый бит блока *State* зависит от всех бит этого же блока "*двухраундовой давности*", или, иначе говоря, изменение одного бита блока *State* с большой вероятностью отразится на половине бит этого блока через *2* раунда. Таким образом, дополнительные *4* раунда могут рассматриваться как добавление шага полного рассеивания в начале и в конце процедуры шифрования.

Высокая степень рассеивания шифра обусловлена целостным характером алгоритма, преобразующего сразу все биты входных данных. Для сетей *Фейстеля* (алгоритмы *DES* и *ГОСТ 28147-89*) раундовое преобразование за шаг изменяет только половину бит входных данных, и полное рассеивание на практике достигается не менее, чем за *4* и более раундов.

При линейном и дифференциальном криптографическом анализе, при проведении атаки методом сокращенных дифференциалов используют зависимости, которые можно проследить сквозь n раундов для того, чтобы атаковать $(n+1)$-й или $(n+2)$-й раунд. Это же верно и для *Square*-атак, использующих зависимости сквозь *4* раунда для атаки на шестой раунд. Для последних дополнительные *4* раунда означают удваивание количества раундов, сквозь которые необходимо отслеживать зависимости.

Для версий шифра с более длинным ключом количество раундов увеличивается на единицу для каждых дополнительных *32* разрядов ключа шифрования, исходя из следующих соображений: одна из главных целей – невозможность сокращенных атак, поскольку с ростом длины ключа трудоемкость простого перебора возрастает, то и сокращенная атака может позволить себе быть более трудоемкой.

Атаки по известному (или частично известному) ключу, атаки с использованием "*эквивалентных ключей*" основаны соответственно на наличии информации о битах начального ключа шифрования и возможности применения нескольких различных ключей шифрования. Если ключ удлиняется, то и возможностей у криптографического аналитика становится больше.

Поскольку никаких эффективных атак по известному ключу или с использованием "*эквивалентных ключей*" для шифра с шестью раундами найдено не было, то остальные раунды могут рассматриваться как дополнительный запас стойкости.

Для других версий шифра *Rijndael* (не включенных в стандарт) с блоками входных данных, размер которых больше *128* разрядов, количество раундов также увеличивается на единицу с каждым дополнительным *32*-разрядным словом, исходя из следующих соображений: для блоков размером более *128* разрядов полное рассеивание наступает не ранее, чем после проведения третьего раунда преобразований, т.е. относительное рассеивание раунда уменьшается с ростом размера блока входных данных.

С ростом размера блока входных данных также увеличивается количество возможных комбинаций зависимостей входных/выходных данных. А это, в свою очередь, иногда позволяет продлить эффективность атаки еще на один раунд.

Авторы шифра утверждают тем не менее, что угроза распространения эффективности атаки еще на один раунд даже для *256*-разрядного блока маловероятна. Поэтому все раунды, номер которых больше шести, можно рассматривать как дополнительный запас стойкости шифра.

В основу разработки *Rijndael* были положены три критерия:

- стойкость по отношению ко всем известным атакам;
- скорость и компактность кода;
- простота дизайна.

В отличие от предыдущих рассмотренных симметричных блочных шифров, *Rijndael* не использует какой-либо аналог структуры *Фейстеля*. Каждый раунд состоит из трех различных обратимых преобразований, называемых *слоями*:

- линейный смешивающий слой гарантирует высокую степень взаимопроникновения символов блока для маскировки статистических связей;
- нелинейный слой реализован с помощью *S*-блоков, имеющих оптимальную нелинейность, и предотвращает возможность использования дифференциального, линейного и других современных методов криптографического анализа;
- слой сложения с ключом выполняет непосредственно шифрование.

Шифр начинается и заканчивается сложением с ключом. Это позволяет закрыть вход первого раунда при атаке по известному тексту и сделать криптографически значимым результат последнего раунда.

8.5. РАУНДОВЫЕ ПРЕОБРАЗОВАНИЯ АЛГОРИТМА AES

Для шифрования данных алгоритм *AES* (*Rijndael*) использует так называемую *круговую функцию*, которая состоит из четырех различных байт-ориентированных (раундовых) преобразований:

- замены байтов массива состояния с использованием таблицы подстановки – *SubBytes(State)*, т.е. побайтовой подстановки в *S*-блоках с фиксированной таблицей замен размерностью 8×256;
- сдвига строк в массиве состояния на различное количество позиций – *ShiftRows(State)*;
- перемешивание столбцов матрицы состояния – *MixColumns(State)* (умножения столбцов состояния, рассматриваемых как многочлены над $GF(2^4)$, на многочлен третьей степени $g(x)$ по модулю $x^4 + 1$);
- добавление раундового ключа к матрице состояния – *Add-RoundKey(State, RoundKey)*, т.е. поразрядного *xor* матрицы состояния с текущим фрагментом развернутого ключа.

Для расшифрования данных алгоритм *AES* (*Rijndael*) использует также круговую функцию, которая состоит из четырех байт-ориентированных (раундовых) преобразований, которые являются обратными (инверсными) к преобразованиям при зашифровании соответственно: *InvSubBytes*(*State*); *InvShiftRows*(*State*) и *InvMixCo-lumns*(*State*). Инверсное преобразование *InvAddRoundKey*(*State, RoundKey*) аналогичное самому преобразованию *AddRound Key*(*State, RoundKey*), потому что использует операцию поразрядного *xor*.

8.5.1. Замена байтов матрицы состояния – *SubBytes()* и *InvSubBytes ()*

Замена байтов матрицы состояния – SubBytes()

Процедура *SubBytes()* (замена байтов) используется на стороне зашифрования, реализует слой нелинейного преобразования и представляет собой нелинейную замену байтов, выполняемую независимо с каждым байтом состояния. Таблицы замены *S*-блока являются обратимыми и построены из композиции следующих двух преобразований входного байта:

- получение мультипликативно обратного элемента относительно умножения в поле $GF(2^8)$, т.е. решение уравнения:

$$s(x) \cdot s^{-1}(x) \bmod p(x) = 1 \qquad (8.13)$$

относительно элемента $s^{-1}(x)$ (нулевой элемент *{ 00 }* переходит сам в себя);

- применение аффинного преобразования над $GF(2^8)$ определенного следующим образом [13, 29, 38]:

$$d(x) := (a(x) \cdot s^{-1}(x) + b(x)) \bmod (x^8 + 1), \qquad (8.14)$$

где $a(x)$ и $b(x)$ – полиномы, которые имеют фиксированные значения для алгоритма *AES* (*Rijndael*) и равны:

$$a(x) = x^7 + x^6 + x^5 + x^4 + 1; \quad b(x) = x^6 + x^5 + x + 1.$$

Блоки замен в шифре *Rijndael* играют важную роль. Согласно основополагающим принципам, сформулированным еще *К. Шенноном*, преобразования данных, используемые в шифре, должны придавать последнему два основных свойства – рассеивание и перемешивание. *Рассеивание* предполагает распространение влияния каждого бита открытых данных, а также каждого бита ключа на значительное количество битов зашифрованных данных. *Перемешивание* же приводит к потере в процессе шифрования всяческих зависимостей между битами открытых данных. То есть на выходе получаются данные, как если бы их выбрали совершенно случайным образом, а не как значение функций преобразования алгоритма шифрования. Именно эти два свойства обеспечивают защиту от двух возможных угроз: *подделки сообщения* и его *раскрытия*.

За обеспечение этих двух свойств отвечают функции *MixColumns()* и *SubBytes()*. Рассеивание в шифре *Rijndael*, обеспечивается в основном функцией *MixColumns()*, перемешивание – в основном функцией *SubBytes()*. Таким образом, критерии выбора и разработки таблицы замен *S*-блоков обусловлены, с одной стороны, учетом возможностей дифференциального и линейного криптографического анализа (будут рассмотрены далее), а с другой – учетом возможных алгебраических манипуляций, например, атаки методом интерполяции (будет рассмотрена ниже). Таким образом, основными критериями выбора таблицы замен *S*-блоков стали:

- обратимость;

- минимизация наидлиннейшей возможной нетривиальной корреляции между линейными комбинациями бит входных и выходных данных; иначе говоря, для всех случайно выбранных входных дан-

ных количество раундов, в которых еще можно проследить зависимости между входными и выходными данными, будет минимально; именно прослеживание таких зависимостей является основой линейного криптографического анализа;

- минимизация наибольшего нетривиального значения *xor* таблицы; характеризует устойчивость к дифференциальному криптографическому анализу;

- сложность алгебраического представления в поле $GF(2^8)$; важно для устойчивости к алгебраическим атакам;

- простота описания.

В [7, 29, 38] приведено несколько методов конструирования таблиц *S*-блоков, удовлетворяющих первым трем критериям. Для обратимой байтовой таблицы замен максимальная корреляция входных/выходных данных может быть минимизирована до 2^{-3}, а максимальная величина *xor* таблицы – минимизирована до *4* (что соответствует коэффициенту дифференциального проникновения 2^{-6}).

Для формирования таблицы замен было выбрано отображение $s \rightarrow s^{-1}$ (мультипликативная инверсия) в поле $GF(2^8)$. Однако очевидно, что выбранное отображение имеет слишком простое алгебраическое представление. Это делает возможным использование алгебраических манипуляций в таких атаках на шифр, как, например, *атака методом интерполяции* [38]. Поэтому результат преобразования подвергается дополнительному (вполне обратимому) изменению.

Это изменение не умаляет свойств *S*-блока в отношении первых трех критериев, но позволяет удовлетворить требование соответствия таблицы замен четвертому критерию выбора. Было выбрано такое изменение, которое само по себе очень простое в описании, но при этом, в сочетании с нахождением мультипликативного обратного элемента в поле $GF(2^8)$, имеет сложное алгебраическое представление. Оно может быть представлено как перемножение многочленов по модулю $x^8 + 1$ с последующим сложением:

$$\begin{aligned} s(x) &= (a(x) \cdot d(x) + b(x)) \bmod (x^8 + 1) = \\ &= ((x^7 + x^6 + x^5 + x^4 + 1) \cdot d(x) \cdot + \\ &+ (x^6 + x^5 + x + 1)) \bmod (x^8 + 1). \end{aligned} \qquad (8.15)$$

Здесь $d(x)$ – преобразуемый байт в виде многочлена, $s(x)$ – результирующий байт. Модуль $x^8 + 1$ был выбран как самый простой из возможных. Сомножитель $x^7 + x^6 + x^5 + x^4 + 1$ ($a(x)$ в выражении (8.15)) был выбран из набора многочленов, взаимно простых с модулем $x^8 + 1$, как имеющий наиболее простое представление. А многочлен $x^6 + x^5 + x + 1$ ($b(x)$ в выражении (8.15)) был выбран таким образом, чтобы полученная в результате таблица замен не имела точек симметрии ($S(d(x)) = d(x)$) и точек обратной симметрии ($S(d(x)) = d^{-1}(x)$).

Другими словами суть преобразования (8.15) может быть описана уравнениями:

$$s_i = d_i \oplus d_{(i+7) \bmod 8} \oplus d_{(i+6) \bmod 8} \oplus d_{(i+5) \bmod 8} \oplus \oplus d_{(i+4) \bmod 8} \oplus b_i, \tag{8.16}$$

где d_i и s_i - d_i и s_i – соответственно исходное и преобразованное значение i-го бита, $i = 0,1,\ldots,7$; $b_0 = b_1 = b_5 = b_6 = 1$, $b_2 = b_3 = b_4 = b_7 = 0$.

Применение преобразования (8.16) можно описать в матричном виде следующим образом:

$$S(x) = A(x) \cdot D(x) \oplus B(x), \tag{8.17}$$

где $S(x)$, $D(x)$ и $B(x)$ – вектор-столбцы размерностью 8×1; $A(x)$ – матрица размерностью 8×8.

Выражение (8.17) можно представить в развернутом виде:

$$s(x) = \begin{vmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{vmatrix} = \begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{vmatrix} \cdot \begin{vmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{vmatrix} \oplus \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{vmatrix}. \tag{8.18}$$

Как видно из выражения (8.18) матрица $A(x)$ равна:

$$A(x)=\begin{vmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{vmatrix}. \qquad (8.19)$$

Матрицы, подобные матрице $A(x)$, называются *матрицей Тёплица* (теплицевыми матрицами) или диагонально-постоянными матрицами. В линейной алгебре, матрица *Тёплица*, названная в честь немецкого математика *Отто Тёплица* – это матрица, в которой на всех диагоналях (включая и главную), параллельных главной, стоят равные элементы.

Примечание. Существуют также и другие *S*-блоки, удовлетворяющие вышеперечисленным критериям. Так что в случае подозрения на существование в данной таблице "*черного хода* она может быть легко заменена на другую. Более того, структура шифра и выбранное количество раундов позволяют применить такую таблицу замен, которая не удовлетворяет критериям *2* и *3*. Даже "*средняя*" в этом отношении таблица замен тем не менее будет обеспечивать стойкость к дифференциальному и линейному криптографического анализу.

Пример 8.3. Пусть некоторый байт состояния на входе функции *SubBytes()* равен *{2a}*. В полиномиальном представлении это $s(t) = x^5 + x^3 + x$ (ω^{166}). Необходимо определить для него мультипликативно обратное значение в поле $GF(2^8)$.

Решение. В поле $GF(2^8)$ для нахождения мультипликативно обратного элемента должно выполняться равенство относительно примитивных элементов: $\omega^i \cdot \omega^j = 1$ ($i + j = 255$). При $i = 166$ (ω^{166}) j будет равен *89* (ω^{89}). В таком случае примитивному элементу ω^{89} [9] будет соответствовать многочлен $s^{-1}(t) = x^7 + x^4 + x^3$, который является мультипликативно обратным к многочлену $s(t) = x^5 + x^3 + x$. Для проверки этого утверждения умножим многочлены для $s(t)$ и $s^{-1}(t)$ в поле $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 + x^4 + x^3 + x + 1$. Если произведение будет равно единице, то $s(t)$ и $s^{-1}(t)$ действительно есть мультипликативно обратными.

$$s(x)\cdot s^{-1}(x) \bmod p(x) = (x^7 + x^4 + x^3)\cdot(x^5 + x^3 + x) =$$
$$= (x^{12} + x^{10} + x^9 + x^7 + x^6 + x^5 + x^4) \bmod p(x) = 1. \qquad (8.20)$$

Итак, мультипликативно обратным значением байта состояния *{ 2a }* на входе функции *SubBytes()* в поле $GF(2^8)$ будет байт *{ 98 }* (эквивалент многочлена $s^{-1}(x) = x^7 + x^4 + x^3$).

Пример 8.4. Пусть некоторый байт состояния полученный в результате вычисления мультипликативно обратного значения равен *{ 98 }*. В полиномиальном представлении это $d(x) = s^{-1}(x) = x^7 + x^4 + x^3$ (см. пример 8.1). Необходимо определить для него аффинное преобразование в поле $GF(2^8)$.

Решение. Подставляя значение полинома $d(x)$ в выражение (8.16) или (8.18), производя преобразования и вычисляя получим

$$S(x) = \{ s_7, s_6, s_5, s_4, s_3, s_2, s_1, s_0 \} = \{ 1, 1, 1, 0, 0, 1, 0, 1 \}. \qquad (8.21)$$

Делая представления (8.21), получим $s(x)$:

- в двоичной системе – $s(x) =\{ 11100101 \}$;
- в шестнадцатеричной системе – $s(x) =\{ e5 \}$.

Если рассчитать мультипликативные инверсии и аффинные преобразования для всех возможных состояний (байтов) *{00}* до *{ff}*, то получается набор данных, которые можно свести в табл. 8.3.

Преобразование *SubBytes(state)* сводится к выполнению для каждого байта *s* в *state* операции $s \leftarrow S(s)$. Чтобы применить подстановку байта, необходимо интерпретировать байт как две шестнадцатеричные цифры. Левая цифра определяет строку, а права – колонку в табл. 8.3. На пересечении строки и колонки, обозначенных этими шестнадцатеричными цифрами, находится новый байт. Рис. 8.10 иллюстрирует эту идею.

Например, результат преобразования байта *{ 53 }* находится на пересечении *5*-й строки и *3*-го столбца и равен *{ ed }*.

В преобразовании *SubByte()* матрица состояния обрабатывается как матрица байтов *4x4*. В один момент проводится преобразование одного байта. Содержание каждого байта изменяется, но расположения байтов в матрице остается таким же. В процессе преобразования каждый байт превращается независимо от других – это шестнадцать преобразований байт в байт.

Таблица 8.3

Таблица преобразования *SubBytes()*

	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*	*fo*	*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*	*ao*	*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	*do*	*ef*	*aa*	*fb*	*43*	*4d*	*33*	*85*	*45*	*f9*	*02*	*f7*	*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*	*cb*	*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*	*if*	*4b*	*bd*	*8b*	*8a*
d	*70*	*3e*	*b5*	*66*	*48*	*03*	*f6*	*oe*	*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*

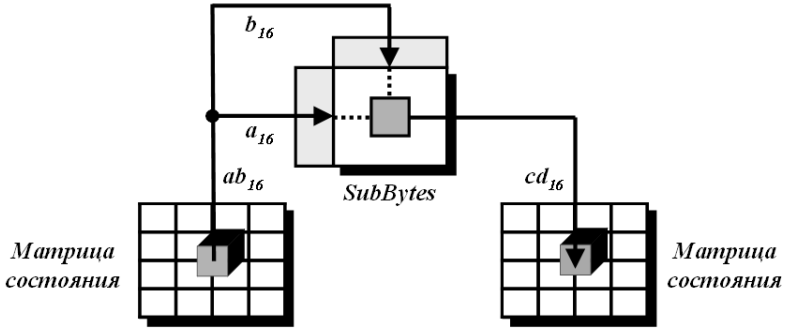


Рис. 8.10. Пояснение идеи табличной замены
в преобразовании *SubBytes()*

Пример 8.5. Пусть на вход функции *SubBytes()* алгоритма *AES* (*Nb = 4*) поступает матрица состояния равная:

$$State = 193de3bea0f4e22b9ac68d2ae9f84808_{16}.$$

Если произвести табличную замены каждого байта (см. табл. 8.3), то получится (рис. 8.11):

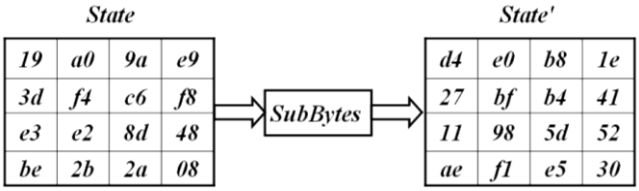


Рис. 8.11. Результат выполнения табличной замены каждого байта матрицы состояния

Преобразования *SubBytes()* обеспечивает эффект перемешивания. Например, два байта, $5a_{16}$ и $5b_{16}$, которые отличаются только одним битом (крайний правый бит), превращенные в be_{16} и 39_{16} соответственно, которые отличаются уже четырьмя битами.

Замена байтов матрицы состояния – InvSubBytes()

Обратные (инверсные) преобразования, использованные в алгоритме, определяются естественным образом.

Процедура *InvSubBytes(State)* (инверсная замена байтов) реализует слой нелинейного преобразования и представляет собой мультипликативно инверсную нелинейную замену байтов, выполняемую независимо с каждым байтом состояния. Таблицы замены *S*-блока процедуры *InvSubBytes()* являются обратимыми и построены из композиции следующих двух преобразований входного байта:

1. Применение инверсии к аффинному преобразованию (8.14) над $GF(2^8)$ определенного следующим образом [7, 29, 38]:

$$d^{-1}(x)\cdot = (s(x) + b(x))\cdot a^{-1}(x) \bmod (x^8 + 1), \qquad (8.22)$$

где $a^{-1}(x) = (x^7 + x^6 + x^5 + x^4 + 1)^{-1} \bmod (x^8 + 1) = x^7 + x^5 + x^2$.

Другими словами суть преобразования (8.22) может быть описана уравнениями:

$$d_i = c_{(i+7) \bmod 8} \oplus c_{(i+5) \bmod 8} \oplus c_{(i+2) \bmod 8}, \qquad (8.23)$$

где $c_i = s_i \oplus b_i$; $b_0 = b_1 = b_5 = b_6 = 1$, $b_2 = b_3 = b_4 = b_7 = 0$, s_i и d_i – соответственно исходное и преобразованное значение i-го бита, $i = 0,1,...,7$.

Применение преобразования (8.22) можно описать в матричном виде следующим образом:

$$d^{-1}(x)=\begin{vmatrix} d_0 \\ d_1 \\ d_2 \\ d_3 \\ d_4 \\ d_5 \\ d_6 \\ d_7 \end{vmatrix} = \begin{vmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{vmatrix} \cdot \left(\begin{vmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{vmatrix} \oplus \begin{vmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{vmatrix} \right). \qquad (8.24)$$

Применение преобразования (8.24) можно описать в матричном виде следующим образом:

$$D^{-1}(x) = A^{-1}(x) \cdot (S(x) \oplus B(x)),$$

где D^{-1}, $S(x)$ и $B(x)$ – вектор-столбцы размерностью 8×1; $A^{-1}(x)$ – матрица размерностью 8×8.

Как видно из выражения (8.24) матрица $A^{-1}(x)$ равна:

$$A^{-1}(x)=\begin{vmatrix} 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \end{vmatrix}.$$

2. Получение мультипликативно обратного элемента относительно умножения в поле $GF(2^8)$, т.е. решение уравнения

$$d(x) \cdot d^{-1}(x) \bmod p(x) = 1 \qquad (8.25)$$

относительно элемента $d^{-1}(x)$ (нулевой элемент *{ 00 }* переходит сам в себя).

Пример 8.6. Пусть некоторый байт состояния на входе функции *InvSubBytes()* равен *{ e5 }*. В полиномиальном представлении это $s(t) = x^7 + x^6 + x^5 + x^2 + 1$. Необходимо определить для него инверсное аффинное преобразование в поле $GF(2^8)$.

Решения. Подставляя значения многочлена $s(x)$ в выражение (8.23) или (8.24), делая преобразования и вычисляя, получим

$$d^{-1}(x) = \{ d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0 \} = \{ 1, 0, 0, 1, 1, 0, 0, 0 \}. \qquad (8.26)$$

Делая представления (8.26), получим $d^{-1}(x)$:

- в двоичной системе – $d^{-1}(x) = \{ 10011000 \}$;
- в шестнадцатеричной системе – $d^{-1}(x) = \{ 98 \}$.

Пример 8.7. Пусть некоторый байт состояния полученный вследствие вычисления инверсного аффинного преобразования функции *InvSubBytes()* равен *{ 98 }*. В полиномиальном представлении это $d^{-1}(x) = x^7 + x^4 + x^3$ (см. пример 8.6). Необходимо определить для него мультипликативно обратное значение в поле $GF(2^8)$.

Решение. В поле $GF(2^8)$ для нахождения мультипликативно обратного элемента должно выполняться равенство относительно примитивных элементов: $\omega^i \cdot \omega^j = 1$ ($i + j = 255$). При $i = 89$ (ω^{89}) j будет равна *166* (ω^{166}). В таком случае примитивному элементу ω^{166} [9] будет соответствовать многочлен $s(x) = x^5 + x^3 + x$, который является мультипликативно обратным многочлену $d^{-1}(x) = x^7 + x^4 + x^3$.

Итак, мультипликативно обратным значением байта состояния *{ 98 }* на входе функции *InvSubBytes()* в поле $GF(2^8)$ будет байт *{ 2a }* (эквивалент многочлена $s(x) = x^5 + x^3 + x$).

Если рассчитать инверсии аффинных и мультипликативно обратных преобразований для всех возможных состояний (байтов) от *{ 00 }* до *{ ff }*, то получится набор данных, которые можно свести в табл. 8.4. Тогда преобразования *InvSubBytes(state)* сводится к выполнению для каждого байта *s* в *state* операции $s \leftarrow S(s)$.

Табл. 8.3 и 8.4 взаимно обратные. Например, если выполнить преобразование над байтом *{ 2d } SubBytes()* (см. табл. 8.3), то получим *{ d8 }*. Если теперь применить к байту *{ d8 }* преобразования *InvSubBytes()* (табл. 8.4), то получим *{ 2d }*.

Пример 8.8. Рис. 8.12 показывает, как матрица состояния преобразуется с использованием *SubBytes()* и *InvSubBytes()*.

Рисунок также показывает, что *InvSubBytes()* однозначно воспроизводит оригинал. Заметим, что если два байта имеют одинаковое значение, то они превратятся одинаково. Например, два байта $4e_{16}$ и $4e_{16}$ в левой матрице состояния превращаются в $b6_{16}$ и $b6_{16}$ в правой матрице состояния и наоборот. Причина в том, что каждый байт использует ту же таблицу преобразований.

Таблица 8.4

Таблица преобразования *InvSubBytes()*

	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*	*de*	*6e*
a	*47*	*e1*	*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*	*d*	*a8*	*33*	*88*	*07*	*c7*	*31*	*b1*	*12*	*10*	*59*	*27*	*80*	*ec*	*5f*
d	*60*	*51*	*7e*	*a9*	*19*	*b5*	*4a*	*0d*	*2d*	*e5*	*7a*	*9f*	*93*	*c9*	*9c*	*ef*
e	*a0*	*e0*	*3b*	*4d*	*ae*	*2a*	*f5*	*b0*	*cb*	*eb*	*bb*	*3c*	*83*	*53*	*99*	*61*
f	*17*	*2b*	*04*	*7e*	*ba*	*77*	*d6*	*26*	*e1*	*69*	*14*	*63*	*55*	*21*	*0c*	*7d*

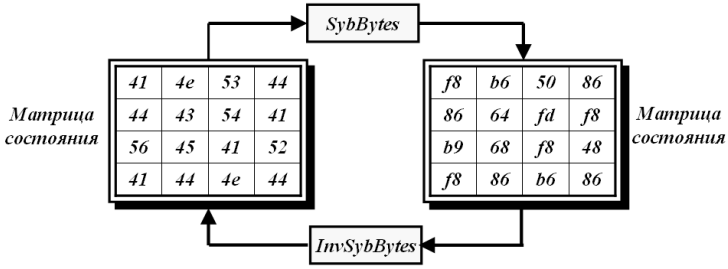


Рис. 8.12. Преобразования *SubBytes()* и *InvSubBytes()* для примером 8.8

Хотя можно использовать таблицы 8.3 и 8.4 для перестановки каждого байта, *AES* дает определение алгебраическим преобразованиям на основе поля $GF(2^8)$ с помощью многочленов, как это показано на рис. 8.13.

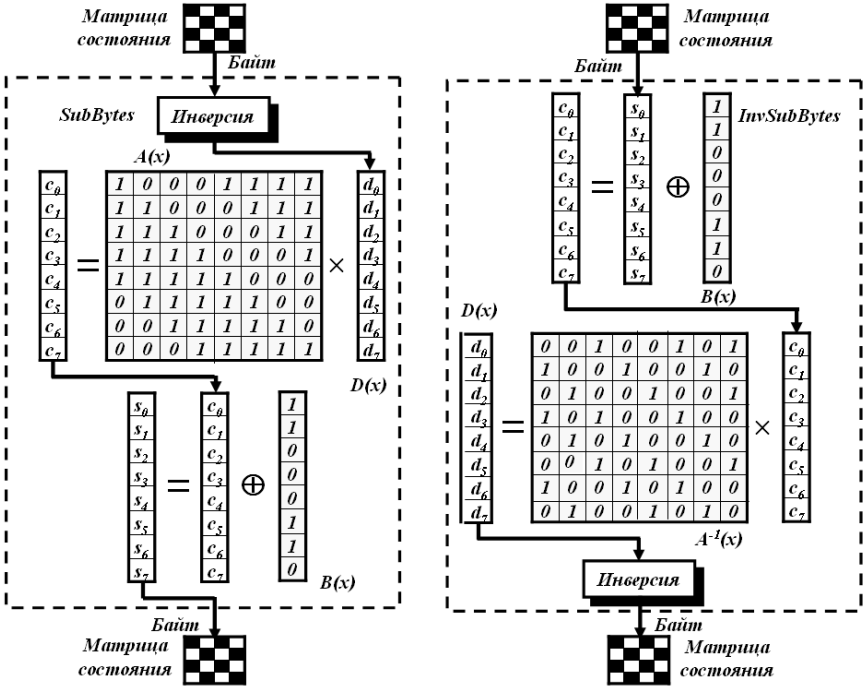


Рис. 8.13. Преобразования *SubByte()* и *InvSubByte()*

В процедуре *SubByte()* байт (двоичной строки на *8* битов) находится в поле $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 + x^4 + x^3 + x + 1$. Заметим, что байт 00_{16} сам является собственным обращением. Обратный байт затем интерпретируется как матрица-столбец с самым младшим битом вверху и старшим битом внизу. Эта матрица-столбец умножается на постоянную квадратную матрицу *A(x)* и результат, который является матрицей-столбцом, суммируется с постоянной матрицей-столбцом *B(x)*, что дает новый байт. Заметим, что умножение или сумма битов происходит в $GF(2^8)$. *InvSubByte()* делает те же действия, но в обратном порядке.

В процессе зашифрования умножение является первой операцией, сложение – второй. В ходе расшифрования вычитание (сложение с инверсией) является первым, а деление (умножение с инверсией) – вторым.

Хотя умножение и сложение матриц в процедуре *SubBytes()* и *InvSubBytes*() – преобразования аффинного типа и линейное, замена байта его мультипликативно обратным значением в $GF(2^8)$ – нелинейная. Этот шаг делает все преобразования нелинейными.

8.5.2. Сдвиг строк матрицы состояния - *ShiftRows()* и *InvShiftRows()*

Процедура *ShiftRows(State)* реализует слой линейного преобразования. Выбор из возможных комбинаций сдвигов в строках преобразуемого блока был сделан на основании следующих критериев:

- все четыре сдвига должны быть различны для каждой строки и $C_0 = 0$;
- стойкость к атакам, использующим сокращенные дифференциалы (см. ниже, а также [20]);
- стойкость к атаке *Square* (будет описана ниже, также см. [38]);
- простота.

Для некоторых комбинаций значений сдвигов атаки с использованием сокращенных дифференциалов становятся эффективны для более чем одного раунда. Для других комбинаций более эффективны атаки типа *Square*. Из набора комбинаций сдвигов, наилучшим образом удовлетворяющих пунктам *2* и *3*, была выбрана простейшая.

Последние три строки состояния циклически сдвигаются влево на различное число байтов. Строка *1* сдвигается на C_1 байт, строка *2* – на C_2 байт, и строка *3* – на C_3 байт. Значения сдвигов C_1, C_2 и C_3 в *Rijndael* зависят от длины блока *Nb*. Их величины приведены в табл. 8.5

Таблица 8.5

Величины сдвигов для блоков данных разной длины

Nb	C_1	C_2	C_3
4	*1*	*2*	*3*
6	*1*	*2*	*3*
8	*1*	*3*	*4*

В стандарте *AES*, где определен единственный размер блока ($N_b = 4$), равный *128* битам, $C_1 = 1$, $C_2 = 2$ и $C_3 = 3$.

Операция сдвига последних трех строк матрицы состояния обозначена как *ShiftRows(State)*. Рис. 8.14 показывает влияние преобразования на строки матрицы состояния.

Заметим, что преобразование *ShiftRows()* работает одновременно только с одной строкой.

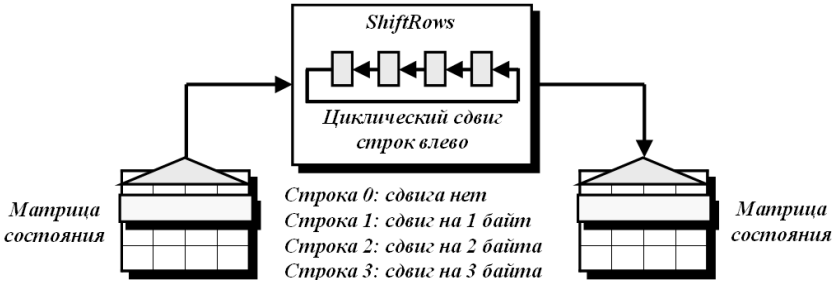


Рис. 8.14. Действие преобразования *ShiftRows()* на строки матрицы состояния

Преобразования *InvShiftRows(State)* обратное преобразование *ShiftRows(State)* и действует на каждую строку r_i в *State* по такому правилу (рис. 8.15): первая строка *State* не сдвигается, т.е. $C'_0 = 0$, последние три строки матрицы состояния циклически сдвигаются вправо на разное число байт; вторая строка сдвигается на C'_1 байт, третий – на C'_2 байтов и четвертый – на C'_3 байтов; значения смещений C'_1, C'_2 и C'_3 в *Rijndael* зависят от длины блока *Nb*, их величины приведены в табл. 8.5.

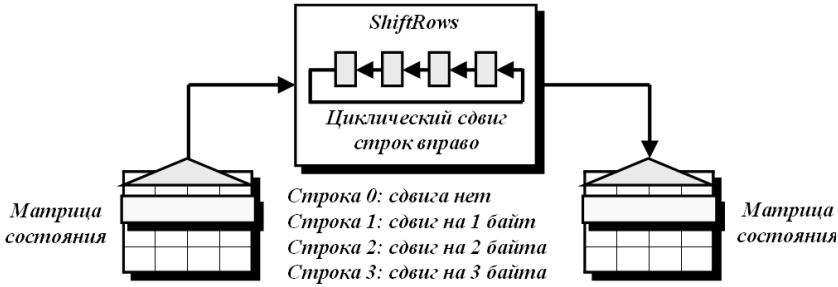


Рис. 8.15. Действие преобразования *InvShiftRows()* на строки матрицы состояния

Пример 8.9. Пусть на вход функции *ShiftRows()* алгоритма *AES* (*Nb = 4*) поступает матрица состояния равная:

$$State = f886b9f8b664688650fdf8b686f84886_{16}.$$

На рис. 8.16 показан результат выполнения сдвигов строк матрицы состояния влево с помощью функции *ShiftRows()* и вправо помощью функции *InvShiftRows()*.

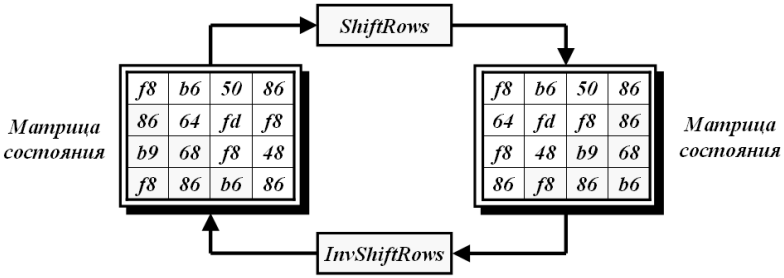


Рис. 8.16. Результат преобразования *ShiftRows()* и *InvShiftRows()* на строки матрицы состояния

8.5.3. Перемешивание столбцов матрицы состояния – *MixColumns()* и *InvMixColumns()*

Преобразование *MixColumns(State)* реализует также слой линейного преобразования и действует на каждый столбец s_i *(i = 0,1,…,3)* в блоке *State*, т.е. на каждое машинное слово, по правилу

$$s'_i(x) = g(x) \cdot \otimes \; s_i(x) \; mod \; (x^4 + 1). \qquad (8.27)$$

В этом преобразовании столбцы матрицы состояния (четыре байта) рассматриваются как многочлены над $GF(2^8)$ и умножаются по модулю $x^4 + 1$ на многочлен $g(x)$, который выглядит следующим образом [29, 38]:

$$g(x) = \{03\} \cdot x^3 + \{01\} \cdot x^2 + \{01\} \cdot x + \{02\}. \qquad (8.28)$$

Элементы поля $GF(2^8)$ записываются либо как битовый вектор, либо как байт. В частности, байт $\{03\}$ – это элемент поля $GF(2^8)$, представляемый многочленом $x + 1$ по модулю $m(x)$.

Так как умножение на многочлен – линейная операция, ее можно представить в виде действия матрицы:

$$\begin{vmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{vmatrix} = C \cdot \begin{vmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{vmatrix} = \begin{vmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{vmatrix} \cdot \begin{vmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{vmatrix}, \qquad (8.29)$$

где c – номер столбца массива *State* ($c = 0,1,…,3$); C – матрица коэффициентов многочлен $g(x)$.

В результате такого умножения байты столбца $s_{o,c}$, $s_{1,c}$, $s_{2,c}$ и $s_{3,c}$ заменяются соответственно на байты:

$$
\begin{aligned}
s'_{0,C} &= \{02\} \bullet s_{0,C} \oplus \{03\} \bullet s_{1,C} \oplus s_{2,C} \oplus s_{3,C}; \\
s'_{1,C} &= s_{0,C} \oplus \{02\} \bullet s_{1,C} \oplus \{03\} \bullet s_{2,C} \oplus \bullet s_{3,C}; \\
s'_{2,C} &= s_{0,C} \oplus s_{1,C} \oplus \{02\} \bullet s_{2,C} \oplus \{03\} \bullet s_{3,C}; \\
s'_{3,C} &= \{03\} \bullet s_{0,C} \oplus s_{1,C} \oplus s_{2,C} \oplus \{02\} \bullet s_{3,C}.
\end{aligned}
\tag{8.30}
$$

Применение этой операции ко всем четырем столбцам матрицы состояния обозначено как *MixColumns(State)*. Рис. 8.17 демонстрирует применение преобразования *MixColumns()* к столбцам матрицы состояния.

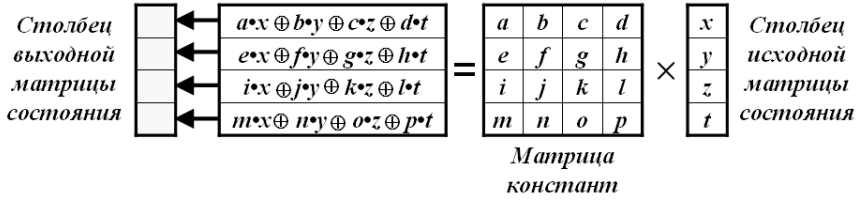


Рис. 8.17. Действие *MixColumns()* – перемешивание столбцов матрицы

Функция рассеивания *MixColumns()* была выбрана из возможных линейных преобразований *4 bytes* → *4 bytes* (четыре байта столбца матрицы состояния в четыре байта) исходя из следующих критериев выбора:

1. Обратимость.

2. Линейность в поле $gf(2^4)$; это свойство функции *MixColumns()* позволяет применять алгоритм прямого расшифрования.

3. Достаточно сильное рассеивание.

4. Скорость выполнения на *8*-разрядных процессорах.

5. Симметричность, т.е. *MixColumns()* должна работать со всеми данными единообразно.

6. Простота описания.

Критерии *2*, *5* и *6* обусловили выбор произведения многочленов по модулю $x^4 + 1$. Критерии *1*, *3* и *4* определили выбор коэффициентов сомножителей. Критерий *4* определил выбор коэффициентов (в порядке предпочтительности) - *{ 00 }, { 01 }, { 02 }, { 03 }, ...* . Значение *{ 00 }* не предполагает вообще никакого преобразования, *{ 01 }*

– не нужно производить умножения, *{ 02 }* – умножение может быть выполнено при помощи функции *xtime()*, и *{ 03 }* – при помощи функции *xtime()* и последующего *xor*.

Третий критерий определяет более сложные условия выбора коэффициентов. Стратегия разработчиков шифра учитывала следующие требования к преобразованию *MixColumns()*. Пусть *Z* будет линейным преобразованием, выполняемым над векторами байт, и пусть байтовым весом вектора будет количество ненулевых байт в нем. Тогда коэффициентом распространения линейного преобразования, характеризующим его силу рассеивания, будем называть величину

$$min_{a \neq 0} \{ Z(a) + Z(F(a)) \},$$

где *Z(a)* – байтовый вес.

Байт, не равный нулю, будем называть *активным байтом*. *MixColumns()* при одном активном байте входного *32*-разрядного массива *State* на выходе будет максимум *4* активных байтов, так как эта функция преобразует столбцы по отдельности. Таким образом, верхняя граница коэффициента распространения равна *5*. Коэффициенты *{ 01 }*, *{ 02 }* и *{ 03 }* и были подобраны так, чтобы получить это максимально возможное значение коэффициента распространения. То, что он равен *5*, говорит о том, что разница во входных данных в один байт рассеивается (распространяется) на четыре байта выходных данных, разница в двух байтах входных данных отразится, как минимум, на трех байтах выходных данных. Более того, можно сказать, что линейная зависимость между битами во входных и выходных данных всегда затрагивает биты как минимум пяти различных байт на входе и выходе функции *MixColumns()*.

Пример 8.5. Пусть на вход функции *MixColumns()* алгоритма *AES* (*Nb = 4*) поступает матрица состояния равная:

$$State = d4bf5d30e0b452aeb84111f11e2798e5_{16},$$

при этом

$$\begin{array}{llll} s_{0,0} = d4; & s_{0,1} = e0; & s_{0,2} = b8; & s_{0,3} = 1e; \\ s_{1,0} = bf; & s_{1,1} = b4; & s_{1,2} = 41; & s_{1,3} = 27; \\ s_{2,0} = 5d; & s_{2,1} = 52; & s_{2,2} = 11; & s_{2,3} = 98; \\ s_{3,0} = 30; & s_{3,1} = ae; & s_{3,2} = f1; & s_{3,3} = e5. \end{array}$$

Как известно матрица состояния на выходе будет определяться выражением (8.29). Определим столбец матрицы состояния на выходе для случая $c = 0$, используя выражение (8.30). В данном случае

$$s'_{0,0} = \{02\} \bullet s_{0,0} \oplus \{03\} \bullet s_{1,0} \oplus s_{2,0} \oplus s_{3,0} =$$

$$= \{02\} \bullet \{d4\} \oplus \{03\} \bullet \{bf\} \oplus \{5d\} \oplus \{30\} =$$

$$= \{02\} \bullet \{d4\} \oplus \{02\} \bullet \{bf\} \oplus \{bf\} \oplus \{5d\} \oplus \{30\}.$$

Представим значения $\{d4\}$, $\{bf\}$, $\{5d\}$ и $\{30\}$ в виде многочленов:

$$\{d4\} \rightarrow x^7 + x^6 + x^4 + x^2,$$

$$\{bf\} \rightarrow x^7 + x^5 + x^4 + x^3 + x^2 + x + 1,$$

$$\{5d\} \rightarrow x^6 + x^4 + x^3 + x^2 + 1,$$

$$\{30\} \rightarrow x^5 + x^4.$$

Вычислим произведение $\{02\} \bullet \{d4\}$:

$$\{02\} \bullet \{d4\} \rightarrow x \cdot (x^7 + x^6 + x^4 + x^2) \bmod p(x) =$$

$$= (x^8 + x^7 + x^5 + x^3) \bmod (x^8 + x^4 + x^3 + x + 1) =$$

$$= x^7 + x^5 + x^4 + x + 1.$$

Вычислим произведение $\{02\} \bullet \{bf\}$:

$$\{02\} \bullet \{bf\} \rightarrow x \cdot (x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) \bmod p(x) =$$

$$= (x^8 + x^6 + x^5 + x^4 + x^3 + x^2 + x) \bmod (x^8 + x^4 + x^3 + x + 1) =$$

$$= x^6 + x^5 + x^2 + 1.$$

Вычислим элемент $s'_{0,0}$:

$$s'_{0,0} = (x^7 + x^5 + x^4 + x + 1) \oplus (x^6 + x^5 + x^2 + 1) \oplus$$

$$\oplus (x^7 + x^5 + x^4 + x^3 + x^2 + x + 1) \oplus (x^6 + x^4 + x^3 + x^2 + 1) \oplus$$

$$\oplus (x^5 + x^4) = x^2 \rightarrow \{04\}.$$

Вычисляя аналогично элементы $s'_{1,0}$, $s'_{2,0}$ и $s'_{3,0}$, получим:

$$s'_{1,0} = x^6 + x^5 + x^2 + x \rightarrow \{66\};$$
$$s'_{2,0} = x^7 + 1 \rightarrow \{81\};$$
$$s'_{3,0} = x^7 + x^6 + x^5 + x^2 + 1 \rightarrow \{e5\}.$$

Представляя значения столбцов матрицы состояния при $c = 1, 2$ и 3 в виде многочленов и производя аналогичные преобразования получим матрицу состояния на выходе функции *MixColumns()* в виде (рис. 8.18):

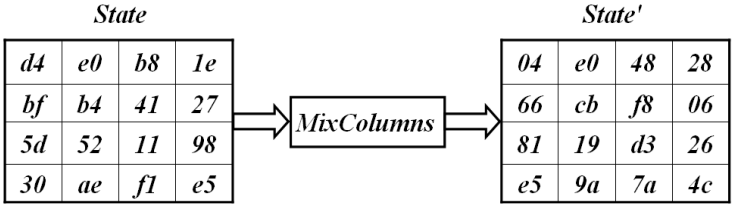


Рис. 8.18. Результат преобразования *MixColumns()* столбцов матрицы состояния

В преобразовании *MixColumns()* матрица коэффициентов C, входящая в (8.29), невырождена над $GF(2^8)$, поэтому операция *MixColumns()* обратима, а обратное к ней действие реализуется матрицей C^{-1}, обратной к исходной C.

В преобразовании *InvMixColumns(State)* столбцы состояния рассматриваются как многочлены над $GF(2^8)$ и умножаются по модулю $x^4 + 1$ на многочлен

$$s'(x) = (s(x) \otimes g^{-1}(x)) \bmod (x^4 + 1),$$

где $g^{-1}(x) = \{0b\} \cdot x^3 + \{0d\} \cdot x^2 + \{09\} \cdot x + \{0e\}$, который получается решением уравнения

$$g(x) \otimes g^{-1}(x) \bmod (x^4 + 1) = 1 \qquad (8.31)$$

относительно $g^{-1}(x)$ при $g(x) = \{03\} \cdot x^3 + \{01\} \cdot x^2 + \{01\} \cdot x + \{02\}$.

Это может быть представлено в матричном виде следующим образом:

$$\begin{vmatrix} s'_{0c} \\ s'_{1c} \\ s'_{2c} \\ s'_{3c} \end{vmatrix} = C^{-1} \cdot \begin{vmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{vmatrix} = \begin{vmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{vmatrix} \begin{vmatrix} s_{0c} \\ s_{1c} \\ s_{2c} \\ s_{3c} \end{vmatrix},$$

где c – номер столбца матрицы C^{-1} ($c = 0,1,...,3$).

В результате такого умножения байты столбца $s_{o,c}$, $s_{1,c}$, $s_{2,c}$ и $s_{3,c}$ заменяются соответственно на байты:

$$s'_{0C} = (\{0e\} \bullet s_{0C}) \oplus (\{0b\} \bullet s_{1C}) \oplus (\{0d\} \bullet s_{2C}) \oplus (\{09\} \bullet s_{3C});$$
$$s'_{1C} = (\{09\} \bullet s_{0C}) \oplus (\{0e\} \bullet s_{1C}) \oplus (\{0b\} \bullet s_{2C}) \oplus (\{0d\} \bullet s_{3C});$$
$$s'_{2C} = (\{0d\} \bullet s_{0C}) \oplus (\{09\} \bullet s_{1C}) \oplus (\{0e\} \bullet s_{2C}) \oplus (\{0b\} \bullet s_{3C});$$
$$s'_{3C} = (\{0b\} \bullet s_{3C}) \oplus (\{0d\} \bullet s_{0C}) \oplus (\{09\} \bullet s_{1C}) \oplus (\{0e\} \bullet s_{2C}).$$

Рис. 8.19 показывает матрицы констант, используемых для преобразований *MixColumns()* и *InvMixColumns()*.

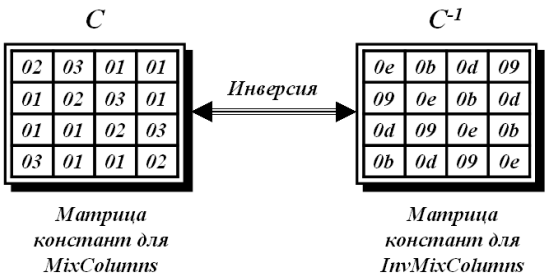


Рис. 8.19. Матрицы констант, использующие преобразование *MixColumns()* и *InvMixColumns()*

Эти две матрицы обратные друг другу, когда элементы интерпретируются как слова из *8* битов (или многочлены) с коэффициентами в $GF(2^8)$ [29, 38].

Преобразование *MixColumns()* работает на уровне столбца; оно превращает каждый столбец матрицы состояния в новый столбец. Это преобразование – фактически матричное умножение столбца матрицы состояния и квадратной матрицы констант. Байты в столб-

це матрицы состояния и в матрице констант интерпретируются как слова по *8* битов (или многочлены) с коэффициентами в $GF(2^8)$. Умножение байтов выполняется в $GF(2^8)$ с неприводимым многочленом $p(x) = x^8 + x^4 + x^3 + x + 1$. Сложение – это применение операции *xor* к словам по *8* бит.

Преобразования *InvMixColumns()* похоже на *MixColumns()*-преобразования. Рис. 8.20 показывает принцип преобразования с использованием *MixColumns()* или *InvMixColumns()*.

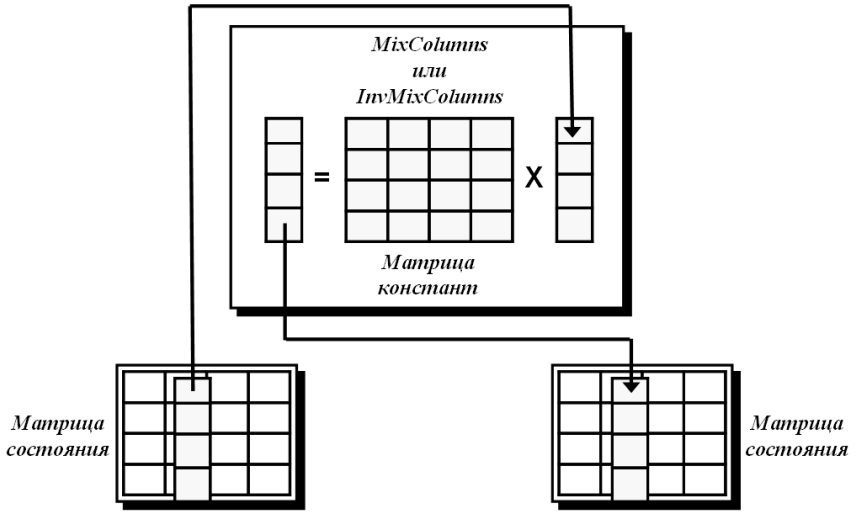


Рис. 8.20. Принцип преобразования с использованием *MixColumns()* или *InvMixColumns()*

Пример 8.11. Пусть на вход преобразования *MixColumns()* поступает матрица состояния, которая равна:

$$State = 63f27dd4c963d4fafe26c96330f2c982_{16}.$$

Рис. 8.21 показывает, как матрица состояния превратится, если использовать преобразование *MixColumns()*. Рисунок также показывает, что преобразования *InvMixColumns()* создает первоначальное значение матрицы состояния.

Заметим, что байты, которые равны между собой в старой матрицы состояний, больше не равны в новой матрицы состояний. Например, два байта *f2* во второй строке изменены на *cf* и *0d*.

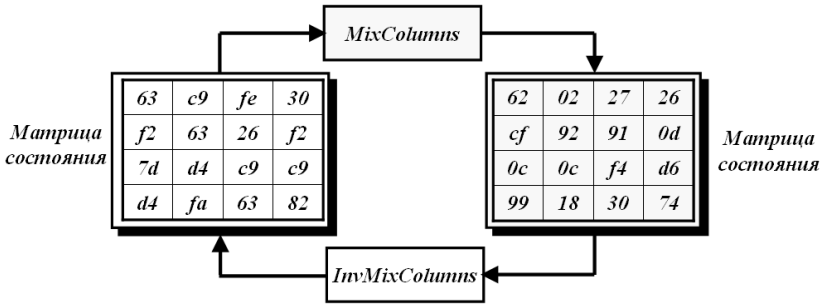


Рис. 8.21. Преобразования *MixColumns()* и *InvMixColumns()* для примера 8.11

8.5.4. Сложение раундового ключа с матрицей состояния – *AddRoundKey(State, RoundKey)*

Вероятно, самое важное преобразование – это преобразование, которое включает ключ шифра. Все предыдущие преобразования используют известные алгоритмы, которые являются обратными. Если не добавлять ключевой шифр в каждом раунде, противник очень просто определит исходное сообщение по данному ему зашифрованному. В этом случае хранителем тайны будет являться только ключ шифра.

AddRoundKey() обрабатывает в один момент времени один столбец. Преобразования подобное *MixColumns()*. *MixColumns()* умножает квадратную матрицу констант на каждый столбец матрицы состояний. *AddRoundKey()* добавляет ключевое слово раунда с каждым столбцом матрицы состояния. В *MixColumns()* применяется матричное умножение, в *AddRoundKey()* – операции сложения и вычитания. Поскольку сложение и вычитание в конечном поле те же самые, то *AddRoundKey()* обратной сама к себе. Рис. 8.22 показывает принцип преобразования AddRoundKey().

Преобразование *AddRoundKey(State, RoundKey)* реализует слой сложения обрабатываемых данных (состоянием) с раундовым ключом. В данной операции раундовый ключ добавляется к матрице состояния посредством простого поразрядного *xor*. Раундовый ключ *W* вырабатывается из ключа шифрования *K* посредством алгоритма выработки ключей. Длина раундового ключа (в *32*-разрядных словах) равна длине блока *Nb*.

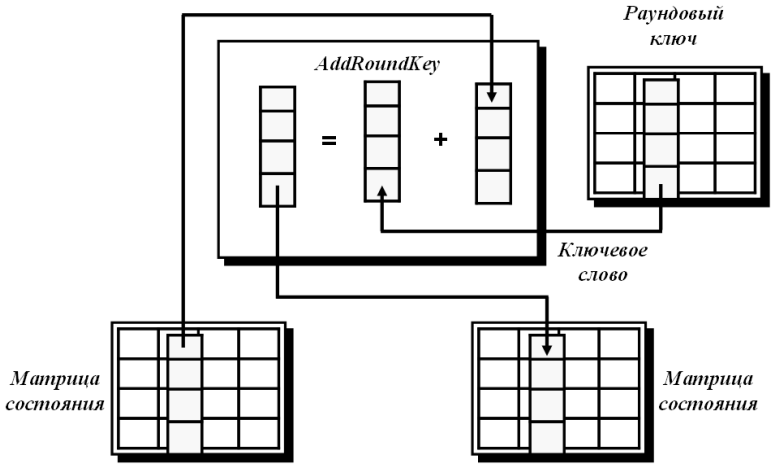


Рис. 8.22. Принцип перетворення масиву стану за допомогою *AddRoundKey()*

Пример 8.12. Пусть на вход функции *AddRoundKey()* алгоритма *AES* (*Nb = 4*) поступает:

- матрица состояния:

$$State = d4bf5d30e0b452aeb84111f11e2798e5_{16};$$

- матрица раундового ключа:

$$RoundKey = 046681e5e0cb199a48f8d37a2806264c_{16}.$$

Как известно матрица состояния на выходе будет определяться путем поразрядного суммирования по модулю *2* (выполнения операции *xor*) столбцов матрицы состояния и матрицы раундового ключа.

Выполняя такие действия над столбцами матрицы состояния и матрицы раундового ключа получим матрицу состояния на выходе функции *AddRoundKey(State, Round Key)* (рис. 8.23).

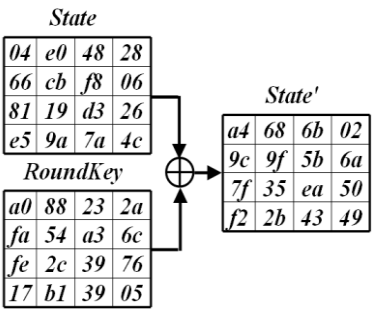


Рис. 8.23. Действие *AddRoundKey()* на байты матрицы состояния для примера 8.12

Подстановка, которая делается преобразованием *SubByte()*, и которое изменяет значения байта, осно-

ванное только на начальном значении и входе в таблицу; процесс не включает соседние байта. Можно сказать, что *SubByte()* – внутреннее байтовое преобразования. Перестановка, которая делается *ShiftRows()* – преобразованием, меняет местами байты, не переставляя биты в байтах.

Можно сказать, что *ShiftRows()* – преобразования обмена байтами. Теперь нам нужно внутреннее байтовое преобразование, изменяющее биты в байтах и основанное на битах в соседних байтах. Необходимо смешать байты, чтобы обеспечить рассеивание на разрядном уровне.

Преобразования смешивания *MixColumns()* изменяет содержание каждого байта, превращая четыре байта одновременно и объединяя их, чтобы получить четыре новых байта. Чтобы гарантировать, что каждый новый байт будет отличаться от другого (даже если все четыре байта те же), процесс сначала умножает каждый байт на разный набор констант и затем смешивает их. Смешивание может быть обеспечено матричным умножением. Когда умножаем квадратную матрицу на матрицу-столбец, результат – новая матрица-столбец. После того как матрица умноженная на значение строки в матрице состояния, каждый элемент в новой матрицы будет зависит от всех четырех элементов старой матрицы.

8.6. АЛГОРИТМ РАЗВОРАЧИВАНИЯ КЛЮЧА ДЛЯ ШИФРОВАНИЯ ДАННЫХ AES

Алгоритм разворачивания ключа определяет порядок получения раундовых ключей W из начального ключа шифрования K. Раундовые ключи получаются из ключа шифрования посредством алгоритма выработки ключей. Он содержит два компонента:

- расширение ключа (*Key Expansion*);
- 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 разворачивается 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.

8.6.1. Расширение ключа *(KeyExpansion)*

Для того, чтобы создать ключ для каждого раунда, *AES* использует процесс ключевого расширения. Если количество раундов *Nr*, процедура расширения ключей создает *Nr + 1* раундовых ключей на *128* битов каждый из единого *128*-битного ключа шифрования. Первые раундовые ключи используются для преобразования перед началом раундов (раунд *0*) с использованием преобразования *AddRoundKey()*; раундовые ключи, которые остаются, применяются для дальнейших преобразований *AddRoundKey()* в конце каждого раунда.

Процедура расширения ключей создает слово за словом, из которых в дальнейшем будут формироваться раундовые ключи. Слово – это массив из четырех байтов. Процедура расширения ключей создает $4 \times (Nr + 1)$ слов, которые обозначаются

$$W = w_0, w_1, w_2 \ldots, w_{4 \times (Nr + 1)}, w_{4 \times (Nr + 1) - 1}.$$

Другими словами, в версии *AES-128* ($Nr = 10$) будет $W = 44$ слова; в версии *AES-192* ($Nr = 12$) – $W = 52$ слова; и в версии *AES-256* ($Nr = 14$) – $W = 60$ слов. Каждый раундовый ключ состоит из четырех слов. Табл. 8.6 показывает отношение между раундами и словами.

Таблица 8.6

Слова для каждого раунда алгоритма AES

Раунд	*Слова*			
0	w_0	w_1	w_2	w_3
1	w_4	w_5	w_6	w_7
2	w_8	w_9	w_{10}	w_{11}
. . .	. . .	. . .	. . .	. . .
Nr	$w_{4 \times Nr}$	$w_{4 \times Nr+1}$	$w_{4 \times Nr+2}$	$w_{4 \times Nr+3}$

Рассмотрим процесс расширения ключа для версии *AES-128*; процессы для других версий за исключением небольших изменений такие же. Рис. 8.24 показывает, как из исходного ключа длинной *128* бит получить *44* слова.

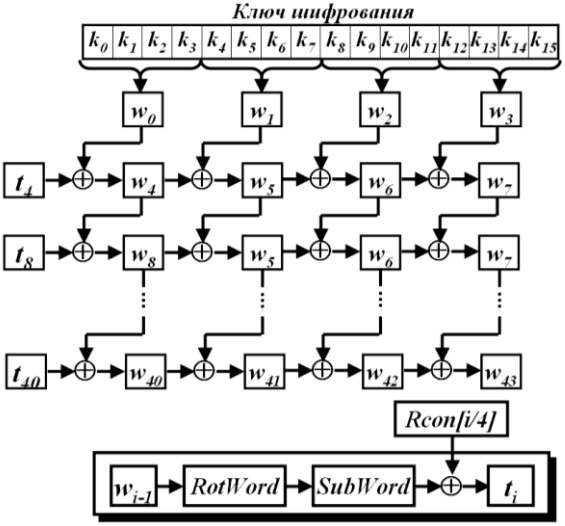


Рис. 8.24. Пояснение процесса расширения ключей в *AES-128*

Процесс расширения ключа *AES-128* такой:

1. Первые четыре слова (w_0, w_1, w_2, w_3) выходят из ключа шифрования. Ключ шифрования представлен как массив из *16* байт (от k_0 до k_{15}). Первые четыре байта (от k_0 до k_3) становятся w_0; следующие четыре байта (от k_4 до k_7) становятся w_1 и так далее. Другими словами, последовательное соединение (конкатенация) слов в этой группе копирует ключ шифрования.

2. Остальные части слов w_i от $i = 4$ до $i = 43$ получается так:

а) если $(i \bmod 4) \neq 0$, то $w_i = w_{i-1} \oplus w_{i-4}$ (согласно рис. 8.24 это означает, что каждое слово получено из одного левого и одного верхнего);

б) если если $(i \bmod 4) = 0$, то $w_i = t_i \oplus w_{i-4}$ (здесь t_i – временное слово, результат применения двух процессов, *SubWord* и *RotWord*, со словом w_{i-1} и применения операции *xor* с константой раунда *Rcon[i/4]*).

Другими словами, имеем

$$t_i = SubWord(RotWord(w_{i-4})) \oplus Rcon[i/4]. \qquad (8.32)$$

RotWord (*Rotate Word*) – процедура, подобная преобразования *ShiftRows()*, но применяется только к одному столбцу. Процедура

принимает слово как массив из четырех байтов и сдвигает каждый байт вверх с конвертацией (циклический сдвиг байтов).

SubWord (*Substitute Word*) – процедура, подобная преобразованию *SubBytes()*, но применяется только к одному столбцу. Процедура принимает каждый байт в слове и заменяет его другим.

Каждая константа раунда *Rcon* (*RoundConstants*) – это *4*-байтовое значение, в котором крайние правые (старшие) три байта всегда нулевыми.

Значение константы раунда *Rcon[i/4]* определяется выражением

$$Rcon[i/4] = R_{con}[j] = (2^{j-1}) \bmod p(x). \qquad (8.33)$$

Значение раундовых констант *Rcon*, рассчитанных по формуле (8.33), приведены в табл. 8.7.

Таблица 8.7

Значения констант раундов *Rcon* (*Nk = 4*)

i	*i/Nk*	*Rcon[i/Nk]*	*i*	*i/Nk*	*Rcon[i/Nk]*
4	*1*	*{ 01000000 }*	*24*	*6*	*{ 20000000 }*
8	*2*	*{ 02000000 }*	*28*	*7*	*{ 40000000 }*
12	*3*	*{ 04000000 }*	*32*	*8*	*{ 80000000 }*
16	*4*	*{ 08000000 }*	*36*	*9*	*{ 1b000000 }*
20	*5*	*{ 10000000 }*	*40*	*10*	*{ 36000000 }*

Процедура расширения ключа может использовать или приведенную выше таблицу, когда вычисляет слова, или поле $GF(2^8)$, когда вычисляет крайние левые биты динамично, как это показано ниже.

$$Rcon[i = 4/Nk]|_{Nk=4} = Rcon[1] = 2^{1-1} \bmod p(x) = 01_{16};$$
$$Rcon[i = 8/Nk]|_{Nk=4} = Rcon[2] = 2^{2-1} \bmod p(x) = 02_{16};$$
$$Rcon[i = 12/Nk]|_{Nk=4} = Rcon[3] = 2^{3-1} \bmod p(x) = 04_{16};$$
$$Rcon[i = 16/Nk]|_{Nk=4} = Rcon[4] = 2^{4-1} \bmod p(x) = 08_{16};$$
$$Rcon[i = 20/Nk]|_{Nk=4} = Rcon[5] = 2^{5-1} \bmod p(x) = 10_{16};$$
$$Rcon[i = 24/Nk]|_{Nk=4} = Rcon[6] = 2^{6-1} \bmod p(x) = 20_{16};$$
$$Rcon[i = 28/Nk]|_{Nk=4} = Rcon[7] = 2^{7-1} \bmod p(x) = 40_{16};$$
$$Rcon[i = 32/Nk]|_{Nk=4} = Rcon[8] = 2^{8-1} \bmod p(x) = 80_{16};$$
$$Rcon[i = 36/Nk]|_{Nk=4} = Rcon[9] = 2^{9-1} \bmod p(x) = 1b_{16};$$
$$Rcon[i = 40/Nk]|_{Nk=4} = Rcon[10] = 2^{10-1} \bmod p(x) = 36_{16}.$$

Байт, который обозначен как *Rcon[i]* – это x^{i-1}, где i – номер раунда.

Ниже приведен псевдокод программы, которая показывает процесс развертывания ключа (*Key Expansion*).

```
//================================================
//= Псевдокод функции развертывания ключа KeyExpansion() =
//================================================
KeyExpansion (byte key[4*Nk], word w[Nb*(Nr + 1)], Nk)
begin
    word temp
    i = 0
    while (i < Nk)
        w[i] = word(key[4*i], key[4*i+l], key[4*i+2], key[4*i+3])
        i = i+1
    end while
    i = Nk
    while (i < Nb (Nr+1))
        temp = w[i-l]
        if (i mod Nk = 0)
                temp = SubWord(RotWord(temp)) xor Rcon[i/Nk]
                else if (Nk > 6 and i mod Nk = 4) temp = SubWord(temp)
        end if
        w[i] = w[i-Nk] xor temp
        i = i + 1
    end while
end
//================================================
```

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 со слов с меньшими индексами. Алгоритм формирования ключей зависит от величины *Nk*.

Пример 8.13. Пусть на вход функции *KeyExpansion()* алгоритма *AES* (*Nb = 4*, *Nk = 4*) поступает ключ шифрования *K*:

w_0	w_1	w_2	w_3
2b	28	ab	09
7e	ae	f7	cf
15	d2	15	4f
16	a6	88	3c

Сформировать четырехбайтовые слова расширенного ключа: w_4, w_5, w_6 и w_7.

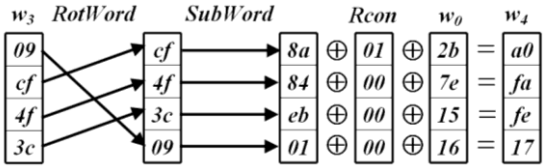
Решение. Сформируем слово w_4. В связи с тем, что

$$i \bmod N_k = 4 \bmod 4 = 0,$$

то слово w_4 будет сформировано с использованием (8.32)

$$w_4 = SubWord(RotWord(w_3)) \oplus w_0 \oplus R_{con}[1]$$

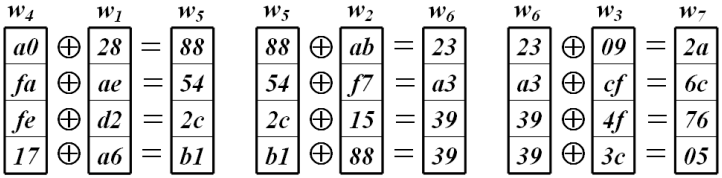
или



Сформируем слова w_5, w_6 и w_7. Поскольку при $i = 5, 6, 7$ $i \bmod Nk \neq 0$, то слова будут сформированы с использованием (8.32) таким образом:

$$w_5 = w_4 \oplus w_1;\ w_6 = w_5 \oplus w_2;\ w_7 = w_6 \oplus w_3$$

или



Следовательно, с учетом четырехбайтовых слов ключа шифрования K и сформированных четырехбайтовых слов: w_4, w_5, w_6 и w_7, слова расширенного ключа примут вид

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7
2b	28	ab	09	a0	88	23	2a
7e	ae	f7	cf	fa	54	a3	6c
15	d2	15	4f	fe	2c	39	76
16	a6	88	3c	17	b1	39	05

8.6.2. Выбор раундового ключа *(Round Key Selection)*

Раундовый ключ для i-го раунда получается из слов массива раундового ключа от w_{Nb-i} и до $w_{Nb(i+1)-1}$, как показано на рис. 8.25.

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}	w_{12}	. . .
Раундовый ключ 0				*Раундовый ключ 1*				*Раундовый ключ 2*				. . .	

Рис. 8.25. Процедуры расширения ключа и выбора раундового ключа для *Nk = 4*

Пример 8.14. Пусть после выполнения функции *KeyExpansion()* алгоритма *AES* (*Nb = 4, Nk = 4*) получены следующие четырехбайтовых слова расширенного ключа:

w_0	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}	w_{11}
2b	*28*	*ab*	*09*	*a0*	*88*	*23*	*2a*	*f2*	*7a*	*59*	*73*
7e	*ae*	*f7*	*cf*	*fa*	*54*	*a3*	*6c*	*c2*	*96*	*35*	*59*
15	*d2*	*15*	*4f*	*fe*	*2c*	*39*	*76*	*95*	*b9*	*80*	*f6*
16	*a6*	*88*	*3c*	*17*	*b1*	*39*	*05*	*f2*	*43*	*7a*	*7f*

Сформировать раундовые ключи для нулевого, первого и другого раундов шифрования данных.

Решение. Раундовый ключ для нулевого раунда (K_0) выходит со слов массива раундового ключа от $w_{Nb\cdot 0} = w_0$ и до $w_{Nb\cdot(0+1)-1} = w_3$. Раундовый ключ для первого раунда (K_1) выходит со слов массива раундового ключа от $w_{Nb\cdot 1} = w_4$ и до $w_{Nb\cdot(1+1)-1} = w_7$. Раундовый ключ для второго раунда (K_2) выходит со слов массива раундового ключа от $w_{Nb\cdot 2} = w_8$ и до $w_{Nb\cdot(2+1)-1} = w_{11}$. Итак, раундовые ключи с учетом его представления, как показано на рис. 8.25, примут вид

Раундовый ключ 0			
2b	*28*	*ab*	*09*
7e	*ae*	*f7*	*cf*
15	*d2*	*15*	*4f*
16	*a6*	*88*	*3c*

Раундовый ключ 1			
a0	*88*	*23*	*2a*
fa	*54*	*a3*	*6c*
fe	*2c*	*39*	*76*
17	*b1*	*39*	*05*

Раундовый ключ 2			
f2	*7a*	*59*	*73*
c2	*96*	*35*	*59*
95	*b9*	*80*	*f6*
f2	*43*	*7a*	*7f*

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.

Алгоритм разворачивания ключа должен обеспечивать стойкость против следующих типов атак:

- атак, в которых часть начального ключа шифрования криптографическому аналитику известна;

- атак, в которых ключ шифрования известен заранее или может быть выбран, например, если шифр применяется для компрессии данных при хешировании;

- атак “эквивалентных ключей” [38]; необходимым условием стойкости к таким атакам является отсутствие слишком больших одинаковых наборов раундовых ключей, полученных из двух разных начальных ключей шифрования.

Процедура разворачивания ключа также играет важную роль в исключении:

- симметрии одноraундового преобразования, которое обрабатывает все входные байты единообразно; для ее устранения в процедуре разворачивания ключа при получении каждого первого *32*-разрядного слова раундового ключа используются функция *SubWord(RotWord())* и раундовая константа;

- межраундовой симметрии (раундовое преобразование одинаково для всех итераций цикла преобразования); для ее нарушения в алгоритм разворачивания ключа введены константы, значения которых различны для каждого раунда.

Таким образом, алгоритм разворачивания выбран в соответствии со следующим критериями:

- обратимость используемых преобразований, т.е. из любых *Nk* последовательных слов развернутого ключа можно однозначно восстановить весь развернутый ключ;

- хорошая скорость на различных типах процессоров;

- наличие раундовых констант для уменьшения симметричности;

- хорошее рассеивание изменений в начальном ключе шифрования на формирующийся раундовый ключ;

- отсутствие возможности на основе знания части бит раундового или начального ключа вычислить значительную часть остальных бит;

- достаточная нелинейность для предупреждения полного определения межбитовых зависимостей развернутого раундового ключа лишь на основании знания таких зависимостей в начальном ключе шифрования;

- простота описания.

Была выбрана байт-ориентированная схема алгоритма, которая может эффективно реализовываться на *8*-разрядных процессорах. Применение *SubBytes()* обеспечивает нелинейность преобразования и требует небольшого дополнительного пространства памяти на *8*-разрядных процессорах.

Раундовые ключи для расшифрования используются в обратном порядке по отношению к раундовым ключам для шифрования.

8.7. ЗАШИФРОВАНИЕ ДАННЫХ AES

Структура шифра *AES* (*Rijndael)* состоит из (рис. 8.26):

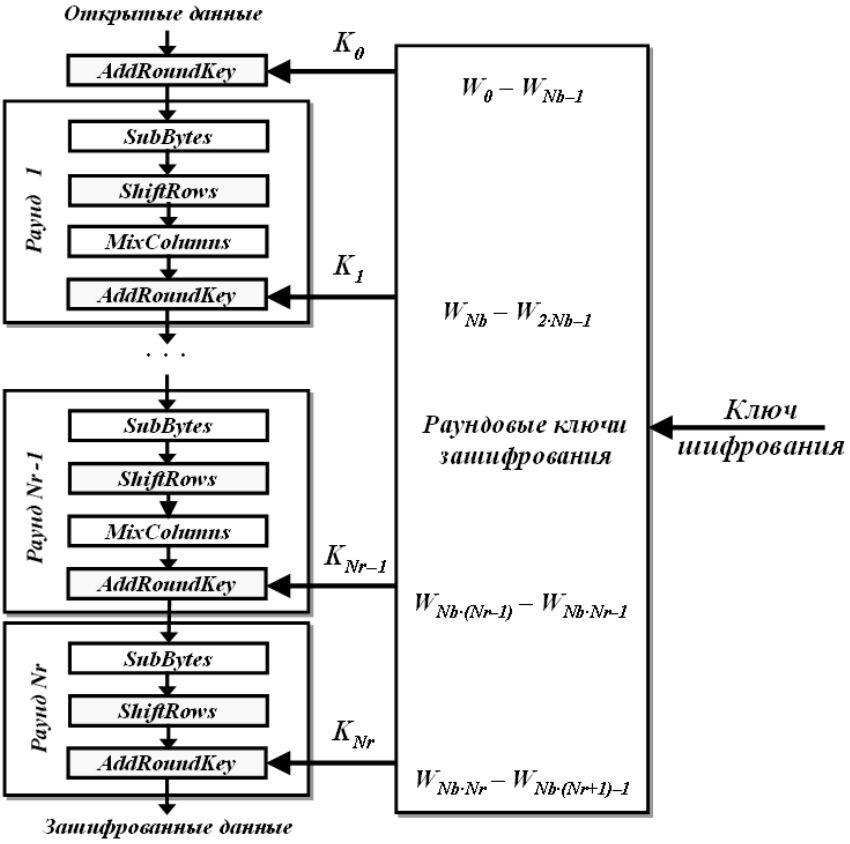


Рис. 8.26. Схема процедуры зашифрования в криптографическом алгоритме *AES* (*Rijndael*)

- начального добавления раундового ключа;

- *Nr - 1* раундов зашифрования данных;

- заключительного раунда зашифрования данных, в котором отсутствует операция *MixColumns()*.

На вход алгоритма подаются блоки данных *State* (открытые данные), в ходе преобразований содержимое блоков изменяется и на выходе образуется зашифрованные данные, организованные опять же в виде блоков *State*, как показано на рис. 8.27, где $Nb = 4$, in_m и out_m – m-е байты соответственно входного и выходного блоков, $m = 0,1,\ldots,15$, s_{ij} – байт, находящийся на пересечении i-й строки и j-го столбца массива *State*, $i = 0,1,\ldots,3$ и $j = 0,1,\ldots,3$.

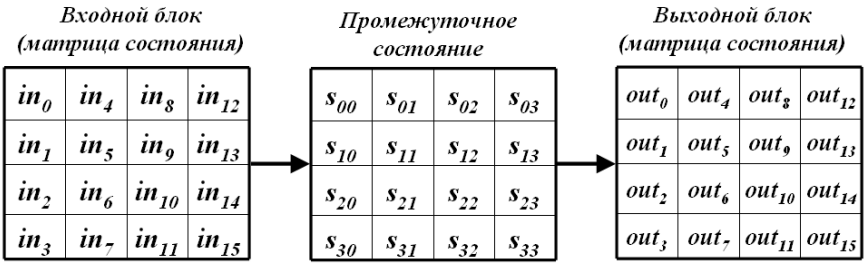


Рис. 8.27. Ход преобразования данных, организованных в виде блоков *State*

Перед началом первого раунда зашифрования данных происходит суммирование по модулю *2* с начальным ключом шифрования (раундовым ключом *0*), затем – преобразование матрицы состояния (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()*. На первый взгляд это кажется странным решением, ухудшающим структуру шифра. Но это не так.

Рис. 8.28 демонстрирует рассеивающие и перемешивающие свойства шифра. Видно, что уже два раунда обеспечивают полное рассеивание и перемешивание.

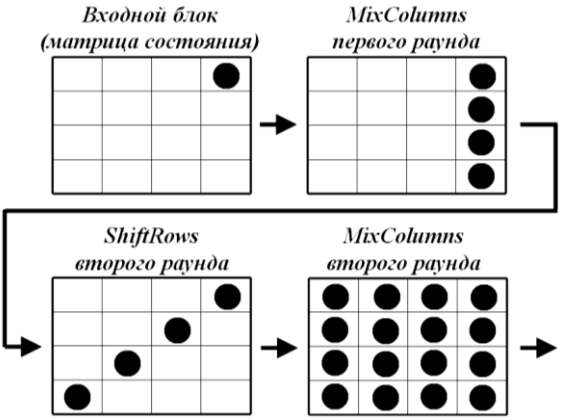


Рис. 8.28. Принцип действия криптографического алгоритма *AES* (*Rijndael*): ● – измененный байт

Ниже приведен псевдокод программы, которая показывает процесс зашифрования данных криптографическим алгоритмом *AES* (*Rijndael*) – *Cipher*.

```
//=============================================
//=====  Процедура зашифрования данных в псевдокоде  ===
//=============================================
Cipher (byte in[4*Nb], byte out[4*Nb], word w[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, w[0, Nb-1])
    for round = 1 step 1 to Nr-1
        SubBytes(state)
        ShiftRows(state)
        MixColumns(state)
        AddRoundKey(state, w[round*Nb, (round+1)*Nb-1])
    end for
    SubBytes(state)
    ShiftRows(state)
    AddRoundKey(state, w[Nr*Nb, (Nr+1)*Nb-1])
    out = state
end
//=============================================
```

8.8. РАСШИФРОВАНИЕ ДАННЫХ AES

8.8.1. Обратное (инверсное) расшифрование данных

Обозначим преобразование *SubBytes()*, *ShiftRows()*, *MixColumns()* и *AddRoundKey()* алгоритма зашифрования соответственно через *B*, *R*, *C* и *K*. Запишем всю последовательность действий алгоритма зашифрования для случая длины ключа *128* бит в виде линейной цепочки:

$$\underbrace{KBRCK}_{\text{1-й раунд}}\underbrace{BRCK}_{\text{2-й раунд}}\ldots\underbrace{BRCK}_{\text{9-й раунд}}\underbrace{BRK}_{\text{10-й раунд}}. \qquad (8.34)$$

Если вместо *SubBytes()*, *ShiftRows()*, *MixColumns()* и *AddRoundKey()* в последовательности (8.34) выполнить инверсные им преобразования и перегруппировав преобразования в раундах, можно построить функцию обратного расшифрования (рис. 8.29).

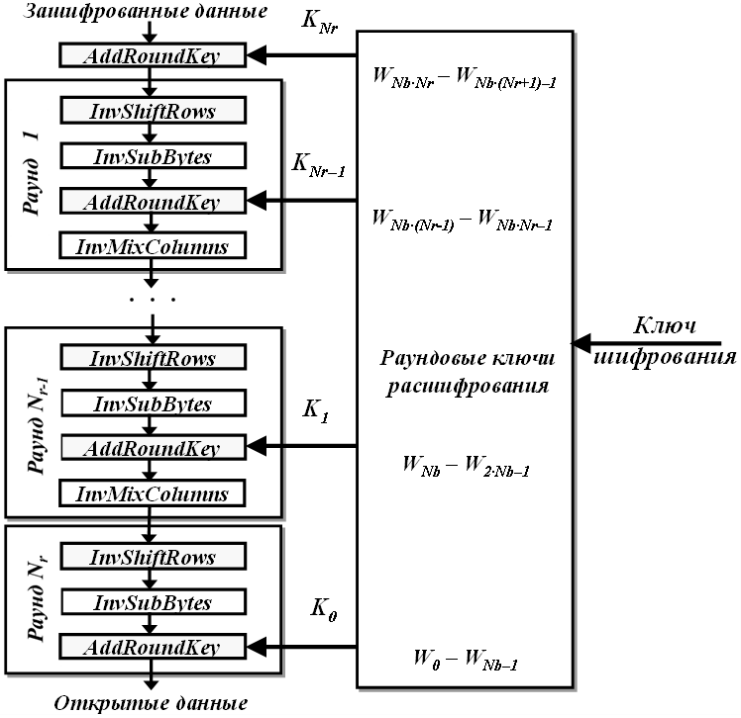


Рис. 8.29. Схема процедуры обратного (инверсного) расшифрования в криптографическом алгоритме *AES* (*Rijndael*)

Как видно на рис. 8.29 порядок использования раундовых ключей является обратным по отношению к тому, который используется при зашифровании.

Ниже приведены псевдокод программы, которая показывает процесс обратного (инверсного) расшифрования данных (*InvDecipher*).

```
//=============================================
//= Процедура обратного расшифрования в псевдокоде =
//=============================================
InvDecipher (byte in[4*Nb], word w[Nb*(Nr+1)], byte out[4*Nb])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey (state, w[Nr*Nb,(Nr+1)*Nb-1])
    InvShiftRows (state)
    InvSubBytes (state)
    for round = 1 step 1 to Nr-1
        AddRoundKey (state, w[(Nr-round)*Nb,
        (Nr+1-round)*Nb-1])
        InvMixColumns (state)
        InvShiftRows (state)
        InvSubBytes (state)
    end for
    AddRoundKey (state, w[0,Nb-1])
    out = state
end
//=============================================
```

На рис. 8.29 *InvAddRoundKey()* в начальном преобразовании и последнем раунде заменено на *AddRoundKey()*. Это обусловлено тем, что операция сложение и вычитание по модулю *2* являются одинаковыми.

В первом (начальном) проекте *AES* порядок преобразований в каждом раунде при зашифровании и расшифровании не совпадал (рис. 8.30).

Как видно из рис. 8.30, во-первых, при расшифровании изменяется порядок выполнения преобразований *SubBytes* (*InvSubBytes*) и *ShiftRows* (*InvShiftRows*); во-вторых, при расшифровании изменен порядок выполнения преобразований *MixColumns* (I*nvMixColumns*) и *AddRoundKey*.

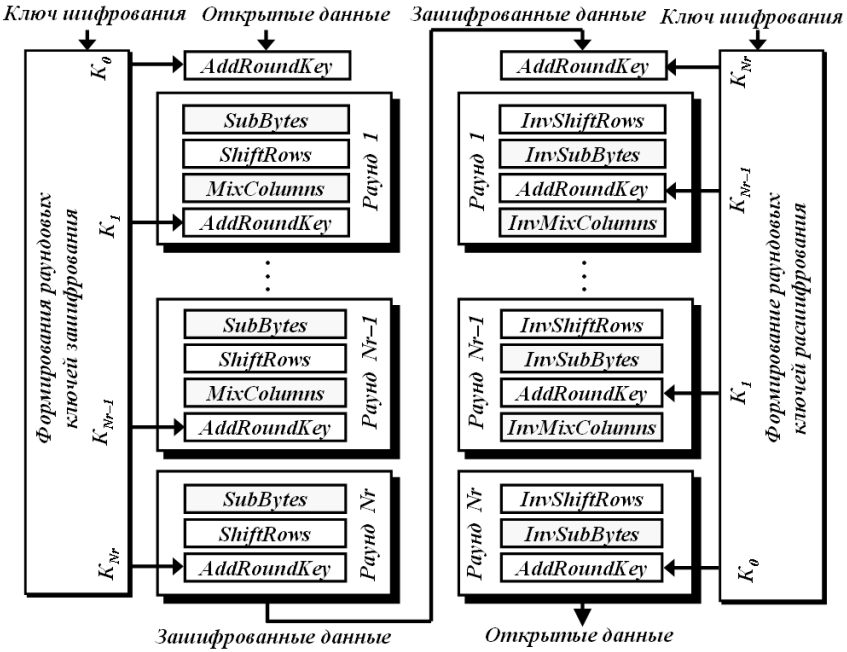


Рис. 8.30. Пояснение процессов зашифрования и обратного расшифрования в криптографическом алгоритме *AES* (*Rijndael*)

Эти изменения в порядке необходимы, чтобы при расшифровании сделать порядок работы преобразований инверсным в отношении порядка работы при зашифровании. Итак, алгоритм расшифрования в целом – инверсия алгоритма зашифрования. На рис. 8.30 показано только три раунда, но другие имеют тот же вид.

Необходимо обратить внимание, что ключи раунда при расшифровании используются в измененном (обратном) порядке, а также на то, что алгоритмы зашифрования и расшифрования в первоначальном проекте не совпадают.

8.8.2. Прямое (эквивалентное) расшифрование данных

Для тех приложений, которым нужны соответствующие алгоритмы для зашифрования и расшифрования, был разработан алгоритм прямого (эквивалентного) расшифрования. В такой версии, которая называется альтернативной, преобразования при расшиф-

рировании построены так, чтобы сделать порядок преобразований таким же как и при зашифровании.

Если преобразования *B* и *R*, а также *K* и *C* в (8.34) поменять местами без изменения результата

$$\underbrace{KRBKC}_{\text{1-й раунд}}\underbrace{RBKC}_{\text{2-й раунд}}\ldots\underbrace{RBKC}_{\text{Nr-1-й раунд}}\underbrace{RBK}_{\text{Nr-й раунд}},$$

то эту последовательность прочитанную справа налево, можно записать в виде

$$\underbrace{KBRCK}_{\text{1-й раунд}}\underbrace{BRCK}_{\text{2-й раунд}}\ldots\underbrace{BRCK}_{\text{9-й раунд}}\underbrace{BRK}_{\text{10-й раунд}}. \qquad (8.35)$$

Последовательность (8.35) теперь точно совпадает с (8.34). Это означает, что блок (8.34) можно расшифровать, используя ту же последовательность действий, что и при его зашифрувании, но с использованием раундовых ключей в обратном порядке.

Действительно, преобразование *SubBytes()* можно поменять местами с преобразованием *ShiftRows()*. То же самое будет правильным и для преобразований *InvSubBytes()* и *InvShiftRows()*. Это происходит потому, что функции *SubBytes()* и *InvSubBytes()* работают с байтами, а операции *ShiftRows()* и *InvShiftRows()* сдвигают байты, не затрагивая их значений.

Преобразования *SubBytes()* изменяет содержание каждого байта, не меняя порядок байтов матрицы состояния; *ShiftRows()* изменяет порядок байтов в матрице состояния, не изменяя содержание байтов. Поэтому, можно изменить порядок этих двух преобразований при расшифровании, не затрагивая обратимость целого алгоритма; рис. 8.31 иллюстрирует эту идею.

Рис. 8.31. Замена порядка выполнения преобразований *SubBytes()* и *InvShiftRows()*, а также *ShiftRows()* и *InvSubBytes()*

Заметим, что комбинации двух преобразований при зашифровании и расшифровании инверсные друг другу.

Алгоритм обратного (инверсного) расшифрования, описанный выше, имеет порядок преобразований, обратный порядку в алгоритме зашифрования, но использует те же параметры (развернутый ключ). Однако некоторые свойства алгоритма зашифрования *AES* (*Rijndael*) позволяют применить для расшифрования такой же порядок преобразований (используемый для зашифрования) благодаря изменении некоторых параметров, а именно – развернутого ключа:

$$\begin{aligned} &InvMixColumns(AddRoundKey()) = \\ &= InvMixColumns(State \oplus RoundKey) = \\ &= InvMixColumns(State) \oplus InvMixColumns(RoundKey). \end{aligned} \tag{8.36}$$

Эти свойства функций алгоритма расшифрования позволяют изменить порядок применения преобразований *AddRounKey()* и *InvMixColumns()*. Такое изменение порядка исполнения, как следует из (8.36), возможно при условии, что все раундовые ключи расшифрования (кроме начального раундового ключа – K_0 и последнего – K_{Nr}) предварительно пропущенные (преобразованные) через функцию *InvMixColumns()*.

Преобразования *InvMixColumns()* и *AddRoundKey()* имеют различные свойства, присущие только им. Однако эти преобразования могут стать инверсиями друг другу, если умножить матрицу раундовых ключей на инверсию матрицы констант, используемой в преобразовании *InvMixColumns()*. Назовем новое превращение *InvAddRoundKey()*. Рис. 8.32 показывает новую конфигурацию.

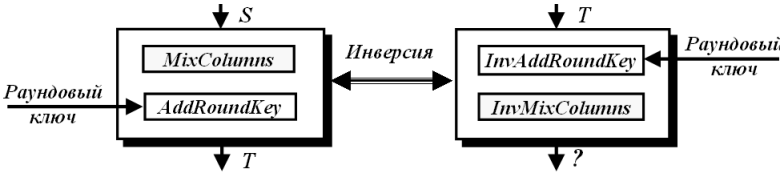


Рис. 8.32. Замена порядка выполнения преобразований *MixColumns()* и *InvAddRoundKey()*, а также *AddRoundKey()* и *InvMixColumns()*

Можно показать, что эти два преобразования *MixColumns()* и *AddRoundKey()* теперь инверсные друг другу.

При зашифрувании входную матрицу состояния обозначим как *S*, а выходную – *T*, а при расшифровании входную матрицу состоя-

ния обозначим *T*. Покажем, что в таком случае исходная матрица состояния – *S*.

Необходимо обратить внимание на то, что преобразование *MixColumns()* – фактически результат умножения матрицы констант *C* на матрицу состояния *S*. Итак, результат зашифрования – матрица состояния *T*

$$T = C \bullet S \oplus K.$$

При расшифровании

$$C^{-1} \cdot T \oplus C^{-1} \cdot K = C^{-1} \cdot (C \cdot S \oplus K) \oplus C^{-1} \cdot K =$$

$$= C^{-1} \cdot C \cdot S \oplus C^{-1} \cdot K \oplus C^{-1} \cdot K = S,$$

поскольку $C^{-1} \cdot C \cdot S = I \cdot S = S$, а $C^{-1} \cdot K \oplus C^{-1} \cdot K = 0$. Здесь *I* – единичная матрица.

С учетом (8.36) получим прямое (эквивалентное) расшифрование данных, показанную на рис. 8.33.

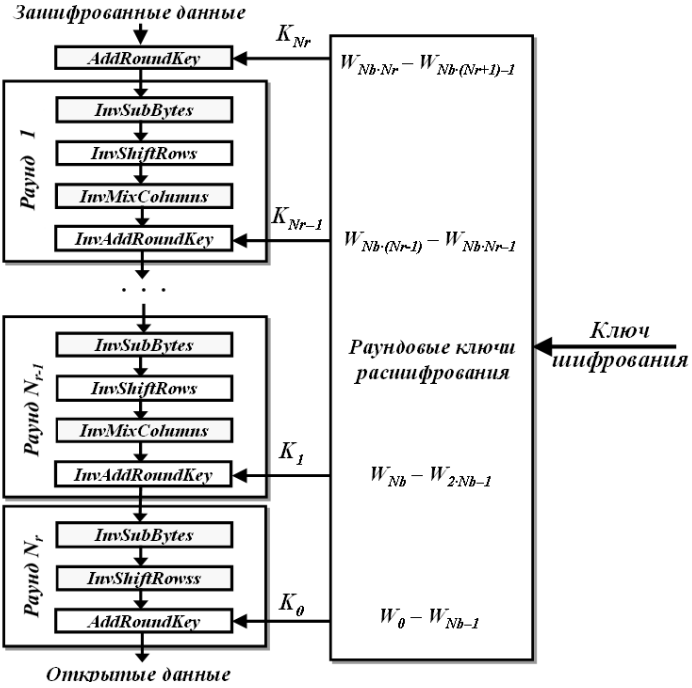


Рис. 8.33. Схема процедуры прямого (эквивалентного) расшифрования в криптографическом алгоритме *AES* (*Rijndael*)

Следует обратить внимание, что необходимо использовать два преобразования *AddRoundKey()* и девять *InvAddRoundKey()*, как это показано на рис. 8.33.

Если в схеме на рис. 8.33 раундовые ключи от K_1 до K_{Nr-1} превратить с помощью функции *InvMixColumns()*, то есть провести вычисления с использованием выражения:

$$K_i^* = InvMixColumns(K_i),\ i = 1 \ldots Nr-1, \qquad (8.37)$$

то для расшифрования можно применить такой же порядок преобразований как при зашифровании, но им обратный.

В таком случае процессы при зашифровании и прямом расшифровании криптографического алгоритма *AES* (*Rijndael*) можно показать на рис. 8.34.

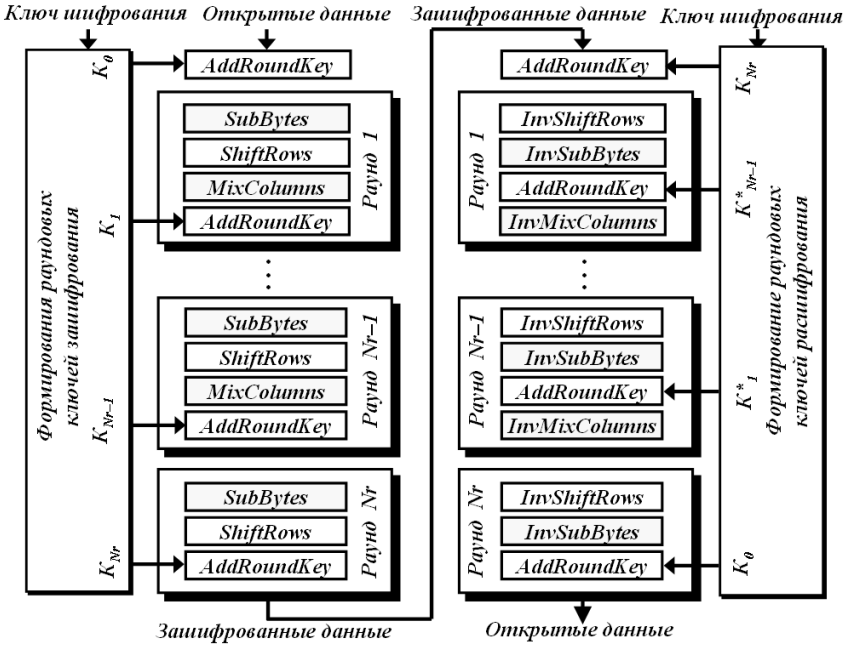


Рис. 8.34. Пояснение процессов зашифрования и прямого расшифрования в криптографическом алгоритме *AES* (*Rijndael*)

Ниже приведен псевдокод программы (*EqDeCipher*), которая показывает процесс прямого (эквивалентного) расшифрования данных.

```
//=============================================
//=== Процедура прямого расшифрования в псевдокоде ===
//=============================================
EqDeCipher (byte in[4*Nb], byte out[4*Nb], word dw[Nb*(Nr+1)])
begin
    byte state[4,Nb]
    state = in
    AddRoundKey(state, dw[Nr*Nb, (Nr+1)*Nb-1])
    for round = Nr-1 step -1 downto 1
        InvSubBytes(state)
        InvShiftRows(state)
        InvMixColumns(state)
        AddRoundKey(state, dw[round*Nb, (round+1)*Nb-1])
    end for
    InvSubBytes(state)
    InvShiftRows(state)
    AddRoundKey(state, dw[0, Nb-1])
    out = state
end
//=============================================
```

Для формирования развернутого ключа расшифрования с учетом (8.37), к процедуре развертывания ключа необходимо добавить такой код

```
//=============================================
EqKeyExpansion (byte key[4*Nk], word w[Nb*(Nr + 1)], Nk)
begin
    for i = 0 step 1 to (Nr+1)*Nb-1
        dw[i] = w[i]
    end for
    for round = 1 step 1 to Nr-1
        InvMixColumns(dw[round*Nb, (round+1)*Nb-1])
    end for
//=============================================
```

Примечание. В последнем операторе (в функции *InvMixColumn()*) происходит преобразование типа данных, поскольку развернутый ключ сохраняется в виде линейного массива *32*-разрядных слов, тогда как входной параметр функции – двумерный массив байтов.

В табл. 8.8 приведены процедуры зашифрования, а также два эквивалентных варианта процедуры расшифрования при использовании двухраундового варианта *AES* (*Rijndael*).

Таблица 8.8

Последовательность преобразований в двухраундовом варианте *AES* (*Rijndael*)

Преобразования для двухраундового варианта *AES* (*Rijndael*)		
зашифрование	обратное расшифрование	прямое расшифрование
AddRoundKey	*AddRoundKey*	*AddRoundKey*
SubBytes	*InvShiftRows*	*InvSubBytes*
ShiftRows	*InvSubBytes*	*InvShiftRows*
MixColumns	*AddRoundKey*	*InvMixColumns*
AddRoundKey	*InvMixColumns*	*AddRoundKey*
SubBytes	*InvShiftRows*	*InvSubBytes*
ShiftRows	*InvSubBytes*	*InvShiftRows*
AddRoundKey	*AddRoundKey*	*AddRoundKey*

Первый вариант функции расшифрования (обратного) – обычная инверсия функции зашифрования. Второй вариант функции расшифрования (прямого) полученный из первого после изменения порядка следования преобразования в трех парах: *InvShiftRows – InvSubBytes* (дважды) и *AddRoundKey – InvMixColumns*.

Очевидно, что результат преобразования при переходе от начальной к обратной последовательности выполнения операций, в указанных парах не изменится.

Видно, что процедура зашифрования и вариант процедуры прямого расшифрования совпадают с точностью до порядка использования раундовых ключей (в ходе выполнения операций *AddRoundKey()*), таблиц замен (в ходе выполнения операций *SubBytes()* и *InvSubBytes()*) и матриц преобразования (в ходе выполнения операций *MixColumns()* и *InvMixColumns()*). Этот результат так легко обобщить на любое другое число раундов [38].

8.9. БЕЗОПАСНОСТЬ AES

8.9.1. Свойства симметричности и слабые ключи

Учитывая значительную степень присущей шифру симметричности (однородности) в работе с данными, были предприняты специальные меры, позволившие уменьшить ее влияние. Это достигается благодаря различным для каждого раунда константам (см.

обоснование выбора алгоритма разворачивания ключа). Тот факт, что алгоритмы зашифрования и расшифрования состоят из различных операций-функций, практически исключает возможность существования слабых и полуслабых ключей шифрования (что имело место в *DES*). Нелинейность процедуры разворачивания ключа также практически исключает возможность получения эквивалентных раундовых ключей после разворачивания различных начальных ключей шифрования.

8.9.2. Дифференциальный и линейный криптографические анализы

Дифференциальный криптографический анализ, как уже говорилось, был впервые описан *Э. Бихамом* и *А. Шамиром*. Линейный криптографический анализ был впервые описан *М. Мацуи*.

Дифференциальный криптографический анализ

Дифференциальный криптографический анализ – это атака на шифр, базирующаяся на возможности свободного подбора открытых данных для последующего их зашифрования. Анализу подвергаются зависимости между парами блоков данных до и после применения шифра. *ДК*-атаки становятся возможны тогда, когда можно установить проникновение таких зависимостей сквозь почти все (за исключением *2–3*) раунды шифрующего преобразования. При этом относительное количество фиксированных пар бит данных (назовем это число коэффициентом проникновения), для которых можно установить зависимость различий на выходе от различий на входе при зашифровании, должно быть значительно выше, чем 2^{1-n}, где n – длина входных данных в битах. Попробуем объяснить этот критерий успешности дифференциального криптографического анализа.

Представим себе, что можем произвольно выбирать два различных блока открытых данных длиной по n бит каждый для зашифрования и последующего анализа двух полученных блоков зашифрованных данных. Напоминаем, будем искать пары бит во входных данных, различия между которыми подсказали бы, каково будет различие между битами какой-то определенной пары бит в выходных данных.

Таким образом, выбираем по одному определенному биту из каждого из двух блоков открытых данных и анализируем результа-

ты зашифрования при всех возможных значениях входных данных. Поскольку теперь в каждом блоке осталось $n–1$ битов, которые могут принимать различные значения, то нам нужно проанализировать всего 2^{n-1} значений каждого блока входных данных. И если обнаружится, что в выходных данных существует пара бит такая, что различия между этими битами можно предсказать, исходя из различий между битами выбранной пары во входных данных, и это событие не случайно (т.е. его вероятность больше, чем $1/2^{n-1} = 2^{1-n}$), то можно считать, что преобразование зашифрования имеет явную слабость и может быть подвергнуто дифференциальному криптографическому анализу.

Раунд зашифрования при дифференциальном криптографическом анализе удобно рассматривать как отдельную разностную сессию преобразования данных со своими входными и выходными данными. Полная разностная сессия, таким образом, состоит из отдельных сессий – раундов зашифрования, а полный коэффициент проникновения при зашифровании является произведением коэффициентов проникновения составляющих сессий.

Таким образом, для того чтобы обеспечить стойкость к *ДК*-атакам, достаточно чтобы коэффициент проникновения после некоторого числа раундов стал не больше 2^{1-n}. Далее будет приведено доказательство того, что уже после четырех раундов преобразования коэффициент проникновения становится не больше, чем 2^{-150} (а после 8 раундов – менее 2^{-300}), что учитывая данные табл. 8.1, доказывает стойкость шифра при различных n.

Линейный криптографический анализ

Линейный криптографический анализ также применяется в атаках с возможностью подбора открытых данных для их последующего зашифрования. Он основан на анализе линейных зависимостей между битами входных данных, которые обуславливают вполне определенные зависимости между битами выходных данных.

Пусть $A[i_1, i_2, i_3, \ldots, i_n]$ – сумма по модулю 2 бит массива данных A, индексами которых являются значения $i_1, i_2, i_3, \ldots, i_n$, то есть

$$A[i_1, i_2, i_3, \ldots, i_n] = A[i_1] \oplus A[i_2] \oplus A[i_3] \oplus \ldots \oplus A[i_n].$$

Тогда линейной зависимостью будем называть выражение вида

$$P[i_1, i_2, i_3, \ldots, i_n] = C[i_1, i_2, i_3, \ldots, i_n] \oplus K[i_1, i_2, i_3, \ldots, i_n].$$

где *P*, *C* и *K* означают, соответственно, массивы бит входных данных, выходных данных и ключа.

Таким образом, если для достаточно большого количества различных *P* и *C* удается подобрать *K*, удовлетворяющее данному выражению, то можно говорить о раскрытии одного бита информации о ключе на основании существования линейной зависимости входных и выходных данных. Существуют также методы определения более чем одного бита информации о ключе на основании анализа линейных зависимостей.

ЛК-атаки становятся эффективны тогда, когда можно установить существование таких линейных зависимостей (иначе – корреляций) после почти всех (за исключением *2–3*) раундов шифрующего преобразования. При этом относительное число (коэффициент) таких корреляций в данных должно быть значительно выше чем $2^{-n/2}$ (то есть более, чем одна пара бит). Корреляции имеют знак, то есть могут быть отрицательными или положительными, в зависимости от того, совпадает ли зависимость во входных данных с зависимостью в выходных или же обратна ей. Это дает ключ (игра слов) к нахождению бит раундового ключа. Считается, что общая корреляция при шифровании является композицией (или суммой) всех корреляций, наблюдаемых в каждой из отдельных линейных сессий (т.е. в каждом раунде преобразования), где есть предсказуемые комбинации бит во входных и выходных данных.

Достаточным условием стойкости к *ЛК*-атаке является отсутствие каких-либо линейных сессий с коэффициентом общей корреляции выше, чем $2^{-n/2}$. Далее будет приведено доказательство того, что уже после четырех раундов преобразования коэффициент корреляции становится меньше, чем 2^{-75} (а после 8 раундов меньше 2^{-150}).

Эффективность разностных и линейных сессий

В [38] показано, что:

- полный коэффициент проникновения разностной сессии может быть аппроксимирован к произведению коэффициентов проникновения всех активных в данной разностной сессии таблиц замен *S*-блоков;

- общая корреляция линейной сессии может быть аппроксимирована к произведению коэффициентов корреляции между входными и выходными данными всех активных в данной линейной сессии таблиц замен *S*-блоков.

Таким образом, стратегия сессии преобразования любого шифра должна заключаться в следующем:

- выбрать *S*-блок такой, что максимальный коэффициент проникновения и максимальный коэффициент корреляции между входными и выходными данными были бы как можно меньше; для *S*-блоков *AES* (*Rijndael*) они равны соответственно 2^{-6} и 2^{-3};

- сконструировать рассеивание таким образом, чтобы не было многораундовых преобразований с малым числом активных *S*-блоков.

Далее будет показано, что минимальное число активных *S*-блоков в любой *4*-раундовой разностной или линейной сессии равно *25*. Это дает коэффициент проникновения 2^{-150} для любой *4*-раундовой разностной сессии и самое большее 2^{-75} для коэффициента корреляции любой *4*-раундовой линейной сессии. Это верно для всех размеров блоков шифра и не зависит от значения раундового ключа.

Примечание. Нелинейность *S*-блока, выбранного наугад из набора возможных обратимых *8*-разрядных блоков замены, будет, скорее всего, меньше оптимальной. Типичны значения от 2^{-5} до 2^{-4} для максимального коэффициента проникновения и 2^{-2} для коэффициента корреляции между входными и выходными данными таблиц замены [38].

Проникновение образов активности

Для дифференциального криптографического анализа количество активных *S*-блоков в одном раунде определяется количеством ненулевых байт, задающих различия во входных данных раунда (количество ненулевых байт после применения операции *xor* к двум массивам входных данных). Пусть образ (*pattern*) массива *State*, определяющий позиции активных *S*-блоков, обозначается термином "*разностный образ активности*", и пусть *разностный байтовый вес* обозначает число активных (ненулевых) байт в образе.

Для линейного криптографического анализа количество активных *S*-блоков определяется количеством ненулевых байт в массиве входных данных. Пусть образ, определяющий позиции активных *S*-блоков, обозначается термином "*корреляционный образ активно-*

сти", и пусть *корреляционный байтовый вес Z(a)* будет количеством активных (ненулевых) байт в образе *a*. Более того, пусть столбец в *образе активности* называется *активным*, если он содержит хотя бы один активный байт. Пусть $Z_c(a)$ – количество активных столбцов в *a*, т.е., по сути, вес столбца этого образа. Байтовый вес столбца *j* образа *a*, обозначаемый $Z_j(a)$, задает количество активных байт в этом столбце.

Общий вес сессии равен сумме весов образов активности входных данных для каждого раунда, входящего в состав этой сессии.

Можно рассматривать проникновение разностных (линейных) образов активности сквозь раунды преобразования как проникновение зависимостей входных/выходных данных в разностной (линейной) сессии. Это проиллюстрировано на рис. 8.35, где темным цветом выделены активные байты.

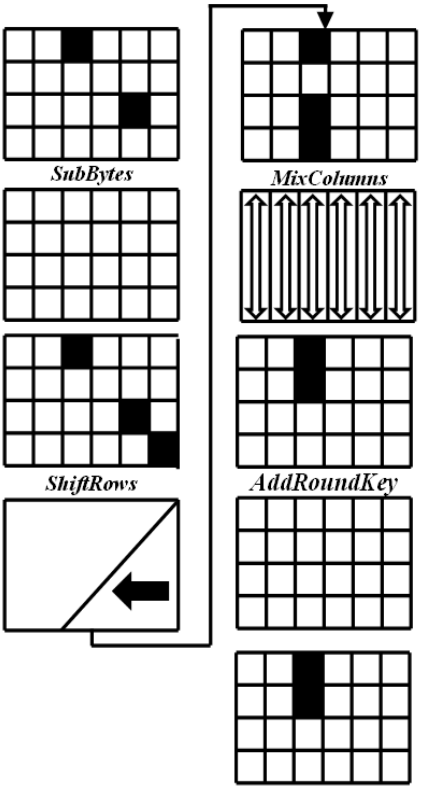


Рис. 8.35. Проникновение образов активности через одноraундовое преобразование

SubBytes() и *AddRound Key()*. Образы активности, байтовый и столбцовый веса не меняются.

ShiftRows(). Байтовый вес не изменяется, поскольку значения самих байт вообще не меняются.

MixColumns(). Поскольку столбцы преобразуются каждый по отдельности, столбцовый вес не меняется.

Таким образом, функции *SubBytes()* и *AddRoundKey()* не играют никакой роли в проникновении образов активности, и поэтому в рассмотрении эффект раунда преобразований может быть сведен к эффекту от функций *ShiftRows()* и *MixColumns()*.

В дальнейшем *SubBytes()* и *AddRoundKey()* в расчет браться не будут.

Функция *MixColumns()*, как было отмечено выше, имеет коэффициент распространения, равный *5*. Это означает, что для любого активного столбца из образа входных (или выходных) данных, сумма байтовых весов на входе и выходе раунда будет ограничена снизу значением *5*.

Функция, в свою очередь, имеет следующие свойства:

- вес столбцов выходных данных ограничен снизу максимальным байтовым весом столбца из входных данных;
- вес столбцов входных данных ограничен снизу максимальным байтовым весом столбца из выходных данных.

Итак, пусть далее в нашем описании образ активности на входе *i*-го раунда обозначается как a_{i-1}, а после приложения функции *ShiftRows()* обозначается как b_{i-1}. Нумерация раундов начинается с единицы, поэтому начальный образ активности имеет обозначение a_0. Тогда a_i и b_i разделены лишь функцией *ShiftRows()* и имеют один и тот же байтовый вес, а b_{i-1} и a_i разделены лишь функцией *MixColumns()* и имеют одинаковый столбцовый вес. Общий вес *m*-раундовой сессии равен сумме весов образов от a_0 до a_{m-1}. Свойства проникновения образов активности проиллюстрированы на рис. 8.36, где темным цветом выделены активные байты, а серым – активные столбцы.

Теорема 8.1. Общий байтовый вес двухраундовой сессии с *Q* активными столбцами на входе второго раунда ограничен снизу значением *5Q*.

Доказательство. Тот факт, что функция *MixColumns()* имеет коэффициент распространения, равный *5*, означает, что сумма байтовых весов в столбцах образов b_0 и a_1 для каждого столбца ограничена снизу значением *5*. Поэтому, если столбцовый вес a_1 равен *Q*, то это дает ограничение снизу *5Q* для суммы байтовых весов b_0 и a_1. Поскольку a_0 и b_0 имеют одинаковый байтовый вес, это ограничение действительно и для суммы байтовых весов a_0 и a_1, что и требовалось доказать. Иллюстрация теоремы 8.1 показана на рис. 8.37.

Отсюда следует, что любая двухраундовая сессия имеет как минимум *5* активных *S*-блоков.

Лемма 8.1. В двухраундовой сессии сумма активных столбцов во входных данных и активных столбцов в выходных данных не меньше *5*. Другими словами сумма столбцовых весов образов a_0 и a_2 не меньше *5*.

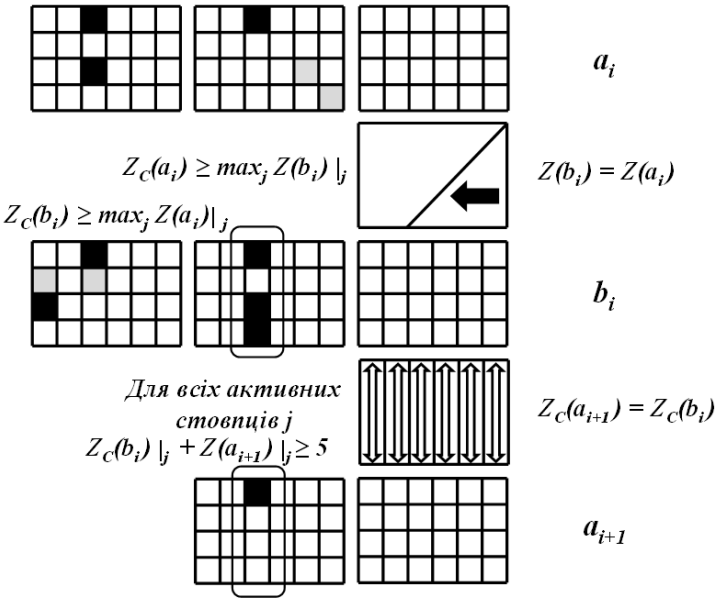


Рис. 8.36. Проникновения образов в преобразованиях одного раунда

Доказательство. Функция *ShiftRows()* сдвигает байты любого столбца из образа a_i в различные столбцы образа b_i и наоборот. Отсюда следует, что столбцовый вес образа a_i ограничен снизу байтовым весом отдельных столбцов b_i. Так же и столбцовый вес образа b_i ограничен снизу байтовым весом отдельных столбцов образа a_i.

В сессии активен как минимум один столбец образа a_1 (или, что тоже самое, образа b_0). Обозначим этот столбец как столбец g. Поскольку коэффициент распространения функции *MixColumns()* равен 5, то сумма байтовых весов столбца g в образе b_0 и в образе a_1 будет не менее 5. Но столбцовый вес образа a_0 ограничен снизу байтовым весом столбца g образа b_0. А столбцовый вес образа b_1 ограничен снизу байтовым весом столбца g образа a_1. Следовательно, сумма столбцовых весов образов a_0 и b_1 ограничена снизу значением 5. Поскольку столбцовый вес образа a_2 равен столбцовому весу образа b_1 то лемму можно считать доказанной. Иллюстрация леммы 8.1 приведена на рис. 8.38.

Теорема 8.2. Любая сессия, состоящая из 4 раундов, имеет как минимум 25 активных байт.

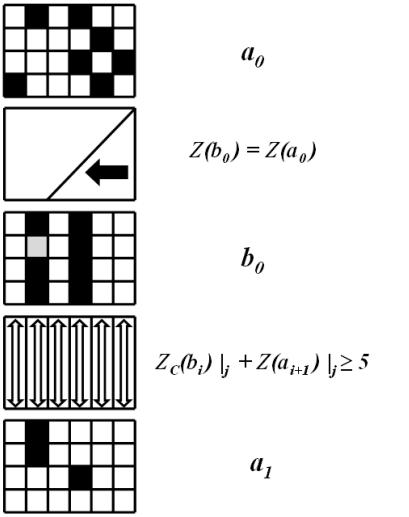


Рис. 8.37. Иллюстрация теоремы 8.1 при $Q = 2$

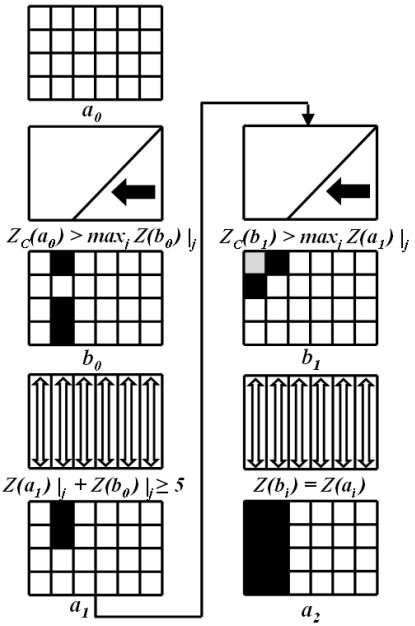


Рис. 8.38 Иллюстрация леммы 8.1 с одним активным столбцом в образе a_1

Доказательство. Общий байтовый вес *4*-раундовой сессии равен сумме байтовых весов $Z(a_0)$, $Z(a_1)$, $Z(a_2)$ и $Z(a_3)$. Применив теорему 8.1 к первым двум раундам (*1* и *2* на рис. 8.39) и к последним двум раундам (*3* и *4* на рис. 8.39), получаем, что общий байтовый вес *4*-раундовой сессии ограничен снизу произведением суммы столбцовых весов образов a_1 и a_3 на *5*. А согласно лемме 8.1 сумма столбцовых весов для образов a_1 и a_3 не меньше *5*.

Отсюда следует, что байтовый вес *4*-раундовой сессии ограничен снизу значением *25*. Теорема 8.2 проиллюстрирована на рис. 8.39.

8.9.3. Атака методом сокращенных дифференциалов

Концепция сокращенных дифференциалов была впервые опубликована в работе *Л. Кнудсена* [7, 20]. Атаки этого класса основаны на том, что для некоторых шифров разностные сессии имеют тенденцию кластеризации [38].

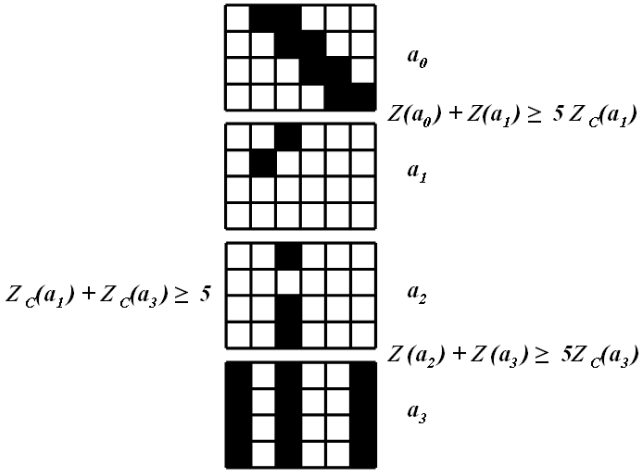


Рис. 8.39. Иллюстрация теоремы 8.2

Это означает, что для некоторых комбинаций линейных зависимостей во входных и выходных данных количество раундов, сквозь которые можно отследить значительное число таких зависимостей, становится слишком большим. Другими словами эффективность дифференциального криптографического анализа для таких комбинаций данных резко возрастает. Шифры, у которых все преобразования выполняются над выровненными блоками массива входных данных, имеют тенденцию быть особенно подверженными такого рода атакам. Поскольку *AES* (*RIJNDAEL*) как раз обладает тем свойством, что его функции преобразования оперируют целыми байтами, то его подверженность этому виду атак была специально исследована. Установлено, что для числа раундов, большего *6*, не существует более эффективной атаки, чем полный перебор по всему ключевому пространству.

8.9.4. Атака “Квадрат”

Атака “*Квадрат*” была специально разработана для одноименного шифра *Square* (авторы *Й. Демен* (*J. Daemen*), *Л. Кнудсен* (*L. Knudsen*), *В. Раймєн* (*V. Rijmen*)). Атака использует при своем проведении байт-ориентированную структуру шифра. Описание ее было опубликовано вместе с самим шифром в работе [38]. Учиты-

вая, что *AES* (*Rijndael*) унаследовал многие свойства шифра *Square*, эта атака применима и к нему. Далее приведено описание атаки "*Квадрат*" применительно к *AES* (*Rijndael*).

Атака "*Квадрат*" основана на возможности свободного подбора атакующим некоторого набора открытых данных для последующего их зашифрования. Она независима от таблиц замен *S*-блоков, многочлена функции *MixColumns()* и способа разворачивания ключа. Эта атака для *6*-раундового шифра *AES* (*Rijndael*), состоящего из *6* раундов, эффективнее, чем полный перебор по всему ключевому пространству. После описания базовой атаки на *4*-раундовый *AES* (*Rijndael*), 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 λ-набор – такой набор из *256* входных блоков (массивов *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 λ-набора. То есть для всех x и y:

$$x,y \in \lambda : \begin{cases} x_{i,j} \neq y_{i,j}, & \text{если байт с номером } ij \text{ активный} \\ x_{i,j} = y_{i,j}, & \text{в противном случае} \end{cases}$$

Будучи подвергнутыми обработке функциями *SubBytes()* и *AddRoundKey()* блоки λ-набора дадут в результате другой λ-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()* λ-набор в общем случае необязательно останется λ-набором (т.е. результат преобразования может перестать удовлетворять определению λ-набора). Но поскольку каждый байт результата функции *MixColumns()* является линейной комбинацией (с обратимыми коэффициентами) четырех входных байт того же столбца

$$b_{ij} = 2 \cdot a_{ij} \oplus 3 \cdot a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j},$$

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 λ-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 λ-набором до того самого момента, когда он поступает на вход функции *MixColumns()* третьего раунда.

Основное свойство λ-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 входного 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:

$$\bigoplus_{b=MixColumns(a),\,a\in\lambda} b_{ij} = \bigoplus_{a\in\lambda} (2a_{ij} \oplus 3a_{(i+1)j} \oplus a_{(i+2)j} \oplus a_{(i+3)j} =$$

$$= 2 \bigoplus_{a\in\lambda} a_{ij} \oplus 3 \bigoplus_{a\in\lambda} a_{(i+1)j} \oplus \bigoplus_{a\in\lambda} a_{(i+2)j} \oplus \bigoplus_{a\in\lambda} a_{(i+3)j}\,.$$

Таким образом, все данные на входе четвертого раунда сбалансированы (т.е. их полная сумма равна нулю). Этот баланс в общем случае нарушается последующим преобразованием данных функцией *SubBytes()*.

Предположим 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:

$$a_{ij} = SubBytes(b_{ij}) \oplus k_{ij} .$$

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, 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 шестого раунда в начало базовой атаки "Квадрат"

Основная идея заключается в том, чтобы подобрать такой набор блоков открытых данных, который на выходе после первого раунда давал бы λ-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 такие:

$$b_i = 2 \cdot a_i \oplus 3 \cdot a_{i+1} \oplus a_{i+2} \oplus a_{i+3}, \quad 0 \le i \le 3,$$

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 приме-

нить 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:

$$b_{ij} = 2 \cdot SubBytes(a_{ij} \oplus k_{ij}) \oplus 3 \cdot SubBytes(a_{(i+1)(j+1)} \oplus$$

$$\oplus\ k_{(i+1)(j+1)}) \oplus SubBytes(a_{(i+2)(j+2)} \oplus k_{(i+2)(j+2)}) \oplus$$

$$\oplus\ SubBytes(a_{(i+3)(j+3)} \oplus k_{(i+3)(j+3)}), \quad 0 \le i, j \le 3.$$

Таким образом, получаем следующий алгоритм взлома. Имеем всего 2^{32} различных значений a 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 λ-набором после первого раунда. К этому λ-набору 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 взламывание 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.

Эффективность и требования к памяти

Таким образом атака "*Квадрат*" может быть применена к *6* раундам шифра *AES* (*Rijndael*), являясь при этом более эффективной, чем полный перебор по всему ключевому пространству. Трудоемкость и требования к памяти для атаки "*Квадрат*" обобщены в табл. 8.9 [38]. Любое известное продолжение атаки "*Квадрат*" на *7* и более раундов становится более трудоемким, чем даже обычный полный перебор значений ключа.

Таблица 8.9

Ресурсы, необходимые для проведения атаки “*Квадрат*”

Тип атаки	*Необходимое количество блоков открытых данных*	*Число повторений процедуры зашифрования*	*Требования к памяти*
Базовая *4*-раундовая атака	29	29	Небольшие
Расширенная, с добавлением раунда в конце	211	240	Небольшие
Расширенная, с добавлением раунда в начале	232	240	232
Расширенная, с добавлением раундов в начале и в конце	232	272	232

8.9.5. Атака методом интерполяций

В работе [7, 20, 38] *Т. Джéфферсон* и *Л. Кнудсен* представили новый вид атаки на блочные шифры. Суть ее состоит в том, что атакующий пытается получить многочлен из известных ему пар входных/выходных данных. Это становится возможным лишь в тех случаях, когда функции, входящие в состав преобразования, могут быть достаточно компактно выражены алгебраически, а все преобразование – представлено в виде алгебраического выражения разрешимой сложности.

Атака основывается на следующем факте: если полученный многочлен (или рациональное выражение) имеет небольшую степень, то достаточно нескольких пар входных/выходных данных для того, чтобы подобрать коэффициенты многочлена, значения которых прямо связаны со значением ключа. Сложность алгебраического представления подстановки *SubBytes()* в поле $GF(2^8)$ в сочетании с рассеиванием функцией *MixColumns()* делает невозможным применение этого типа атак к многораундовому шифру *AES* (*Rijndael*). К примеру, алгебраическое выражение для функции *SubBytes()*:

$$\{63\} + \{8f\}\cdot x^{127} + \{b5\}\cdot x^{191} + \{01\}\cdot x^{223} + \{f4\}\cdot x^{239} +$$

$$+ \{25\}\cdot x^{247} + \{f9\}\cdot x^{251} + \{09\}\cdot x^{253} + \{05\}\cdot x^{259}.$$

8.9.6. О существовании слабых ключей

Слабыми считаются ключи, использование которых ведет к плохому преобразованию входных данных (недостаточное рассеивание или перемешивание). Наиболее известный случай существования слабых ключей для шифра *IDEA* описан в [12, 29].

Как правило, такой недостаток свойственен шифрам, в которых имеются нелинейные операции, зависящие от значения ключа. Это не так в случае *AES* (*Rijndael*) – добавление ключа в процедуру шифрования осуществляется с использованием операции *xor*, а нелинейное преобразования реализуют *S*-блоки с фиксированными таблицами замен. Таким образом, в *AES* (*Rijndael*) отсутствуют ограничения на выбор ключей.

8.9.7. Атака “эквивалентных ключей”

В [38] *Э. Бихэм* описал *атаку* “*эквивалентных ключей*”. Позже *Д. Келси*, *Б. Шнайер* и *Д. Вагнер* показали в работе [20, 47], что некоторые шифры являются недостаточно стойкими к этой атаке.

В *атаке* “*эквивалентных ключей*” криптографический аналитик выполняет шифрование, используя различные (полностью или частично неизвестные) ключи, которые имеют заданные взаимозависимости. Учитывая высокую степень рассеивания при развертывании ключа, возможность этой атаки на *AES* (*Rijndael*) маловероятна.

Контрольные вопросы и задания

1. Перечислите параметры (размер блока, размер ключа и количество раундов) для трех версий *AES*.

2. Сколько превращений есть в каждой версии *AES*? Сколько раундовых ключей необходимо для каждой версии?

3. Сравните *DES* и *AES*. Который из них ориентирован на работу с битом, а какой – на работу с байтом?

4. Определите матрицу состояния в *AES*. Сколько матриц состояний есть в каждой версии *AES*?

5. Какие из четырех преобразований, определенных для *AES*, изменяют содержание байтов, а какие не изменяют?

6. Сравните подстановку в *DES* и *AES*. Почему в *AES* есть только одна таблица подстановки и несколько *в DES* (*S*-блоков)?

7. Сравните перестановки в *DES* и *AES*. Почему нужно иметь расширение и сжатие перестановки в *DES* и не нужно в *AES*?

8. Сравните ключи раунда в *DES* и *AES*. В каком шифре размер ключа раунда равен размеру блока?

9. Почему преобразования *MixColumns*, которое смешивает данные, необходимо в *AES*, но не требуется в *DES*?

10. Перечислите режимы работы для алгоритма *AES*.

11. Произвести сложение двух элементов конечного поля $x^7 + x^3 + x + 1$ и $x^6 + x^3 + x^2 + 1$.

12. Определить частное и остаток от деления элемента $x^{13}+x^{11}+x^9+x^7+x^5+x+1$ на элемент $x^6+x^3+x^2+1$ в конечном поле.

13. Выполнить сложение двух многочленов с коэффициентами конечного поля $GF(2^8)$ $\{f5\}\cdot x^7+\{03\}\cdot x^3+\{09\}\cdot x+\{02\}$ и $\{07\}\cdot x^7+\{1b\}\cdot x^5+\{1e\}\cdot x+\{1f\}$. Неприводимый многочлен $p(x) = x^8+x^4+x^3+x+1$.

14. Вычислить произведение двух многочленов с коэффициентами из конечного поля $GF(2^8)$ $\{02\}\cdot x^3+\{05\}\cdot x^2+\{03\}\cdot x+\{04\}$ і $\{03\}\cdot x^3+\{05\}\cdot x^2+\{04\}\cdot x+\{01\}$. Неприводимый многочлен $p(x) = x^8+x^4+x^3+x+1$.

15. Вычислить произведение двух многочленов с коэффициентами из конечного поля $GF(2^8)$ $\{02\}\cdot x^3+\{05\}\cdot x^2+\{03\}\cdot x+\{04\}$ і $\{03\}\cdot x^3+\{05\}\cdot x^2+\{04\}\cdot x+\{01\}$. Результат умножения должен быть представлен 4-байтовым словом, при этом использовать многочлен $m(x) = x^4+1$.

16. Определить аддитивную инверсию многочлена длиной в один байт из конечного поля $GF(2^8)$ $x^6+x^3+x^2+1$.

17. Определить мультипликативную инверсию многочлена длиной в один байт из конечного поля $GF(2^8)$ $x^6+x^3+x^2+1$. Неприводимый многочлен $p(x) = x^8+x^4+x^3+x+1$.

18. Определить результат аффинного преобразования многочлена длиною в один байт x^5+x^2+1 из конечного поля $GF(2^8)$ в алгоритме *AES*. Неприводимый многочлен $p(x) = x^8+x^4+x^3+x+1$.

19. Значения байтов матрицы состояния данных на входе функции *SubBytes()* криптографической системы *AES-128* в шестнадцатеричной системе равны: *a49c7ff2689f352b6b5bea43026a5049*. Определить значения байтов матрицы состояния данных на выходе функции *SubBytes()*.

20. Значения байтов матрицы состояния данных на входе функции *InvSubBytes()* криптографической системы *AES-128* в шестна-

дцатеричной системе равны: *f196db453b53027789d2de491a87397f*. Определить значения байтов матрицы состояния данных на выходе функции *InvSubBytes()*.

21. Значения байтов матрицы состояния данных на входе функции *ShiftRows()* криптографической системы *AES-128* в шестнадцатеричной системе равны: *49ded28945db96f17f39871a7702533b*. Определить значения байтов матрицы состояния данных на выходе функции *ShiftRows()*.

22. Значения байтов матрицы состояния данных на входе функции *InvShiftRows()* криптографической системы *AES-128* в шестнадцатеричной системе равны: *f196db453b53027789d2de491a87397f*. Определить значения байтов матрицы состояния данных на выходе функции *InvShiftRows()*.

23. Значения байтов матрицы состояния данных на входе функции *MixColumns()* криптографической системы *AES-128* в шестнадцатеричной системе равны: *49db873b453953897f02d2f177de961a*. Определить значения байтов матрицы состояния данных на выходе функции *MixColumns()*.

24. Значения байтов матрицы состояния данных на входе функции *InvMixColumns()* криптографической системы *AES-128* в шестнадцатеричной системе равны: *31adb22485c967caedd5e4a2ff9e4ac3*. Определить значения байтов матрицы состояния данных на выходе функции *InvMixColumns()*.

25. Значения байтов матрицы состояния данных на входе функции *AddRoundKey()* криптографической системы *AES-128* в шестнадцатеричной системе равны: *584dcaf11b4b5aacdbe7caa81b6db0e5*. Значения байтов матрицы состояния раундового ключа на входе функции *AddRoundKey()* – *f2c295f27a9bb9435935807a7359f67f*. Определить значения байтов матрицы состояния на выходе функции *AddRoundKey()*.

26. Значения байтов матрицы состояния раундового ключа криптографической системы *AES-128* для девятого раунда зашифрования в шестнадцатеричной системе равны: *ac7766f319fadc2128*

d12941575c006e. Определить значения байтов матрицы состояния раундового ключа для десятого раунда.

27. Значения первых шести слов развернутого ключа зашифрования *AES–192* в шестнадцатеричной системе равны: *w[0] = = 00010203*; *w[1] = 04050607*; *w[2] = 08090a0b*; *w[3] = 0c0d0e0f*; *w[4] = 10111213*; *w[5] = 14151617*. Определить значения следующих шести слов развернутого ключа зашифрования *AES–192*.

28. Значения первых восьми слов развернутого ключа зашифрования *AES–256* в шестнадцатеричной системе равны: *w[0] = = 00010203*; *w[1] = 04050607*; *w[2] = 08090a0b*; *w[3] = 0c0d0e0f*; *w[4] = 10111213*; *w[5] = 14151617*; *w[6] = 18191a1b*; *w[7] = = 1c1d1e1f*. Определить значения следующих восьми слов развернутого ключа зашифрования *AES–256*.

Раздел 9

ЕВРОПЕЙСКИЕ СТАНДАРТЫ ШИФРОВАНИЯ ДАННЫХ

9.1. ИСТОРИЯ РАЗРАБОТКИ ЕВРОПЕЙСКИХ СТАНДАРТОВ

До недавнего времени основным используемым криптографическим алгоритмом блочного симметричного шифрования, используемого в коммерческих и финансовых учреждениях, был *DES*. Однако скачок в увеличении вычислительных мощностях и появление ряда прогрессивных математических методов криптографического анализа показали несостоятельность алгоритма *DES* в обеспечении требуемого уровня безопасности. В первую очередь это связано с малой длинной ключа и блока, что позволяет производить обычный перебор на специализированных криптографических процессорах за достаточно короткое время. Требовалось найти достойную замену устаревающему *DES*. Первым решили эту проблему американцы, организовав конкурс *AES* в 1998 на новый американский стандарт шифрования. В ходе нескольких туров был выбран стандарт XX века *Rijndael*, который негласно задал новый стандарт для блочных симметричных криптографических алгоритмов. В частности, это использование блока не менее *128* бит длинной и длинами ключей не менее *128* бит.

В Европе в 2000-2002 проведен аналогичный конкурс *NESSIE* (*New European Schemes for Signatures, Integrity and Encryption - Новые европейские схемы для подписи, целостности и шифрование*), суть которого выбор криптографического европейского стандарта [29]. В связи с прогрессом в усовершенствовании методов криптографических аналитических атак, к кандидатам были предъявлены повышенные требования. Рассматривались три класса безопасности. К первому, самому высокому, относятся криптографические

алгоритмы с длиной блока не менее *128* и длиной ключа *256* бит; к нормальному – криптографические алгоритмы с длиной блока не менее *128* бит и длиной ключа не менее *128* бит; к третьему – удовлетворительному, криптографические алгоритмы с длиной блока *64* бит и длиной ключа *128* бит.

При отборе оценивались и требования, которые аналогичны требованиям предъявляемым к алгоритмам участвовавших в *AES*: защищенность алгоритмов от криптографических аналитических атак с учетом новых предложений; статистическая безопасность алгоритмов; надежность математической базы; вычислительная сложность (скорость) зашифрования/расшифрования; сложность программной, аппаратной и программно-аппаратной реализации.

В результате конкурса были отобраны криптографические алгоритмы [29]:

- удовлетворительный класс безопасности: *IDEA* (*Швейцария*); *Khazad* (*Бразилия* и *Бельгия*); *MISTY1* (*Япония*); *SAFER++64* (*Швейцария*);
- нормальный класс безопасности: *Camellia* (*Япония*); *SAFER++128* (*Япония*) и *RC6* (*Швеция* и *США*);
- высокий класс безопасности: *Shacal* (*Франция*).

Стоит сказать, что наравне с ними эксперты в качестве одного из финалистов рассматривали также стандарты шифрования *США AES* и *Triple DES*, которые не были заявлены на конкурс *NESSIE*.

Алгоритм *AES* стал новым стандартом *США* и де-факто – стандартом блочного симметричного шифрования почти во всем мире.

Хуже с европейскими стандартами: в *США* выбором *AES* занимался уполномоченный государственный институт *NIST*, а в Европе – научные организации (например, Католический университет *г. Лювен, Бельгия*) и крупные корпорации (например, *Siemens*), причем выводы экспертов носили лишь рекомендательный характер. Кроме того, конкурс *NESSIE* был чересчур широк – возникло впечатление, что огромное количество алгоритмов эксперты просто не успели досконально изучить за три года его проведения.

Несмотря на окончание конкурса *NESSIE* в феврале 2003 г., общеевропейских криптографических стандартов до сих пор нет.

Наиболее интересными с точки зрения построения из отобранных криптографических алгоритмов являются: *IDEA* и *Camellia*. Рассмотрим эти алгоритмы более подробно.

9.2. АЛГОРИТМ БЛОЧНОГО ШИФРОВАНИЯ ДАННЫХ IDEA

9.2.1. История создания алгоритма IDEA

Алгоритм *IDEA* является симметричным блочным шифром, запатентованным швейцарской фирмой *Ascom*.

Первая версия алгоритма *IDEA* была предложена в 1990 г., ее авторы – *Сюэцзя Лай* (*Xuejia Lai*) и *Джеймс Мэсси* (*James Massey*) из Швейцарского института *ETH Zürich* (по контракту с *Hasler Foundation*, которая позже влилась в *Ascom – Tech AG*) в качестве замены *DES* и назвали ее *PES* (англ. *Proposed Encryption Standard – предложенный стандарт шифрования*). Затем, после публикации работ *Бихама* и *Шамира* по дифференциальному криптографическому анализу *PES*, алгоритм был улучшен с целью усиления криптографической стойкости и назван *IPES* (англ. *Improved Proposed Encryption Standard – улучшенный предложенный стандарт шифрования*). Через год его переименовали в *IDEA*.

Пересмотренная версия алгоритма, усиленная средствами защиты от дифференциальных криптографических атак, была представлена в 1991 году и подробно описана в 1992 году.

IDEA является одним из нескольких симметричных криптографических алгоритмов, которыми первоначально предполагалось заменить *DES*.

9.2.2. Принципы построения алгоритма IDEA

Как и большинство других блочных шифров, алгоритм *IDEA* использует при шифровании процессы смешивания и рассеивания, причем все процессы легко реализуются аппаратными и программными средствами.

IDEA оперирует *64*-битовыми блоками данных. Несомненным достоинством алгоритма *IDEA* является то, что его ключ имеет длину *128* бит. Один и тот же алгоритм используется и для зашифрования, и для расшифрования.

Целью разработки *IDEA* было создание относительно стойкого криптографического алгоритма с достаточно простой реализацией.

Следующие характеристики *IDEA* характеризуют его криптографическую стойкость:

Длина блока: длина блока должна быть достаточной, чтобы скрыть все статистические характеристики исходного сообщения. С другой стороны, сложность реализации криптографической функции возрастает экспоненциально в соответствии с размером блока. Использование блока размером в *64* бита в 90-е годы означало достаточную силу. Более того, использование режима шифрования *CBC* говорит о дальнейшем усилении этого аспекта алгоритма.

Длина ключа: длина ключа должна быть достаточно большой для того, чтобы предотвратить возможность простого перебора ключа. При длине ключа *128* бит *IDEA* считается достаточно безопасным.

Конфузия: зашифрованные данные должны зависеть от ключа сложным и запутанным способом.

Диффузия: каждый бит исходных данных должен влиять на каждый бит зашифрованных данных. Распространение одного незашифрованного бита на большое количество зашифрованных битов скрывает статистическую структуру исходных данных. Определить, как статистические характеристики зашифрованных данных зависят от статистических характеристик исходных данных, должно быть непросто. *IDEA* с этой точки зрения является очень эффективным алгоритмом.

В *IDEA* два последних пункта выполняются с помощью трех операций. Это отличает его от *DES*, где все построено на использовании операции *xor* и маленьких нелинейных *S*-блоках замены.

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

1. Побитовое *xor*, обозначаемое как $\oplus$.

2. Сумма целых чисел по модулю 2^{16} (*65536*), при этом входы и выходы трактуются как беззнаковые *16*-битные целые. Эту операцию обозначают как $\boxplus$.

3. Умножение целых чисел по модулю $2^{16} + 1$ (*65537*), при этом входы и выходы трактуются как беззнаковые *16*-битные целые, за исключением того, что блок из одних нулей трактуется как 2^{17}. Эту операцию обозначают как $\otimes$.

Эти три операции являются несовместимыми в том смысле, что:

1. Не существует пары из трех операций, удовлетворяющих дистрибутивному закону. Например

$$a \otimes (b \boxplus c) \neq a \otimes b \boxplus a \otimes c.$$

2. Не существует пары из трех операций, удовлетворяющих ассоциативному закону. Например

$$a \boxplus (b \oplus c) \neq (a \boxplus b) \oplus c.$$

Использование комбинации из этих трех операций обеспечивает комплексную трансформацию входа, делая криптографический анализ более трудным, чем в таком алгоритме как *DES*, основанном исключительно на функции *xor*.

9.2.3. Структура алгоритма IDEA

Рассмотрим общую схему шифрования *IDEA* (см. рис. 9.1). Как и в любом алгоритме шифрования, здесь существует два входа: незашифрованный блок и ключ. В данном случае незашифрованный блок имеет длину *64* бита, ключ имеет длину *128* бит.

Алгоритм *IDEA* состоит из восьми раундов, за которыми следует завершающее преобразование. Алгоритм разделяет блок на четыре *16*-битных подблока. Каждый раунд получает на входе четыре *16*-битных подблока и создает четыре *16*-битных выходных подблока.

Завершающее преобразование так же получает на входе четыре *16*-битных подблока и создает на выходе четыре *16*-битных подблока.

Каждый раунд использует шесть *16*-битных раундовых ключей, завершающее преобразование использует четыре подключа, т.е. всего в алгоритме используется *52* подключа.

Одним из основных элементов алгоритма, обеспечивающих диффузию, является структура, называемая *MA* (умножение/сложение).

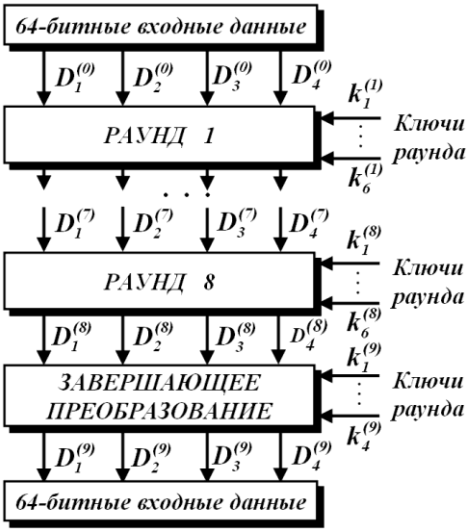


Рис. 9.1. Структура алгоритма *IDEA*

Структура устройства *MA* (умножение/сложение) показана на рис. 9.2.

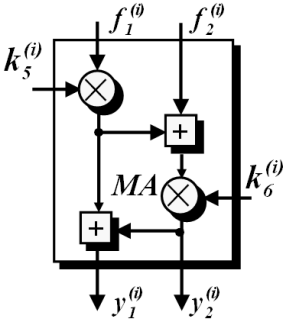


Рис. 9.2. Структура устройства *MA*

На вход этой структуры подаются два *16*-битных значения ($f_1^{(i)}$ и $f_2^{(i)}$) и два *16*-битных раундовых ключа ($k_1^{(i)}$ и $k_2^{(i)}$), на выходе создаются два *16*-битных значения ($y_1^{(i)}$ и $y_2^{(i)}$).

Исчерпывающая компьютерная проверка показывает, что каждый бит выхода этой структуры зависит от каждого бита входов незашифрованного блока данных и от каждого бита раундовых ключей. Данная структура повторяется в алгоритме восемь раз, обеспечивая высокоэффективную диффузию.

Рассмотрим последовательность преобразований отдельного раунда.

Раунд начинается с преобразования (см. рис. 9.3), которое комбинирует четыре входных подблока данных: $D_1^{(i)}$, $D_2^{(i)}$, $D_3^{(i)}$ и $D_4^{(i)}$ длинною каждый *16* бит с четырьмя раундовыми ключами: $k_1^{(i)}$, $k_2^{(i)}$, $k_3^{(i)}$ и $k_4^{(i)}$ длинною каждый *16* бит, используя операции сложения и умножения.

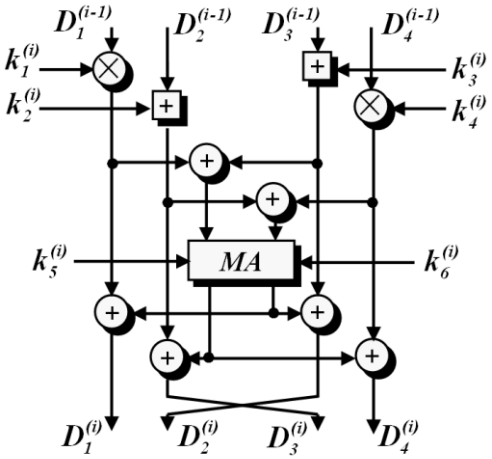


Рис. 9.3. Один раунд преобразования данных в *IDEA*

Четыре выходных блока этого преобразования комбинируются, используя операцию *xor* для формирования двух *16*-битных подблоков, которые являются входами *MA* структуры. Кроме того, *MA* структура имеет на входе еще два раундовых ключа и создает два *16*-битных подблока выхода.

В завершении четыре выходных подблока первого преобразования комбинируются с двумя выходными подблоками *MA* структуры, используя *xor* для создания четырех выходных подблоков данной итерации. Заметим, что два выхода, которые частично создаются вторым и третьим входами, меняются местами для создания второго и третьего выходов ($D_2^{(i)}$ и $D_3^{(i)}$). Это увеличивает перемешивание битов и делает алгоритм более стойким для дифференциального криптографического анализа.

Рассмотрим девятый раунд алгоритма, обозначенный как завершающее преобразование (см. рис. 9.4).

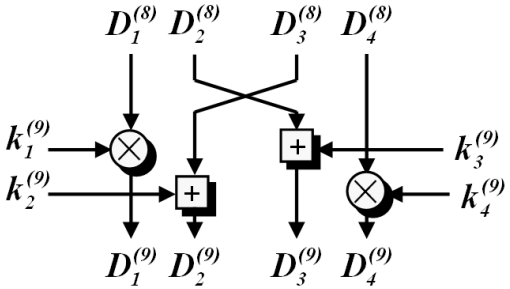


Рис. 9.4. Завершающее преобразование (раунд 9) в *IDEA*

Это та же структура, что была описана выше. Единственная разница состоит в том, что второй и третий входы меняются местами. Это сделано для того, чтобы расшифрование имело ту же структуру, что и зашифрование. Заметим, что девятая раунд (завершающее преобразование) требует только четыре входных раундовых ключа, в то время как для первых восьми раундов для каждой из них необходимо шесть входных раундовых ключей.

9.2.4. Генерация раундовых ключей для шифрования данных в IDEA

Генерация раундовых ключей для зашифрования данных в IDEA

Алгоритм разворачивания ключа определяет порядок получения раундовых ключей из начального ключа шифрования *K*. Раундовые ключи получаются из ключа шифрования посредством алгоритма выработки ключей. Он содержит два компонента:

- расширение ключа шифрования *K*;
- выбор раундовых ключей.

Основополагающие принципы алгоритма выглядят следующим образом:

- ключ шифрования *K* расширяется в расширенный ключ K_P;
- количество бит каждого раундового ключа равно длине подблока данных;
- количество раундовых ключей определяется из расчета шести ключей на каждый раунд зашифрования или расшифрования данных (*48* ключей, каждый длиной *16* бит) и четыре ключа на завершающее преобразование данных (каждый длиной *16* бит).

Всего должно быть сгенерировано пятьдесят два *16*-битных раундовых ключа.

Расширение ключа шифрования *K* осуществляется следующим образом. Ключ шифрования *K* делится на восемь частей по *16* бит каждая. В результате получаются первые восемь ключей для зашифрования данных, которые обозначим как k_1, k_2, ..., k_8, при этом ключ k_1 равен первым *16* битам ключа шифрования *K*, k_2 равен следующим *16* битам ключа *K* и т.д. Затем происходит циклический сдвиг ключа шифрования *K* влево на *25* битов. Полученная *128*-разрядная последовательность также делится на восемь частей по *16* бит каждая. В результате получаются вторые восемь ключей для зашифрования данных, которые обозначим как k_8, k_9, ..., k_{17}. Эта процедура повторяется до тех пор, пока не будут созданы *56* ключей. Последние четыре ключа (k_{53}, k_{54}, k_{55} и k_{56}) в шифре *IDEA* не используются и они просто отбрасываются.

Выбор раундовых ключей зашифрования осуществляется из расширенного ключа K_P следующим образом: в качестве ключей первого раунда $k_1^{(1)}$, $k_2^{(1)}$, $k_3^{(1)}$, $k_4^{(1)}$, $k_5^{(1)}$ и $k_6^{(1)}$ берутся первые шесть

ключей расширенного ключа K_P – k_1, k_2, k_3, k_4, k_5 и k_6 соответственно; в качестве ключей второго раунда $k_1^{(2)}$, $k_2^{(2)}$, $k_3^{(2)}$, $k_4^{(2)}$, $k_5^{(2)}$ и $k_6^{(2)}$ берутся вторые шесть ключей расширенного ключа K_P – k_7, k_8, k_9, k_{10}, k_{11} и k_{12} соответственно и т.д. В качестве ключей завершающего преобразования (девятого раунда) $k_1^{(9)}$, $k_2^{(9)}$, $k_3^{(9)}$ и $k_4^{(9)}$ берутся последние четыре ключа расширенного ключа K_P – k_{49}, k_{50}, k_{51} и k_{52} соответственно.

Процесс получения расширения ключа шифрования K_P из ключа шифрования K, а также выбора раундовых ключей показан на рис. 9.5 и сведен в табл. 9.1.

Рис. 9.5. Процесс получения раундовых ключей шифрования

Таблица 9.1

Ключи для каждого раунда шифрования *IDEA*

Номер раунда	*Номер ключа в раунде*					
	1	*2*	*3*	*4*	*5*	*6*
1	$k_1^{(1)}$	$k_2^{(1)}$	$k_3^{(1)}$	$k_4^{(1)}$	$k_5^{(1)}$	$k_6^{(1)}$
2	$k_1^{(2)}$	$k_2^{(2)}$	$k_3^{(2)}$	$k_4^{(2)}$	$k_5^{(2)}$	$k_6^{(2)}$
3	$k_1^{(3)}$	$k_2^{(3)}$	$k_3^{(3)}$	$k_4^{(3)}$	$k_5^{(3)}$	$k_6^{(3)}$
4	$k_1^{(4)}$	$k_2^{(4)}$	$k_3^{(4)}$	$k_4^{(4)}$	$k_5^{(4)}$	$k_6^{(4)}$

Окончание табл. 9.1

Номер раунда	*Номер ключа в раунде*					
	1	*2*	*3*	*4*	*5*	*6*
5	$k_1^{(5)}$	$k_2^{(5)}$	$k_3^{(5)}$	$k_4^{(5)}$	$k_5^{(5)}$	$k_6^{(5)}$
6	$k_1^{(6)}$	$k_2^{(6)}$	$k_3^{(6)}$	$k_4^{(6)}$	$k_5^{(6)}$	$k_6^{(6)}$
7	$k_1^{(7)}$	$k_2^{(7)}$	$k_3^{(7)}$	$k_4^{(7)}$	$k_5^{(7)}$	$k_6^{(7)}$
8	$k_1^{(8)}$	$k_2^{(8)}$	$k_3^{(8)}$	$k_4^{(8)}$	$k_5^{(8)}$	$k_6^{(8)}$
Завершающее преобразование (раунд *9*)	$k_1^{(9)}$	$k_2^{(9)}$	$k_3^{(9)}$	$k_4^{(9)}$		

Заметим, что каждый первый ключ раунда получен из своего подмножества битов ключа *K*. Если весь ключ обозначить как *K[1..128]*, то первыми ключами в восьми раундах будут ключи, включающие в себя биты подмножества битов ключа *K* (см. табл. 9.2).

Таблица 9.2

Значение первых ключей для каждого раунда шифрования *IDEA*

Номер раунда	*Ключи Расширенного ключа* K_P	*Ключ раунда*	*Биты ключа шифрования K*
1	k_1	$k_1^{(1)}$	*K[1...16]*
2	k_7	$k_1^{(2)}$	*K[97...112]*
3	k_{13}	$k_1^{(3)}$	*K[90...105]*
4	k_{19}	$k_1^{(4)}$	*K[83...98]*
5	k_{25}	$k_1^{(5)}$	*K[77...91]*
6	k_{31}	$k_1^{(6)}$	*K[44...59]*
7	k_{37}	$k_1^{(7)}$	*K[37...52]*
8	k_{43}	$k_1^{(8)}$	*K[30...45]*
9	k_{49}	$k_1^{(9)}$	*K[23...38]*

Хотя в каждом раунде за исключением первого и восьмого используются только *96* битов ключа шифрования *K*, множество битов этого ключа на каждой итерации не пересекаются, и не существует отношения простого сдвига между ключами разных раундов. Это происходит потому, что в каждом раунде используется только шесть раундовых ключей, в то время как при каждой ротации ключа получается восемь ключей расширенного ключа K_P.

Пример 9.1. Пусть ключ шифрования данных *K* равен:

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Необходимо определить значения раундовых ключей зашифрования данных.

Решение. Определим ключи расширенного ключа K_P путем разбиения ключа шифрования данных *K* на восемь частей и циклического сдвига битов ключа шифрования данных *K*. Данные сведем в табл. 9.3.

Таблица 9.3

Пример ключей для каждого раунда шифрования *IDEA*

Раунд	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$
1	*0001*	*0002*	*0003*	*0004*	*0005*	*0006*
2	*0007*	*0008*	*0400*	*0600*	*0800*	*0a00*
3	*0c00*	*0e00*	*1000*	*0200*	*0010*	*0014*
4	*0018*	*001c*	*0020*	*0004*	*0008*	*000c*
5	*2800*	*3000*	*3800*	*4000*	*0800*	*1000*
6	*1800*	*2000*	*0070*	*0080*	*0010*	*0020*
7	*0030*	*0040*	*0050*	*0060*	*0000*	*2000*
8	*4000*	*6000*	*8000*	*a000*	*c000*	*e001*
9	*0080*	*00c0*	*0100*	*0140*	-	-

Генерация раундовых ключей для расшифрования данных в IDEA

Метод вычисления, использующийся для расшифрования данных по существу такой же, как и при его зашифровании. Единственное отличие состоит в том, что для расшифрования используются другие раундовые ключи. В процессе расшифрования раундовые ключи должны использоваться в обратном порядке по отношению к процессу зашифрования.

Для формирования раундовых ключей расшифрования вначале формируются раундовые ключи зашифрования.

Процесс создания раундовых ключей расшифрования определяется следующим образом. Первый и четвёртый ключи *i*-го раунда расшифрования получаются из первого и четвёртого ключа (*10*-*i*)-го раунда зашифрования мультипликативной инверсией. Для *1*-го и *9*-го раундов второй и третий ключи расшифрования получаются из второго и третьего ключей *9*-го и *1*-го раундов зашифрования аддитивной инверсией. Для раундов со *2*-го по *8*-й второй и третий ключи расшифрования получаются из третьего и второго ключей с *8*-го по *2*-й раунды зашифрования аддитивной инверсией. Последние два ключа *i*-го раунда расшифрования равны последним двум ключам (*9*-*i*)-го раунда шифрования.

Мультипликативную инверсию раундового ключа *k* обозначим как

$$k^{-1} = \frac{1}{k}$$

и тогда

$$\frac{1}{k} \cdot k \bmod (2^{16} + 1) = 1. \qquad (9.1)$$

Так как $2^{16} + 1$ – простое число, то каждое целое не равное нулю *k* имеет уникальную мультипликативную инверсию по модулю $2^{16} + 1$.

Аддитивную инверсию раундового ключа *k* обозначим как $-k$ и тогда

$$(-k + \cdot k) \bmod (2^{16} + 1) = 0. \qquad (9.2)$$

Для реализации алгоритма *IDEA* было принято предположение, что нулевой субблок равен $2^{16} = -1$, при этом мультипликативная обратная величина *0* равна *0* [7, 29]. Вычисление значений мультипликативных обратных величин требует некоторых затрат, но это приходится делать только один раз для каждого ключа расшифрования.

В связи с указанным ключи расшифрования для разных раундов представлены в табл. 9.4.

Таблица 9.4

Ключи для каждого раунда расшифрования *IDEA*

Номер раунда	*Номер ключа в раунде*					
	1	*2*	*3*	*4*	*5*	*6*
1	$1/k_1^{(9)}$	$-k_2^{(9)}$	$-k_3^{(9)}$	$1/k_4^{(9)}$	$k_5^{(8)}$	$k_6^{(8)}$
2	$1/k_1^{(8)}$	$-k_3^{(8)}$	$-k_2^{(8)}$	$1/k_4^{(8)}$	$k_5^{(7)}$	$k_6^{(7)}$
3	$1/k_1^{(7)}$	$-k_3^{(7)}$	$-k_2^{(7)}$	$1/k_4^{(7)}$	$k_5^{(6)}$	$k_6^{(6)}$
4	$1/k_1^{(6)}$	$-k_3^{(6)}$	$-k_2^{(6)}$	$1/k_4^{(6)}$	$k_5^{(5)}$	$k_6^{(5)}$
5	$1/k_1^{(5)}$	$-k_3^{(5)}$	$-k_2^{(5)}$	$1/k_4^{(5)}$	$k_5^{(4)}$	$k_6^{(4)}$
6	$1/k_1^{(4)}$	$-k_3^{(4)}$	$-k_2^{(4)}$	$1/k_4^{(4)}$	$k_5^{(3)}$	$k_6^{(3)}$
7	$1/k_1^{(3)}$	$-k_3^{(3)}$	$-k_2^{(3)}$	$1/k_4^{(3)}$	$k_5^{(2)}$	$k_6^{(2)}$
8	$1/k_1^{(2)}$	$-k_3^{(2)}$	$-k_2^{(2)}$	$1/k_4^{(2)}$	$k_5^{(1)}$	$k_6^{(1)}$
9	$1/k_1^{(1)}$	$-k_2^{(1)}$	$-k_3^{(1)}$	$1/k_4^{(1)}$		

Пример 9.2. Пусть ключ шифрования данных *K* равен

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Необходимо определить значения раундовых ключей расшифрования данных.

Решение. Воспользуемся определенными значениями раундовых ключей зашифрования данных из примера 9.1, которые приведены в табл. 9.3. Определим значения раундовых ключей расшифрования данных с помощью табл. 9.4 и выражений (9.1) и (9.2). Полученные значения раундовых ключей расшифрования данных сведем в табл. 9.5.

Таблица 9.5

Пример ключей для каждого раунда шифрования *IDEA*

Раунд	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$
1	*fe01*	*ff40*	*ff00*	*659a*	*c000*	*e001*
2	*fffd*	*8000*	*a000*	*cccc*	*0000*	*2000*
3	*a556*	*ffb0*	*ffc0*	*52ab*	*0010*	*0020*
4	*554b*	*ff90*	*e000*	*fe01*	*0800*	*1000*

Раунд	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$
5	*332d*	*c800*	*d000*	*fffd*	*0008*	*000c*
6	*4aab*	*ffe0*	*ffe4*	*c001*	*0010*	*0014*
7	*aa96*	*f000*	*f200*	*ff81*	*0800*	*0a00*
8	*4925*	*fc00*	*fff8*	*552b*	*0005*	*0006*
9	*0001*	*fffe*	*fffd*	*c001*	–	–

9.2.5. Шифрование данных в IDEA

Зашифрование данных в IDEA

IDEA симметричный блоковый шифр и как уже было отмечено, процесс зашифрования данных аналогичен процессу их расшифрования. Структурная схема алгоритма зашифрования (расшифрования) данных в *IDEA* представлена на рис. 9.1 – 9.4.

Процесс зашифрования (расшифрования) данных представляет собой восемь одинаковых раундов, а девятый раунд – выходное преобразование.

Рассмотрим общую схему зашифрования (расшифрования) *IDEA* (рис. 9.6).

Как и в любом алгоритме шифрования, здесь существует два входа: исходный блок данных и раундовые ключи.

Трансформация представляет собой ряд операций, показанных на рис. 9.7, а шифрование представляет собой ряд операций, показанных на рис. 9.8.

64-битовый блок данных делится на четыре *16*-битовых подблока (см. рис. 9.3):

$$D_1^{(0)},\ D_2^{(0)},\ D_3^{(0)},\ D_4^{(0)}.$$

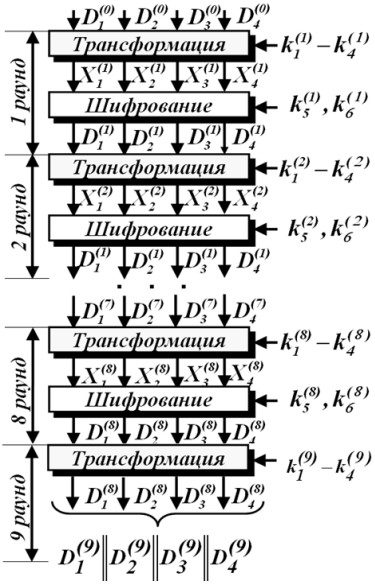


Рис. 9.6. Структурная схема алгоритма шифрования данных в *IDEA*

Эти четыре подблока становятся входом в первый раунд алгоритма шифрования. Всего выполняется восемь раундов. Между раундами второй и третий подблоки меняются местами (см. рис. 9.3).

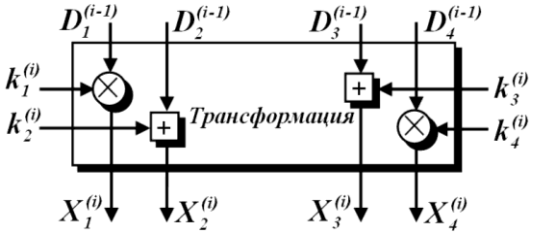


Рис. 9.7. Структура процесса трансформации данных
в раунде шифра *IDEA*

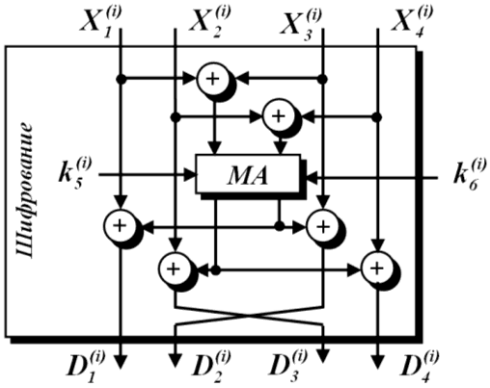


Рис. 9.8. Структура процесса шифрования данных
в раунде шифра *IDEA*

В каждом раунде (кроме девятого) имеет место следующая последовательность операций (см. рис. 9.8):

1. Умножение подблока $D_1^{(i-1)}$ и первого раундового ключа $k_1^{(i)}$ по модулю $2^{16} + 1$ (*65537*):

$$X_1^{(i)} = D_1^{(i-1)} \otimes k_1^{(i)}.$$

2. Сложение подблока $D_2^{(i-1)}$ и второго раундового ключа $k_2^{(i)}$ по модулю 216 (65536):

$$X_2^{(i)} = D_2^{(i-1)} \boxplus k_2^{(i)}.$$

3. Сложение подблока $D_3^{(i-1)}$ и третьего раундового ключа $k_3^{(i)}$ по модулю 2^{16} (*65536*):

$$X_3^{(i)} = D_3^{(i-1)} \boxplus k_3^{(i)}.$$

4. Умножение подблока $D_4^{(i-1)}$ и четвертого раундового ключа $k_4^{(i)}$ по модулю $2^{16} + 1$ (*65537*):

$$X_4^{(i)} = D_4^{(i-1)} \otimes k_4^{(i)}.$$

5. Сложение результатов шагов (1) и (3) по модулю *2*:

$$A^{(i)} = X_1^{(i)} \oplus X_3^{(i)}.$$

6. Сложение результатов шагов (2) и (4) по модулю *2*:

$$B^{(i)} = X_2^{(i)} \oplus X_4^{(i)}.$$

7. Умножение результата шага (5) и пятого раундового ключа $k_5^{(i)}$ по модулю $2^{16} + 1$ (*65537*):

$$C^{(i)} = A^{(i)} \otimes k_5^{(i)}.$$

8. Сложение результатов шагов (6) и (7) по модулю 2^{16} (*65536*):

$$E^{(i)} = B^{(i)} \boxplus C^{(i)}.$$

9. Умножение результата шага (8) с шестым раундовым ключом по модулю $2^{16} + 1$ (*65537*) (див. рис. 9.2):

$$F^{(i)} = E^{(i)} \otimes k_6^{(i)}.$$

10. Сложение результатов шагов (7) и (9) по модулю 2^{16} (*65536*):

$$G^{(i)} = C^{(i)} \boxplus F^{(i)}.$$

11. Сложение результатов шагов (1) и (9) по модулю *2*:

$$D_1^{(i)} = X_1^{(i)} \oplus F^{(i)}.$$

12. Сложение результатов шагов (3) и (9) по модулю *2*:

$$D_2^{(i)} = X_3^{(i)} \oplus F^{(i)}.$$

13. Сложение результатов шагов (2) и (10) по модулю *2*:

$$D_3^{(i)} = X_2^{(i)} \oplus G^{(i)}.$$

14. Сложение результатов шагов (4) и (10) по модулю *2*:

$$D_4^{(i)} = X_4^{(i)} \oplus G^{(i)}.$$

Выходом раунда являются четыре подблока, которые получаются как результаты выполнения шагов 11–14, и в результате формируется вход для следующего раунда.

После восьмого раунда осуществляется завершающее преобразование (см. рис. 9.7):

1. Умножение подблока $D_1^{(8)}$ и первого раундового ключа $k_1^{(9)}$ по модулю $2^{16} + 1$ (*65537*):

$$D_1^{(9)} = D_1^{(8)} \otimes k_1^{(9)}. \qquad (9.3)$$

2. Сложение подблока $D_3^{(8)}$ и второго раундового ключа $k_2^{(9)}$ по модулю 2^{16} (*65536*):

$$D_2^{(9)} = D_3^{(8)} \boxplus k_2^{(9)}. \qquad (9.4)$$

3. Сложение подблока $D_2^{(8)}$ и третьего раундового ключа $k_3^{(9)}$ по модулю 2^{16} (*65536*):

$$D_3^{(9)} = D_2^{(8)} \boxplus k_3^{(9)}. \qquad (9.5)$$

4. Умножение подблока $D_4^{(8)}$ и четвертого раундового ключа $k_4^{(9)}$ по модулю $2^{16} + 1$ (*65537*):

$$D_4^{(9)} = D_4^{(8)} \otimes k_4^{(9)}. \qquad (9.6)$$

Наконец, эти результирующие подблоки, полученные после завершающего преобразования вновь объединяют для получения блока зашифрованных данных

$$C = D_1^{(9)} \| D_2^{(9)} \| D_3^{(9)} \| D_4^{(9)}.$$

Затем берется следующий *64*-битный блок исходных данных и алгоритм зашифрования повторяется. Так продолжается до тех

пор, пока не будут зашифрованы все *64*-битные блоки исходных данных.

Пример 9.3. Пусть ключ шифрования данных *K* равен

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Исходные данные для зашифрования

$$M = 0000\ 0001\ 0002\ 0003_{16}.$$

Необходимо получить результат зашифрования указанных данных.

Решение. Процесс зашифрования данных алгоритмом *IDEA* поясняется с помощью табл. 9.6.

Таблица 9.6

Раундовые ключи и подблоки для каждого раунда зашифрования

Раунд	*Раундовые ключи*						*Значения подблоков данных*			
	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$	$D_1^{(0)}$	$D_2^{(0)}$	$D_3^{(0)}$	$D_4^{(0)}$
							0000	*0001*	*0002*	*0003*
1	*0001*	*0002*	*0003*	*0004*	*0005*	*0006*	*00f0*	*00f5*	*010a*	*0105*
2	*0007*	*0008*	*0400*	*0600*	*0800*	*0a00*	*222f*	*21b5*	*f45e*	*e959*
3	*0c00*	*0e00*	*1000*	*0200*	*0010*	*0014*	*0f86*	*39be*	*8ee8*	*1173*
4	*0018*	*001c*	*0020*	*0004*	*0008*	*000c*	*57df*	*ac58*	*c65b*	*ba4d*
5	*2800*	*3000*	*3800*	*4000*	*0800*	*1000*	*8e81*	*ba9c*	*f77f*	*3a4a*
6	*1800*	*2000*	*0070*	*0080*	*0010*	*0020*	*6942*	*9409*	*e21b*	*1c64*
7	*0030*	*0040*	*0050*	*0060*	*0000*	*2000*	*99d0*	*c7f6*	*5331*	*620e*
8	*4000*	*6000*	*8000*	*a000*	*c000*	*e001*	*0a24*	*0098*	*ec6b*	*4925*
9	*0080*	*00c0*	*0100*	*0140*	-	-	*11fb*	*ed2b*	*0198*	*6de5*

Таким образом, результат зашифрования исходных данных будет равен:

$$C = 11fb\ ed2b\ 0198\ 6de5_{16}.$$

Расшифрование данных в IDEA

IDEA – симметричный блочный шифр и как уже было отмечено, процесс расшифрования данных аналогичен процессу их зашифрования, поэтому структурная схема алгоритма расшифрования данных в *IDEA* соответствует схеме алгоритма зашифрования (рис. 9.6).

Расшифрование состоит в использовании зашифрованных данных в качестве входа в ту же самую структуру *IDEA*, но с другим набором раундовых ключей чем при зашифровании.

Для доказательства того, что алгоритм расшифрования с соответствующими раундовыми ключами имеет корректный результат, рассмотрим одновременно процессы зашифрования и расшифрования. Каждый из восьми раундов разбит на две стадии преобразования, первая из которых называется трансформацией, а вторая – шифрованием, как показано на рис. 9.6, 9.7 и 9.8.

Рассмотрим преобразования, выполняемые в прямоугольниках на рис. 9.7. При зашифровании поддерживаются следующие соотношения на выходе трансформации (9.3)…(9.6).

При расшифровании данных, необходимо на вход схемы, представленной на рис. 9.6 подать такие данные:

$$D_1^{(0)} = D_1^{(9)},\ D_2^{(0)} = D_2^{(9)},\ D_3^{(0)} = D_3^{(9)},\ D_4^{(0)} = D_4^{(9)}.$$

Первая стадия первого раунда процесса расшифрования схемою, представленной на рис. 9.6, будет поддерживать следующие соотношения:

$$\begin{aligned} X_1^{(1)} &= D_1^{(9)} \otimes 1/k_1^{(9)}; \quad X_2^{(1)} = D_2^{(9)} \boxplus 1/k_2^{(9)}; \\ X_3^{(1)} &= D_3^{(9)} \boxplus 1/k_3^{(9)}; \quad X_4^{(1)} = D_4^{(9)} \otimes 1/k_4^{(9)}. \end{aligned} \tag{9.7}$$

Подставляя соответствующие значения (9.3), (9.4), (9.5) и (9.6) в (9.7) получим:

$$\begin{aligned} X_1^{(1)} &= D_1^{(8)} \otimes 1/k_1^{(9)} \otimes k_1^{(9)} = D_1^{(8)}; \\ X_2^{(1)} &= D_3^{(8)} \boxplus (-k_2^{(9)}) \boxplus k_2^{(9)} = D_3^{(8)}; \\ X_3^{(1)} &= D_2^{(8)} \boxplus (-k_3^{(9)}) \boxplus k_3^{(9)} = D_2^{(8)}; \\ X_4^{(1)} &= D_4^{(8)} \otimes 1/k_4^{(9)} \otimes k_4^{(9)} = D_4^{(8)}. \end{aligned} \tag{9.8}$$

Таким образом, выход первой стадии процесса расшифрования эквивалентен входу последней стадии процесса зашифрования за исключением чередования второго и третьего подблоков.

Значение подблоков на выходе восьмого раунда в соответствие со схемой на рис. 9.3 и рис. 9.6 будут определяться следующими соотношениями:

$$\begin{aligned} D_1^{(8)} &= X_1^{(8)} \oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}); \\ D_2^{(8)} &= X_3^{(8)} \oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}); \\ D_3^{(8)} &= X_2^{(8)} \oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}); \\ D_4^{(8)} &= X_4^{(8)} \oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}), \end{aligned} \tag{9.9}$$

где $MA_R(X,Y)$ – правый выход MA структуры со входами X и Y, и $MA_L(X,Y)$ – левый выход MA структуры со входами X и Y.

Значение подблока $D_1^{(1)}$ на выходе первого раунда в соответствие со схемой на рис. 9.3 и рис. 9.6 будет равно:

$$\begin{aligned} D_1^{(1)} &= X_1^{(1)} \oplus MA_R(X_1^{(1)} \oplus X_2^{(1)}, X_3^{(1)} \oplus X_4^{(1)}) = \\ &= D_1^{(8)} \oplus MA_R(D_1^{(8)} \oplus D_2^{(8)}, D_3^{(8)} \oplus D_4^{(8)}). \end{aligned} \tag{9.10}$$

Подставляя значения (9.9) в (9.10) получим

$$\begin{aligned} D_1^{(1)} = X_1^{(8)} &\oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}) \oplus \\ \oplus MA_R(X_1^{(8)} &\oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}) \oplus \\ \oplus X_3^{(8)} &\oplus MA_R(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}), \\ X_2^{(8)} &\oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)}) \oplus \\ \oplus X_4^{(8)} &\oplus MA_L(X_1^{(8)} \oplus X_3^{(8)}, X_2^{(8)} \oplus X_4^{(8)})). \end{aligned} \tag{9.11}$$

Производя преобразования в (9.11) окончательно будем иметь

$$D_1^{(1)} = X_1^{(8)}.$$

Аналогично можно получить и другие значения подблоков на выходе первого раунда:

$$D_2^{(1)} = X_3^{(8)}, \quad D_3^{(1)} = X_2^{(8)}, \quad D_4^{(1)} = X_4^{(8)}.$$

Таким образом, выход второй стадии процесса расшифрования эквивалентен входу предпоследней стадии процесса зашифрования за исключением чередования второго и третьего подблоков.

Аналогично можно показать, что

$$D_1^{(8)} = X_1^{(1)}, \quad D_2^{(8)} = X_3^{(1)}, \quad D_3^{(8)} = X_2^{(1)}, \quad D_4^{(8)} = X_4^{(1)}.$$

Наконец, так как выход трансформации процесса расшифрования эквивалентен первой стадии процесса зашифрования за исключением чередования второго и третьего подблоков, получается, что выход всего процесса расшифрования эквивалентен входу процесса зашифрования. Иными словами, в двух смежных (нечетном и четном) раундах идет дважды перестановка внутренних подблоков данных. Значит, на выходе восьмого раунда значения внутренних подблоков данных будут иметь естественный порядок. В девятом раунде порядок следования внутренних подблоков данных будет изменен дважды и поэтому на выходе раунда порядок следования также будет естественным.

Таким образом, доказано, что алгоритм расшифрования с соответствующими раундовыми ключами имеет корректный результат.

Пример 9.4. Пусть ключ шифрования данных K равен:

$$K = 0001\ 0002\ 0003\ 0004\ 0005\ 0006\ 0007\ 0008_{16}.$$

Зашифрованные данные равны:

$$C = 11fbed2b01986de5_{16}.$$

Необходимо получить результат расшифрования.

Решение. Процесс расшифрования данных алгоритмом *IDEA* поясняется с помощью табл. 9.7.

Таблица 9.7

Раундовые ключи и подблоки данных для каждого раунда зашифрования

Раунд	Раундовые ключи						Значения подблоков данных			
	$k_1^{(i)}$	$k_2^{(i)}$	$k_3^{(i)}$	$k_4^{(i)}$	$k_5^{(i)}$	$k_6^{(i)}$	$D_1^{(0)}$	$D_2^{(0)}$	$D_3^{(0)}$	$D_4^{(0)}$
							11fb	ed2b	0198	6de5
1	fe01	ff40	ff00	659a	c000	e001	d98d	d331	27f6	82b8
2	fffd	8000	a000	cccc	0000	2000	bc4d	e26b	9449	a576
3	a556	ffb0	ffc0	52ab	0010	0020	0aa4	f7ef	da9c	24e3
4	554b	ff90	e000	fe01	0800	1000	ca46	fe5b	dc58	116d
5	332d	c800	d000	fffd	0008	000c	748f	8f08	39da	45cc
6	4aab	ffe0	ffe4	c001	0010	0014	3266	045e	2fb5	b02e
7	aa96	f000	f200	ff81	0800	0a00	0690	050a	00fd	1dfa
8	4925	fc00	fff8	552b	0005	0006	0000	0005	0003	000c
9	0001	fffe	fffd	c001	–	–	0000	0001	0002	0003

Таким образом, результат расшифрования зашифрованных данных будет равен:

$$M = 0000\ 0001\ 0002\ 0003_{16}.$$

Из полученного результата и результата примера 9.3 следует, что расшифрование произведено корректно.

9.2.6. Безопасность IDEA

Анализ алгоритма IDEA

Алгоритм *IDEA* появился в результате незначительных модификаций алгоритма *PES*. На рис. 9.9 приведена структура одного из восьми раундов шифрования данных алгоритмом *PES*, а на рис. 9.10 – завершающего преобразования.

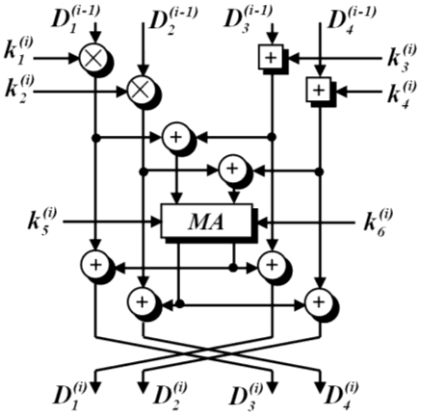


Рис. 9.9. Один раунд преобразования данных в *PES*

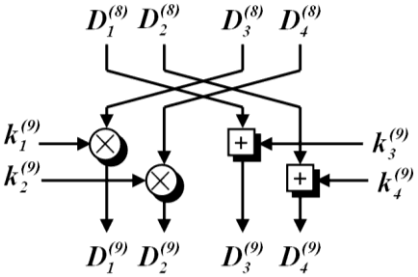


Рис. 9.10. Завершающее преобразование (раунд 9) в *PES*

Из сопоставления структур на рис. 9.4 и рис. 9.9, на рис. 9.4 и рис. 9.10 видно, что изменений не так уж и много.

В алгоритме *IDEA* по сравнением с алгоритмом *PES* произведены следующие изменения (во всех раундах, включая и девятый):

- умножение подблока $D_2^{(i)}$ со вторым раундовым ключом $k_2^{(i)}$ по модулю 2^{16} заменено их сложением по модулю $2^{16}+1$;

- сложение подблока $D_4^{(i)}$ с четвертым раундовым ключом $k_4^{(i)}$ по модулю $2^{16} + 1$ заменено их умножением по модулю 2^{17}.

Кроме того, в алгоритме *IDEA* по сравнением с алгоритмом *PES* изменен сдвиг подблоков данных в конце раунда (только в первых восьми раундах).

Также, в алгоритме *IDEA* по сравнением с алгоритмом *PES* изменен сдвиг подблоков данных в начале завершающего преобразования.

Один из наиболее известных в мире криптологов *Б. Шнайер* в своей книге "*Прикладная криптография*" заметил: "*…удивительно, как такие незначительные изменения могут привести к столь большим различиям*" [47].

В той же книге, вышедшей в 1996 году, *Б. Шнайер* отозвался об *IDEA* так: "*Мне кажется, это самый лучший и надежный блочный алгоритм, опубликованный до настоящего времени*".

В алгоритме *IDEA* используются *64*-битные блоки данных. Длина блока данных должна быть достаточной, чтобы скрыть статистические характеристики исходного сообщения. Но с увеличением размера блока экспоненциально возрастает сложность реализации криптографического алгоритма.

Слабость в ключе и слабые ключи в алгоритме IDEA

В алгоритме *IDEA* используется *128*-битный ключ. Длина ключа должна быть достаточно большой, чтобы предотвратить возможность перебора ключа. Для вскрытия *128*-битного ключа полным перебором ключей при условии, что известны открытые и соответствующие им зашифрованные данные, потребуется 2^{128} (порядка 10^{38}) шифрований. При такой длине ключа *IDEA* считается довольно безопасным. Высокая криптографическая стойкость *IDEA* обеспечивается также такими характеристиками:

- *запутывание* – шифрование зависит от ключа сложным и запутанным образом;

- *рассеивание* – каждый бит исходных данных влияет на каждый бит зашифрованных данных.

Сюэцзя Лай (*Xuejia Lai*) и *Джеймс Мэсси* (*James Massey*) провели тщательный анализ *IDEA* с целью выяснения его криптографической стойкости к дифференциальному криптографическому анализу. Для этого ими было введено понятие марковского шифра и продемонстрировано, что устойчивость к дифференциальному криптографическому анализу может быть промоделирована и оценена ко-

личественно. Линейных или алгебраических слабостей у *IDEA* выявлено не было. Попытка вскрытия с помощью криптографического анализа со связанными ключами, проведенная *Бихамом* (*Biham*), также не увенчалась успехом [20, 26, 44].

Существуют успешные атаки, применимые к *IDEA* с меньшим числом раундов (полный *IDEA* имеет 8 полных и один неполный раунд). Успешной считается атака, если вскрытие шифра с её помощью требует меньшего количества операций, чем при полном переборе ключей. Метод вскрытия *Вилли Майера* (*Willi Meier*) оказался эффективнее вскрытия полным перебором ключей только для *IDEA* с двумя раундами. Методом "*встреча посередине*" был вскрыт *IDEA* с *4,5* раундами. Для этого требуется знание всех 2^{64} блоков из словаря кодов и сложность анализа составляет 2^{112} операций. Лучшая атака на 2007 год применима ко всем ключам и может взломать *IDEA* с *6*-ю раундами [7, 29].

Существуют большие классы слабых ключей *IDEA*. Слабые они в том смысле, что существуют процедуры, позволяющие определить, относится ли ключ к данному классу, а затем и сам ключ. В настоящее время известны следующие:

1. $2^{23} + 2^{35} + 2^{51}$ слабых к дифференциальному криптографическому анализу ключей. Принадлежность к классу 2^{51} можно вычислить за 2^{12} операций с помощью подобранных открытых данных. Авторы данной атаки предложили модификацию алгоритма *IDEA*. Данная модификация заключается в замене раундовых $k_j^{(i)}$ на соответствующие

$$k_j^{'(i)} = a \oplus k_j^{(i)},$$

где i – номер раунда шифрования; j – номер раундового ключа.

Точное значение a не критично. Например, при $a = 0dae_{16}$ (в шестнадцатеричной системе счисления) данные слабые ключи исключаются.

2. 2^{63} слабых к линейному криптографическому анализу ключей. Принадлежность к данному классу выясняется с помощью теста на связанных ключах.

3. $2^{53} + 2^{56} + 2^{64}$ слабых ключей было найдено с использованием *метода бумеранга* (англ. *boomerang attack*), предложенного *Дэвидом Вагнером* (*David Wagner*). Тест на принадлежность к данному классу выполняется за 2^{16} операций и потребует 2^{16} ячеек памяти.

Существование столь больших классов слабых ключей не влияет на практическую криптографическую стойкость алгоритма *IDEA*, так как полное число всех возможных ключей равно 2^{128}.

9.2.7. Применение IDEA

Алгоритм *IDEA* является торговой маркой и запатентован в Австрии, Франции, Германии, Италии, Нидерландах, Испании, Швеции, Швейцарии, Англии, США и в Японии. Сегодня лицензия принадлежит компании *MediaCrypt* и позволяет свободно использовать алгоритм в некоммерческих приложениях. В мае 2005 года *MediaCrypt* официально представила новый шифр *IDEA NXT* (первоначальное название *FOX*), призванный заменить *IDEA*. Типичные области применения *IDEA*:

- шифрование аудио- и видео данных для кабельного телевидения, видеоконференций, дистанционного обучения и др.;
- защита коммерческой и финансовой информации, отражающей конъюнктурные колебания;
- линии связи через модем, роутер или *ATM* (*Asynchronous Transfer Mode* – асинхронный метод перенесения) линию, *GSM* технологию;
- смарт-карты;
- общедоступный пакет конфиденциальной версии электронной почты *PGP v2.0* и (опционально) в *OpenPGP*.

Алгоритм *IDEA* использует шифрующие *SP*-сети общего типа с инвариантом раунда, являющимся побитовой суммой по модулю *2* старшей и младшей половин шифруемого блока. Особенностью этого алгоритма является:

- относительно сложная структура раунда при небольшом их количестве (восемь полных раундов и завершающее преобразование);
- функция шифрования с использованием аддитивных и мультипликативных операций;
- прямая модификация шифруемого блока между раундами с использованием ключевых элементов;
- манипулирование четвертьблоками (*16*-битовыми целыми);
- отсутствие битовых перестановок и табличных подстановок.

В соответствие свыше указанным алгоритм *IDEA* целиком "*арифметический*" алгоритм. Алгоритм *IDEA* оптимизирован для

16-разрядных процессоров с быстрой командой умножения. Очень простая схема выработки раундовых ключей из ключа шифрования.

Алгоритм *IDEA* благодаря использованию операции умножения по модулю $2^{16}+1$, обладающей сильным перемешивающим эффектом, представляется достаточно стойким по отношению к линейному криптографическому анализу. Стойкость этого алгоритма по отношению к дифференциальному методу криптографического анализа не очевидна.

IDEA ориентирован на программную или аппаратную реализацию с использованием встроенного аппаратного умножителя. Однако даже в этом случае умножение по модулю $2^{16}+1$ выполняется программно заметно медленнее, чем сложение, что обусловлено необходимостью выполнения дополнительных операций, кроме собственно умножения *16*-битных чисел. Количество машинных тактов для шифрования *IDEA* в программной реализации при отсутствии специальных мер зависит от вида ключа и шифруемых данных. Поэтому точное измерение длительности шифрования каждого блока позволяет извлечь дополнительную информацию о ключе. Очевидно, это обстоятельство может заметно снизить стойкость *IDEA*. Кроме того, для этого алгоритма существует класс слабых ключей.

9.3. АЛГОРИТМ БЛОЧНОГО ШИФРОВАНИЯ ДАННЫХ CAMELLIA

9.3.1. История создания алгоритма Camellia

Алгоритм блочного симметричного шифрования *Camellia* был разработан компанией *Mitsubishi Electric* в сотрудничестве с еще одной известной японской корпорацией – *Nippon Telegraph and Telephone (NTT)*. Для последней это был не первый опыт участия в международных криптографических конкурсах – на конкурс *AES* корпорация *NTT* выдвигала алгоритм *E2*, который не прошел в финал конкурса [38].

Среди разработчиков *Camellia Kanda M.* – создатель блочного криптографического алгоритма *E2*. *Matsui M.* – автор ряда работ посвященных стойкости блочного симметричного алгоритма *DES* и автор блочного криптографического алгоритма *Misty1*, *Aoki K.* имеющий работы по криптографическому анализу и стойкости блочных симметричных алгоритмов. Поэтому сравнивая *Camellia*

с *E2* и *Misty1* можно найти много идентичного. Разработчики взяли самое лучшее из этих алгоритмов и на базе этих конструктивных решений создали новый шифр.

9.3.2. Структура алгоритма Camellia

Camellia – это *128*-битный блочный симметричный алгоритм, оперирующий с ключами шифрования длиной *128*, *192* и *256* бит. В основе криптографического алгоритма лежит отработанная с годами схема *Фестеля*, хорошо зарекомендовавшая себя в криптографическом алгоритме *DES* [29]. Использование схемы *Фейстеля* позволяет избежать каких-либо ошибок при создании самого алгоритма, так и при его реализации, что качественно отличает его от *Rijndael*. Кроме этого, использование такой конструкции позволяет сделать зашифрование и расшифрование биективным и отличие заключается только в том, что ключи подаются в обратном порядке.

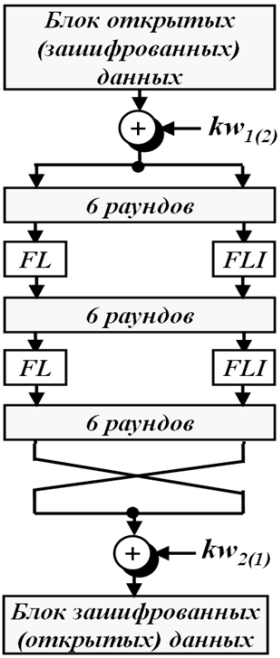


Рис. 9.11. Структура алгоритма шифрования *Camellia*

В зависимости от размера ключа шифрования в алгоритме *Camellia* предусмотрено различное количество раундов *R* [29]:

- *18* – для *128*-битного ключа шифрования;
- *24* – для остальных размеров ключа шифрования.

Структура алгоритма шифрования на примере варианта для *128*-битного ключа шифрования приведена на рис. 9.11.

Перед первым раундом зашифрования выполняется входное отбеливание данных – на блок открытых данных операцией *xor* накладывается *128*-битный фрагмент расширенного ключа kw_1. Затем *128*-битный блок данных делится на два субблока по *64* бита, после чего субблоки "*прогоняются*" через раунды зашифрования. Между каждыми шестью раундами левый субблок обрабатывается функцией *FL*, а правый – функцией *FLI* (эти функции подроб подробно описаны далее).

По завершении последнего раунда субблоки меняются местами. Затем выполняется выходное отбеливание данных; здесь используется еще один *128*-битный фрагмент расширенного ключа – kw_2.

При расшифровании данных *128*-битные фрагменты расширенного ключа используются в обратном порядке

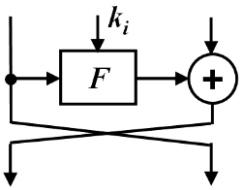


Рис. 9.12. Структура одного раунда шифрования *Camellia*

В каждом раунде левый субблок обрабатывается функцией *F*, которая использует *64*-битный фрагмент ключа k_i (*i* – номер раунда), и накладывается на правый субблок операцией *xor* (см. рис. 9.12). В конце раунда субблоки переставляются местами.

Структура функции *F* алгоритма шифрования *Camellia* приведена на рис. 9.13. На рис. 9.13 S_i это блоки замены.

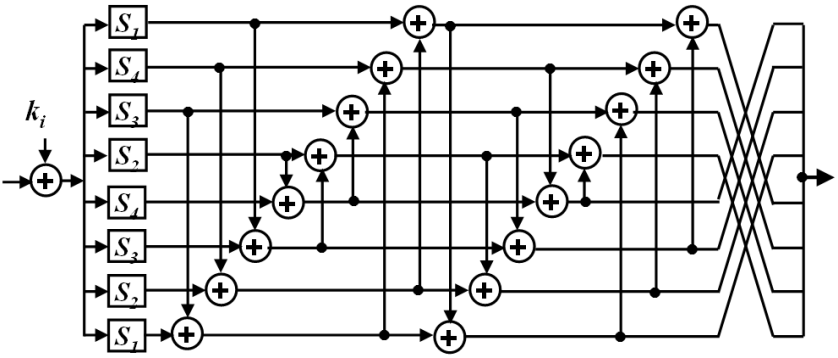


Рис. 9.13. Функция *F* алгоритма шифрования *Camellia*

Перечислим действия функции *F*:

1. Прежде всего выполняется наложение на обрабатываемый субблок данных фрагмента ключа k_i операцией *xor*.

2. Затем *64*-битный результат предыдущей операции разбивается на *8* фрагментов по *8* битов, каждый из которых "*прогоняется*" через табличную замену (*S*-блоки замены). Эта операция подробно описана далее.

3. После этого выполняется наложение байтовых фрагментов друг на друга с помощью операции *xor* по следующему правилу:

$$y_1 = x_1 \oplus x_3 \oplus x_4 \oplus x_6 \oplus x_7 \oplus x_8;$$

$$y_2 = x_1 \oplus x_2 \oplus x_4 \oplus x_5 \oplus x_7 \oplus x_8;$$

$$y_3 = x_1 \oplus x_2 \oplus x_3 \oplus x_5 \oplus x_6 \oplus x_8;$$

$$y_4 = x_2 \oplus x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7;$$

$$y_5 = x_1 \oplus x_2 \oplus x_6 \oplus x_7 \oplus x_8;$$

$$y_6 = x_2 \oplus x_3 \oplus x_5 \oplus x_7 \oplus x_8;$$

$$y_7 = x_3 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_8;$$

$$y_8 = x_1 \oplus x_4 \oplus x_5 \oplus x_6 \oplus x_7;$$

где $x_1 \ldots x_8$ и $y_1 \ldots y_8$ – соответственно, входные и выходные значения обрабатываемых фрагментов.

4. В завершение выполняется простейшая перестановка фрагментов: байты *1...4* меняются местами с байтами *5...8* (см. рис. 9.13), после чего результат снова объединяется в *64*-битный субблок данных.

Псевдокод программы работы функция *F* приведен ниже.

```
x  = F_IN ^ K_E;
t1 =  x >> 56;
t2 = (x >> 48) & MASK8;
t3 = (x >> 40) & MASK8;
t4 = (x >> 32) & MASK8;
t5 = (x >> 24) & MASK8;
t6 = (x >> 16) & MASK8;
t7 = (x >>  8) & MASK8;
t8 =  x & MASK8;
t1 = S1[t1];
t2 = S2[t2];
t3 = S3[t3];
t4 = S4[t4];
t5 = S2[t5];
t6 = S3[t6];
t7 = S4[t7];
```

$t_8 = S_1[t_8];$
$y_1 = t_1 \wedge t_3 \wedge t_4 \wedge t_6 \wedge t_7 \wedge t_8;$
$y_2 = t_1 \wedge t_2 \wedge t_4 \wedge t_5 \wedge t_7 \wedge t_8;$
$y_3 = t_1 \wedge t_2 \wedge t_3 \wedge t_5 \wedge t_6 \wedge t_8;$
$y_4 = t_2 \wedge t_3 \wedge t_4 \wedge t_5 \wedge t_6 \wedge t_7;$
$y_5 = t_1 \wedge t_2 \wedge t_6 \wedge t_7 \wedge t_8;$
$y_6 = t_2 \wedge t_3 \wedge t_5 \wedge t_7 \wedge t_8;$
$y_7 = t_3 \wedge t_4 \wedge t_5 \wedge t_6 \wedge t_8;$
$y_8 = t_1 \wedge t_4 \wedge t_5 \wedge t_6 \wedge t_7;$
$F\_OUT = (y_1 << 56) | (y_2 << 48) | (y_3 << 40) | (y_4 << 32) | (y_5 << 24) | (y_6 << 16) | (y_7 << 8) | y_8.$

В псевдокоде приведенной выше программы символом ^ обозначена операция побитового исключающего *или* (*xor*). S_1, S_2, S_3 и S_4 – блоки замен, которые будут описаны ниже.

Константы $MASK_8$, $MASK_{32}$, $MASK_{64}$ и $MASK_{128}$ приведены в табл. 9.8 (указаны шестнадцатеричные значения).

Таблица 9.8

Значения констант $MASK_8$, $MASK_{32}$, $MASK_{64}$ и $MASK_{128}$

Константа	*Значение*
$MASK_8$	*0xff*
$MASK_{32}$	*0xffffffff*
$MASK_{64}$	*0xffffffffffffffff*
$MASK_{128}$	*0xffffffffffffffffffffffffffffffff*

9.3.3. Процедура расширения ключа в Camellia

Задача расширения ключа состоит в формировании необходимого количества фрагментов расширенного ключа (для перечисленных выше операций) из *128*-, *192*- или *256*-битного исходного ключа шифрования *K*. Данная процедура состоит из нескольких этапов.

Прежде всего, выполняется инициализация переменных K_L и K_R таким образом:

- для *128*-битного ключа $K_L = K$, $K_R = 0$;

- *192*-битный ключ *K* делится на три фрагмента по *64* бита, первые два из которых формируют K_L; третий фрагмент и его побитовый комплемент формируют K_R;

- *256*-битный ключ *K* делится на две части: K_L и K_R.

Пример 9.5. Ключ шифрования *K* длинной *128* бит равен: $0123456789abcdeffedcba9876543210_{16}$. Необходимо определить переменных K_L и K_R.

Решение. Так как ключ шифрования *K* имеет длину *128* бит, то

$$K_L = K = 0123456789abcdeffedcba9876543210_{16},$$

а

$$K_R = 00000000000000000000000000000000_{16}.$$

Пример 9.6. Ключ шифрования *K* длинной *256* бит равен: $0123456789abcdeffedcba9876543210abcdef0123456789fedcba987654210_{16}$. Необходимо определить переменных K_L и K_R.

Решение. Так как ключ шифрования *K* имеет длину *256* бит, то он делится на две части (левую $K_Л$ и правую $K_П$): $K_Л = 0123456789abcdeffedcba9876543210_{16}$; $K_П = abcdef0123456789fedcba9876543210_{16}$. Переменная K_L получает значение:

$$K_L = K_Л = 0123456789abcdeffedcba9876543210_{16},$$

а переменная K_R:

$$K_R = K_П = abcdef0123456789fedcba9876543210_{16}.$$

Пример 9.7. Ключ шифрования *K* длинной *192* бита равен: $0123456789abcdeffedcba9876543210abcdef0123456789_{16}$. Необходимо определить переменных K_L и K_R.

Решение. Так как ключ шифрования *K* имеет длину *192* бита, то он делится на три части (левую $K_Л$, среднюю K_C и правую $K_П$): $K_Л = 0123456789abcdef_{16}$; $K_C = fedcba9876543210_{16}$; $K_П = abcdef0123456789_{16}$. Переменная K_L получает значение:

$$K_L = K_Л \,||\, K_C = 0123456789abcdeffedcba9876543210_{16},$$

а переменная K_R:

$$K_L = K_П \,||\, \neg K_П = abcdef012345678954321 0fedcba9876_{16},$$

где $\neg$ – операция инверсии числа.

После определения переменных K_L и K_R вычисляются две *128*-битные ключевые переменные: K_A и K_B. Это выполняется с использованием описанной выше функции *F* таким образом (рис. 9.14):

1. Результат операции $K_L \oplus K_R$ дважды обрабатывается функцией *F* (см. рис. 9.16), в качестве фрагментов ключа которой берутся константы C_1 и C_2.

2. На результат предыдущей операции операцией *xor* накладывается K_L.

3. Снова дважды применяется функция *F* с использованием констант C_3 и C_4 в качестве фрагментов ключа. В результате получается переменная K_A.

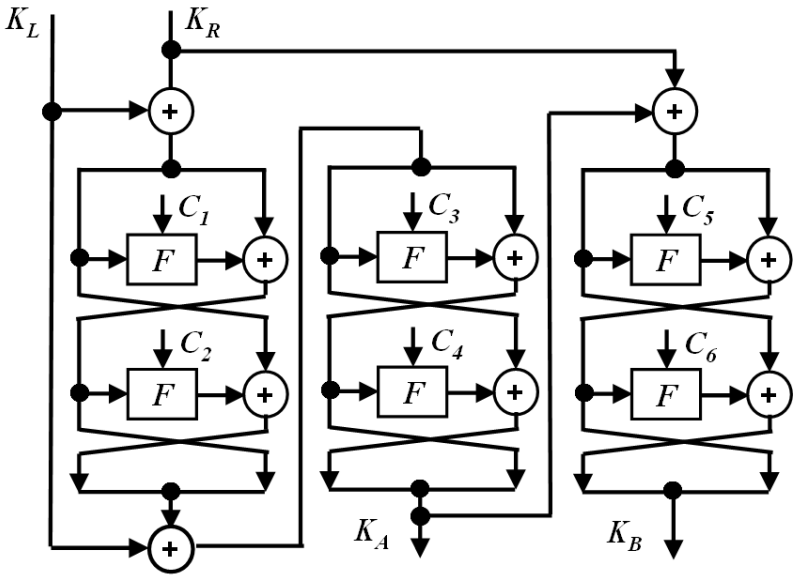


Рис. 9.14. Этап процедуры расширения ключа алгоритма *Camellia*

4. В случае, если используются *192*- или *256*-битные ключи, необходимо также вычислить переменную K_B. Для этого на K_A операцией *xor* накладывается K_R, результат дважды обрабатывается функцией *F* с использованием констант C_5 и C_6.

Константы $C_1...C_6$ приведены в табл. 9.9 (указаны шестнадцатеричные значения).

Таблица 9.9

Значения констант $C_1...C_6$

Константа	*Значение*	*Константа*	*Значение*
C_1	$a09e667f3bcc908b_{16}$	C_4	$54ff53a5f1d36f1c_{16}$
C_2	$в67ae8584caa73в2_{16}$	C_5	$10e527fade682d1d_{16}$
C_3	$c6ef372fe94f82be_{16}$	C_6	$b05688c2b3e6c1fd_{16}$

Псевдокод программы предварительного этапа расширения ключей приведен ниже.

$D_1 = (K_L \wedge K_R) >> 64;$
$D_2 = (K_L \wedge K_R) \,\&\, MASK_{64};$
$D_2 = D_2 \wedge F(D_1, C_1);$
$D_1 = D_1 \wedge F(D_2, C_2);$
$D_1 = D_1 \wedge (K_L >> 64);$
$D_2 = D_2 \wedge (K_L \,\&\, MASK_{64});$
$D_2 = D_2 \wedge F(D_1, C_3);$
$D_1 = D_1 \wedge F(D_2, C_4);$
$K_A = (D_1 << 64) \mid D_2;$
$D_1 = (K_A \wedge K_R) >> 64;$
$D_2 = (K_A \wedge K_R) \,\&\, MASK_{64};$
$D_2 = D_2 \wedge F(D_1, C_5);$
$D_1 = D_1 \wedge F(D_2, C_6);$
$K_B = (D_1 << 64) \mid D_2;$

На следующем этапе вычисляются: вспомогательные *128*-битные ключи kw_1 и kw_2; вспомогательные *64*-битные ключи ke_1, ..., ke_6 и раундовые ключи k_1, ..., k_{24}, в зависимости от размера ключа шифрования. Псевдокоды программ расширения ключей приведены ниже.

128 бит

$kw_1 = K_L;$
$k_1 = (K_A <<< 0) >>> 64;$
$k_2 = (K_A <<< 0) \,\&\, MASK_{64};$
$k_3 = (K_L <<< 15) >> 64;$
$k_4 = (K_L <<< 15) \,\&\, MASK_{64};$
$k_5 = (K_A <<< 15) >> 64;$
$k_6 = (K_A <<< 15) \,\&\, MASK_{64};$
$ke_1 = (K_A <<< 30) >> 64;$
$ke_2 = (K_A <<< 30) \,\&\, MASK_{64};$
$k_7 = (K_L <<< 45) >> 64;$
$k_8 = (K_L <<< 45) \,\&\, MASK_{64};$
$k_9 = (K_A <<< 45) >> 64;$
$k_{10} = (K_L <<< 60) \,\&\, MASK_{64};$
$k_{11} = (K_A <<< 60) >> 64;$
$k_{12} = (K_A <<< 60) \,\&\, MASK_{64};$

$ke_3 = (K_L <<< 77) >> 64;$
$ke_4 = (K_L <<< 77) \& MASK_{64};$
$k_{13} = (K_L <<< 94) >> 64;$
$k_{14} = (K_L <<< 94) \& MASK_{64};$
$k_{15} = (K_A <<< 94) >> 64;$
$k_{16} = (K_A <<< 94) \& MASK_{64};$
$k_{17} = (K_L <<< 111) >> 64;$
$k_{18} = (K_L <<< 111) \& MASK_{64};$
$kw_2 = (K_A <<< 111).$

Вспомогательные *128*-битные ключи используются для отбеливания данных перед первым раундом зашифрования kw_1, а kw_2 – после завершения зашифрования. При расшифровании они используются в обратном порядке.

64-битные раундовые ключи k_1, ..., k_{18} используются в раундах зашифрования данных. При расшифровании они используются в обратном порядке.

При зашифровании вспомогательные *64*-битные ключи ke_1 и ke_3 используются в функциях *FL*, а ke_2 и ke_4 – в функциях *FLI*. При расшифровании вспомогательные *64*-битные ключи ke_4 и ke_2 используются в функциях *FL*, а ke_3 и ke_1 – в функциях *FLI*.

192 и 256 бит

$kw_1 = K_L;$
$k_1 = (K_B <<< 0) >> 64;$
$k_2 = (K_B <<< 0) \& MASK_{64};$
$k_3 = (K_R <<< 15) >> 64;$
$k_4 = (K_R <<< 15) \& MASK_{64};$
$k_5 = (K_A <<< 15) >> 64;$
$k_6 = (K_A <<< 15) \& MASK_{64};$
$ke_1 = (K_R <<< 30) >> 64;$
$ke_2 = (K_R <<< 30) \& MASK_{64};$
$k_7 = (K_B <<< 30) >> 64;$
$k_8 = (K_B <<< 30) \& MASK_{64};$
$k_9 = (K_L <<< 45) >> 64;$
$k_{10} = (K_L <<< 45) \& MASK_{64};$
$k_{11} = (K_A <<< 45) >> 64;$
$k_{12} = (K_A <<< 45) \& MASK_{64};$
$ke_3 = (K_L <<< 60) >> 64;$
$ke_4 = (K_L <<< 60) \& MASK_{64};$

$k_{13} = (K_R <<< 60) >> 64;$
$k_{14} = (K_R <<< 60) \& MASK_{64};$
$k_{15} = (K_B <<< 60) >> 64;$
$k_{16} = (K_B <<< 60) \& MASK_{64};$
$k_{17} = (K_L <<< 77) >> 64;$
$k_{18} = (K_L <<< 77) \& MASK_{64};$
$ke_5 = (K_A <<< 77) >> 64;$
$ke_6 = (K_A <<< 77) \& MASK_{64};$
$k_{19} = (K_R <<< 94) >> 64;$
$k_{20} = (K_R <<< 94) \& MASK_{64};$
$k_{21} = (K_A <<< 94) >> 64;$
$k_{22} = (K_A <<< 94) \& MASK_{64};$
$k_{23} = (K_L <<< 111) >> 64;$
$k_{24} = (K_L <<< 111) \& MASK_{64};$
$kw_2 = (K_B <<< 111).$

Вспомогательные *128*-битные ключи используются для отбеливания данных перед первым раундом зашифрования kw_1, а kw_2 – после завершения зашифрования. При расшифровании они используются в обратном порядке.

64-битные раундовые ключи $k_1, ..., k_{24}$ используются в раундах зашифрования данных. При расшифровании они используются в обратном порядке.

При зашифровании вспомогательные *64*-битные ключи ke_1, ke_3 и ke_5 используются в функциях *FL*, а ke_2, ke_4 и ke_6 – в функциях *FLI*. При расшифровании вспомогательные *64*-битные ключи ke_6, ke_4 и ke_2 используются в функциях *FL*, а ke_5, ke_3 и ke_1 – в функциях *FLI*.

В процессе шифрования переменные K_L, K_R, K_A и K_B используются в качестве фрагментов расширенного ключа согласно приведенным далее таблицам (в порядке использования фрагментов).

Для *128*-битного ключа – см. табл. 9.10, для остальных размеров ключей – табл. 9.11.

Таблица 9.10

Использование переменных K_L, K_R, K_A и K_B в качестве фрагментов расширенного ключа для *128*-битного ключа шифрования

k_{W1}	K_L
k_1 и k_2	Левая и правая половины K_A
k_3 и k_4	Левая и правая половины (K_L<<<*15*)
k_5 и k_6	Левая и правая половины (K_A<<<*15*)

Окончание табл. 9.10

k_{W1}	K_L
ke_1 и ke_2	Левая и правая половины (K_A<<<*30*)
k_7 и k_8	Левая и правая половины (K_L<<<*45*)
k_9	Левая половина (K_A<<<*45*)
k_{10}	Правая половина (K_L<<<*60*)
k_{11} и k_{12}	Левая и правая половины (K_A<<<*60*)
ke_3 и ke_4	Левая и правая половины (K_L<<<*77*)
k_{13} и k_{14}	Левая и правая половины (K_L<<<*94*)
k_{15} и k_{16}	Левая и правая половины (K_A<<<*94*)
k_{17} и k_{18}	Левая и правая половины (K_L<<<*111*)
k_{W2}	K_A<<<*111*

Таблица 9.11

Использование переменных K_L, K_R, K_A и K_B в качестве фрагментов расширенного ключа для *192*- и *256*-битных ключей

k_{W1}	K_L
k_1 и k_2	Левая и правая половины K_B
k_3 и k_4	Левая и правая половины (K_R<<<*15*)
k_5 и k_6	Левая и правая половины (K_A<<<*15*)
ke_1 и ke_2	Левая и правая половины (K_R<<<*30*)
k_7 и k_8	Левая и правая половины (K_B<<<*30*)
k_9 и k_{10}	Левая и правая половины (K_L<<<*45*)
k_{11} и k_{12}	Левая и правая половины (K_A<<<*45*)
ke_3 и ke_4	Левая и правая половины (K_L<<<*60*)
k_{13} и k_{14}	Левая и правая половины (K_R<<<*60*)
k_{15} и k_{16}	Левая и правая половины (K_B<<<*60*)
k_{17} и k_{18}	Левая и правая половины (K_L<<<*77*)
ke_5 и ke_6	Левая и правая половины (K_A<<<*77*)
k_{19} и k_{20}	Левая и правая половины (K_R<<<*94*)
k_{21} и k_{22}	Левая и правая половины (K_A<<<*94*)
k_{23} и k_{24}	Левая и правая половины (K_L<<<*111*)
k_{W2}	K_A<<<*111*

9.3.4. Зашифрование данных в Camellia

Функция FL алгоритма Camellia

Выполняемая через каждые *6* раундов функция *FL* выполняет следующие действия (рис. 9.15):

$$R' = R \oplus ((L \,\&\, ke_L) <<< 1);\ L' = L \oplus (R' \mid ke_R),$$

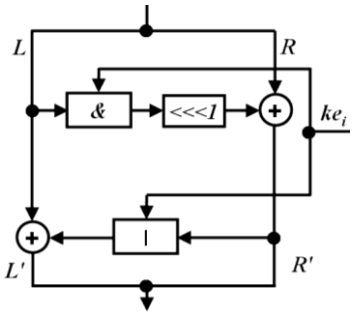


Рис. 9.15. Функция *FL* алгоритма *Camellia*

где L и R – левая и правая части входного значения обрабатывае- мого субблока (каждый длинной по *32* бита); L' и R' – левая и правая части выходного значения (также каждый длинной по *32* бита); ke_L и ke_R – левая и правая части фрагмента *64*-разрядного ключа ke_i ($i = 1, 3$ и 5 в зависимости от числа раундов) для функции *FL;* $<<< 1$ – операция циклического сдвига влево на один бит; & и | – побитовые логические операции "*и*" (*and* – умножение) и "*или*" (*or* – сложение) соответственно.

Псевдокод программы работы функция *FL* приведен ниже.

var x_1, x_2 ***as 32-bit unsigned integer;***
var k_1, k_2 ***as 32-bit unsigned integer;***
$x_1 = FL\_IN >> 32;$
$x_2 = FL\_IN \;\&\; MASK_{32};$
$k_1 = ke_L;$
$k_2 = ke_R;$
$x_2 = x_2 \wedge ((x_1 \;\&\; k_1) <<< 1);$
$x_1 = x_1 \wedge (x_2 \mid k_2);$
$FL\_OUT = (x_1 << 32) \mid x_2.$

Функция FLI алгоритма Camellia

Функция *FLI* определена так (рис. 9.15):

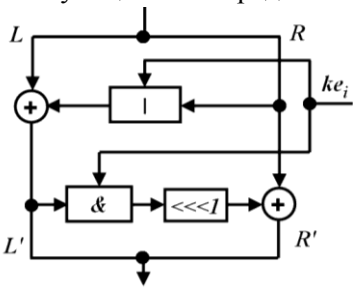


Рис. 9.15. Функция *FLI* алгоритма *Camellia*

$$L' = L \oplus (R' \mid ke_L);$$
$$R' = R \oplus ((L \,\&\, ke_R) <<< 1),$$

где L и R – левая и правая части входного значения обрабатывае- мого субблока (каждый длинной по *32* бита); L' и R' – левая и правая части выходного значения (также каждый длинной по *32* бита); ke_L и ke_R – левая и правая части фрагмента *64*-разрядного ключа ke_i ($i = 2, 4$ и 6 в зависимости от числа раундов) для функции *FLI*.

Псевдокод программы работы функция *FL* приведен ниже.

var y_1, y_2 as 32-bit unsigned integer;
var k_1, k_2 as 32-bit unsigned integer;
y_1 = FLI_IN >> 32;
y_2 = FLI_IN & $MASK_{32}$;
k_1 = ke_L;
k_2 = ke_R;
y_1 = y_1 ^ (y_2 | k_2);
y_2 = y_2 ^ ((y_1 & k_1) <<< 1);
FLI_OUT = (y_1 << 32) | y_2.

Блоки замены алгоритма Camellia

В качестве *S*-блоков замены (рис. 9.13) взяты функции афинно-эквивалентные к функции вычисления обратного элемента в поле *Галуа* (*GF*(2^8)). Хорошо известно, что минимумом для максимальной дифференциальной и линейной вероятности функций в *GF*(2^8) которая была проверена – это 2^{-6} [29]. Авторы выбрали этот тип функций в качестве *S*-блоков замены. Кроме этого, высокая степень булевых полиномов каждого выходного бита из *S*-блоков замены усложняет выполнение атаки дифференциалами высокого порядка. Авторы утверждают, что степень булевских полиномов каждого выходного бита этих *S*-блоков действительно равна *7*. Для усложнения реализации интерполяционной атаки дополнительно используются еще две аффинные функции на входе и выходе из *S*-блоков, что усложняет выражение самих *S*-блоков в *GF*(2^8).

В алгоритме используются *4* блоки замен: S_1 ... S_4. Эти блоки незначительно отличаются друг от друга, и получаются при помощи математических преобразований из основного *S*-блока, что экономит память при реализации криптографического алгоритма Камелия на смарт-картах. Использование *4 S*-блоков повышает безопасность против усеченного дифференциального криптографического анализа.

Блоки замены обрабатывают байтовые фрагменты x_1 ... x_8 следующим образом:

- S_1 блок замены обрабатывает фрагменты x_1 и x_8;
- S_2 блок замены обрабатывает x_4 и x_7;
- S_3 блок замены обрабатывает x_3 и x_6;
- S_4 блок замены обрабатывает x_2 и x_5.

Блоки замены в алгоритме *Camellia* могут быть (в зависимости от требований к реализации шифратора) реализованы как непосредственно с помощью таблиц, так и с помощью следующих вычислений:

- таблица S_1 блока замены:

$$y = h(\, g(\, f(c5 \oplus x))) \oplus 6e\,, \tag{9.12}$$

где *c5* и *6e* – шестнадцатеричные константы; *x* и *y* – соответственно, входное и выходное значения; функции *f, g* и *h* будут описаны далее;

- таблица S_2 блока замены:

$$y = S_1(x) <<< 1; \tag{9.13}$$

- таблица S_3 блока замены:

$$y = S_1(x) >>> 1; \tag{9.14}$$

где >>>1 – операция циклического сдвига вправо на один бит;

- таблица S_4 блока замены:

$$y = S_1(x <<< 1). \tag{9.15}$$

Функция *f* преобразует байт данных так:

$$b_1 = a_6 \oplus a_2; \quad b_2 = a_7 \oplus a_1; \quad b_3 = a_8 \oplus a_5 \oplus a_3; \quad b_4 = a_8 \oplus a_3;$$

$$b_5 = a_7 \oplus a_4; \quad b_6 = a_5 \oplus a_2; \quad b_7 = a_8 \oplus a_1; \quad b_8 = a_6 \oplus a_4,$$

где $a_1 \ldots a_8$ и $b_1 \ldots b_8$ – соответственно, биты входного и выходного значения.

Функция *g* определяет биты выходного значения следующим образом:

$$(b_8 + b_7\alpha + b_6\alpha^2 + b_5\alpha^3) + (b_4 + b_3\alpha + b_2\alpha^2 + b_1\alpha^3)\beta =$$

$$= \frac{1}{(a_8 + a_7\alpha + a_6\alpha^2 + a_5\alpha^3) + (a_4 + a_3\alpha + a_2\alpha^2 + a_1\alpha^3)\beta}.$$

Блоки замены S_1, S_2, S_3 и S_4 аффинно эквивалентны функции инверсии в $GF(2^8)$. Вычисления выполняются в конечном поле $GF(2^8)$,

обратным значением от *0* является *0*, β – элемент конечного поля $GF(2^8)$, удовлетворяющий условию:

$$\beta^8 + \beta^6 + \beta^5 + \beta^3 + 1 = 0,$$

а

$$\alpha = \beta^{238} = \beta^6 + \beta^5 + \beta^3 + \beta^2$$

элемент в $GF(2^4)$, удовлетворяющий равенству $\alpha^4 + \alpha + 1 = 0$.

Функция *h* выполняет следующие преобразования:

$$b_1 = a_5 \oplus a_6 \oplus a_2; \quad b_2 = a_6 \oplus a_2; \quad b_3 = a_7 \oplus a_4; \quad b_4 = a_8 \oplus a_2;$$

$$b_5 = a_7 \oplus a_3; \quad b_6 = a_8 \oplus a_1; \quad b_7 = a_5 \oplus a_1; \quad b_8 = a_6 \oplus a_1.$$

Вычисляя значения блока замены S_1 по формуле (9.12) при *x* от *0* до *255*, можно получить *256* значений, которые приведены в табл. 9.12 [29].

Таблица 9.12

Значения блока замены S_1

	0	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***a***	***b***	***c***	***d***	***e***	***f***
0	***70***	***82***	***2c***	***ec***	***b3***	***27***	***c0***	***e5***	***e4***	***85***	***57***	***35***	***ea***	***0c***	***ae***	***41***
1	***21***	***ef***	***6b***	***93***	***45***	***19***	***a5***	***21***	***ed***	***0e***	***4f***	***4e***	***1d***	***65***	***92***	***bd***
2	***86***	***b8***	***af***	***8f***	***7c***	***eb***	***1f***	***ce***	***3e***	***30***	***dc***	***5f***	***5e***	***c5***	***0b***	***1a***
3	***a6***	***e1***	***39***	***ca***	***d5***	***47***	***5d***	***3d***	***d9***	***01***	***5a***	***d6***	***51***	***56***	***6c***	***4d***
4	***8b***	***0d***	***9a***	***66***	***fb***	***cc***	***b0***	***2d***	***74***	***12***	***2b***	***20***	***f0***	***b1***	***84***	***99***
5	***d9***	***4c***	***cb***	***c2***	***34***	***7e***	***76***	***05***	***6d***	***b7***	***a9***	***31***	***d1***	***17***	***04***	***d7***
6	***14***	***58***	***3a***	***61***	***de***	***1b***	***11***	***1c***	***32***	***0f***	***9c***	***16***	***53***	***18***	***f2***	***22***
7	***fe***	***44***	***cf***	***b2***	***c3***	***b5***	***7a***	***91***	***24***	***08***	***e8***	***a8***	***60***	***fc***	***69***	***50***
8	***aa***	***d0***	***a0***	***7d***	***a1***	***89***	***62***	***97***	***54***	***5b***	***1e***	***95***	***e0***	***ff***	***64***	***d2***
9	***10***	***c4***	***00***	***48***	***a3***	***f7***	***75***	***db***	***8a***	***03***	***e6***	***da***	***09***	***3f***	***dd***	***94***
a	***87***	***5c***	***83***	***02***	***cd***	***4a***	***90***	***33***	***73***	***67***	***f6***	***f3***	***9d***	***7f***	***bf***	***e2***
b	***52***	***9b***	***d8***	***26***	***c8***	***37***	***c6***	***3b***	***81***	***96***	***6f***	***4b***	***13***	***be***	***63***	***2e***
c	***e9***	***79***	***a7***	***8c***	***9f***	***6e***	***bc***	***8e***	***29***	***f5***	***f9***	***b6***	***2f***	***fd***	***b4***	***59***
d	***78***	***98***	***06***	***6a***	***e7***	***46***	***71***	***ba***	***d4***	***25***	***ab***	***42***	***88***	***a2***	***8d***	***fa***
e	***72***	***07***	***b9***	***55***	***f8***	***ee***	***ac***	***0a***	***36***	***49***	***2a***	***68***	***3c***	***38***	***f1***	***a4***
f	***40***	***28***	***d3***	***7b***	***bb***	***c9***	***43***	***c1***	***15***	***e3***	***ad***	***f4***	***77***	***c7***	***80***	***9e***

Табл. 9.12 трактуется следующим образом: входное значение 00_{16} заменяется на 70_{16}, 01_{16} – на $e6_{16}$ и т.д. до входного значения ff_{16}, которое заменяется на $9e_6$.

Вычисляя значения блока замены S_2 по формуле (9.13), S_3 по формуле (9.14) и S_4 по формуле (9.15) при x от 00_{16} до ff_{16}, можно получить таблицы замен (табл. 9.13…9.15 соответственно).

Таблица 9.13

Значения блока замены S_2

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	e0	05	58	d9	67	4e	81	cb	c9	0b	ae	6a	d5	18	5d	82
1	46	df	d6	27	8a	32	4b	42	db	1c	9e	9c	3a	ca	25	7b
2	0d	71	5f	1f	f8	d7	3e	9d	7c	60	b9	be	bc	8b	16	34
3	4d	c3	72	95	ab	8e	ba	7a	b3	02	b4	ad	a2	ac	d8	9a
4	17	1a	35	cc	f7	99	61	5a	e8	24	56	40	e1	63	09	33
5	bf	98	97	85	68	fc	ec	0a	da	6f	53	62	a3	2e	08	af
6	28	b0	74	c2	bd	36	22	38	64	1e	39	2c	a6	30	e5	44
7	fd	88	9f	65	87	6b	f4	23	48	10	d1	51	c0	f9	d2	a0
8	55	a1	41	fa	43	13	c4	2f	a8	b6	3c	2b	c1	ff	c8	a5
9	20	89	00	90	47	ef	ea	b7	15	06	cd	b5	12	7e	bb	29
a	0f	b8	07	04	9b	94	21	66	e6	ce	ed	e7	3b	fe	7f	c5
b	a4	37	b1	4c	91	6e	8d	76	03	2d	de	96	26	7d	c6	5c
c	d3	f2	4f	19	3f	dc	79	1d	52	eb	f3	6d	5e	fb	69	b2
d	f0	31	0c	d4	cf	8c	e2	75	a9	4	57	84	11	45	1b	f5
e	e4	0e	73	aa	f1	dd	59	14	6c	92	54	d0	78	70	e3	49
f	80	50	a7	f6	77	93	86	83	2a	c7	5b	e9	ee	8f	01	3d

Таблица 9.14

Значения блока замены S_3

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	38	41	16	76	d9	93	60	f2	72	c2	ab	9a	75	06	57	a0
1	91	f7	b5	c9	a2	8c	d2	90	f6	07	a7	27	8e	b2	49	de
2	43	5c	d7	c7	3e	f5	8f	67	1f	18	6e	af	2f	e2	85	0d
3	53	f0	9c	65	ea	a3	ae	9e	ec	80	2d	6b	a8	2b	36	a6
4	c5	86	4d	33	fd	66	58	96	3a	09	95	10	78	d8	42	cc

Окончание табл. 9.14

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
5	ef	26	e5	61	1a	3f	3b	82	b6	db	d4	98	e8	8b	02	eb
6	0a	2c	1d	b0	6f	8d	88	0e	19	87	4e	0b	a9	0c	79	11
7	7f	22	e7	59	e1	da	3d	c8	12	04	74	54	30	7e	b4	28
8	55	68	50	be	d0	c4	31	cb	2a	ad	0f	ca	70	ff	32	69
9	08	62	00	24	d1	fb	ba	ed	45	81	73	6d	84	9f	ee	4a
e	39	83	dc	aa	7c	77	56	05	1b	a4	15	34	1e	1c	f8	52
f	20	14	e9	bd	dd	e4	a1	e0	8a	f1	d6	7a	bb	e3	40	4f

Таблица 9.15

Значения блока замены S_4

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	70	2c	b3	c0	e4	57	ea	ae	23	6b	45	a5	ed	4f	1d	92
1	86	af	7c	1f	3e	dc	5e	0b	a6	39	d5	5d	d9	5a	51	6c
2	8b	9a	fb	b0	74	2b	f0	84	df	cb	34	76	6d	a9	d1	04
3	14	3a	de	11	32	9c	53	f2	fe	cf	c3	7a	24	e8	60	69
4	aa	a0	a1	62	54	1e	e0	64	10	00	a3	75	8a	e6	09	dd
5	87	83	cd	90	73	f6	9d	bf	52	d8	c8	c6	81	6f	13	63
6	e9	a7	9f	bc	29	f9	2f	b4	78	06	e7	71	d4	ab	88	8d
7	72	b9	f8	ac	36	2a	3c	f1	40	d3	bb	43	15	ad	77	80
8	82	ec	27	e5	85	35	0c	41	ef	93	19	21	0e	4e	65	bd
9	b8	8f	eb	ce	30	5f	c5	1a	e1	ca	47	3d	01	d6	56	4d
a	0d	66	cc	2d	12	20	b1	99	4c	c2	7e	05	b7	31	17	d7
b	58	61	1b	1c	0f	16	18	22	44	b2	b5	91	08	a8	fc	50
c	d0	7d	89	97	5b	95	ff	d2	c4	48	f7	db	03	da	3f	94
d	5c	02	4a	33	67	f3	7f	e2	9b	26	37	3b	96	4b	be	2e
e	79	8c	6e	8e	f5	b6	fd	59	98	6a	46	ba	25	42	a2	fa
f	07	55	ee	0a	49	68	38	a4	28	7b	c9	c1	e3	f4	c7	9e

Зашифрование происходит по схеме *Фейстеля* с *18* раундами для *128*-битного ключа и *24* этапами для *192*- и *256*-битных ключей. Каждые *6* этапов применяются функции *FL* и *FLI*.

Псевдокоды программ зашифрования данных для *128*-, *192*- и *256*-битных ключей приведены ниже.

128 бит

$D_1 = M >> 64;$ // *Шифруемое сообщение делится на две части*
$D_2 = M \& MASK_{64};$
$D_1 = D_1 \wedge kw_1;$ // *Предварительное забеливание*
$D_2 = D_2 \wedge kw_2;$
$D_2 = D_2 \wedge F(D_1, k_1);$
$D_1 = D_1 \wedge F(D_2, k_2);$
$D_2 = D_2 \wedge F(D_1, k_3);$
$D_1 = D_1 \wedge F(D_2, k_4);$
$D_2 = D_2 \wedge F(D_1, k_5);$
$D_1 = D_1 \wedge F(D_2, k_6);$
$D_1 = FL(D_1, ke_1);$ // *FL*
$D_2 = FLI(D_2, ke_2);$ // *FLI*
$D_2 = D_2 \wedge F(D_1, k_7);$
$D_1 = D_1 \wedge F(D_2, k_8);$
$D_2 = D_2 \wedge F(D_1, k_9);$
$D_1 = D_1 \wedge F(D_2, k_{10});$
$D_2 = D_2 \wedge F(D_1, k_{11});$
$D_1 = D_1 \wedge F(D_2, k_{12});$
$D_1 = FL(D_1, ke3);$ // *FL*
$D_2 = FLI(D_2, ke4);$ // *FLI*
$D_2 = D_2 \wedge F(D_1, k_{13});$
$D_1 = D_1 \wedge F(D_2, k_{14});$
$D_2 = D_2 \wedge F(D_1, k_{15});$
$D_1 = D_1 \wedge F(D_2, k_{16});$
$D_2 = D_2 \wedge F(D_1, k_{17});$
$D_1 = D_1 \wedge F(D_2, k_{18});$
$D_2 = D_2 \wedge kw3;$ // *Финальное забеливание*
$D_1 = D_1 \wedge kw4;$
$C = (D_2 << 64) \mid D_1.$

192 и 256 бит

$D_1 = M >> 64;$ // *Шифруемое сообщение делится на две части*
$D_2 = M \& MASK_{64};$
$D_1 = D_1 \wedge kw_1;$ // *Предварительное забеливание*
$D_2 = D_2 \wedge kw_2;$
$D_2 = D_2 \wedge F(D_1, k_1);$
$D_1 = D_1 \wedge F(D_2, k_2);$
$D_2 = D_2 \wedge F(D_1, k_3);$
$D_1 = D_1 \wedge F(D_2, k_4);$
$D_2 = D_2 \wedge F(D_1, k_5);$

$D_1 = D_1 \wedge F(D_2, k_6);$
$D_1 = FL(D_1, ke_1);$ // *FL*
$D_2 = FLI(D_2, ke_2);$ // *FLI*
$D_2 = D_2 \wedge F(D_1, k_7);$
$D_1 = D_1 \wedge F(D_2, k_8);$
$D_2 = D_2 \wedge F(D_1, k_9);$
$D_1 = D_1 \wedge F(D_2, k_{10});$
$D_2 = D_2 \wedge F(D_1, k_{11});$
$D_1 = D_1 \wedge F(D_2, k_{12});$
$D_1 = FL(D_1, ke3);$ // *FL*
$D_2 = FLI(D_2, ke4);$ // *FLI*
$D_2 = D_2 \wedge F(D_1, k_{13});$
$D_1 = D_1 \wedge F(D_2, k_{14});$
$D_2 = D_2 \wedge F(D_1, k_{15});$
$D_1 = D_1 \wedge F(D_2, k_{16});$
$D_2 = D_2 \wedge F(D_1, k_{17});$
$D_1 = D_1 \wedge F(D_2, k_{18});$
$D_1 = FL(D_1, ke5);$ // *FL*
$D_2 = FLI(D_2, ke6);$ // *FLI*
$D_2 = D_2 \wedge F(D_1, k_{19});$
$D_1 = D_1 \wedge F(D_2, k_{20});$
$D_2 = D_2 \wedge F(D_1, k_{21});$
$D_1 = D_1 \wedge F(D_2, k_{22});$
$D_2 = D_2 \wedge F(D_1, k_{23});$
$D_1 = D_1 \wedge F(D_2, k_{24});$
$D_2 = D_2 \wedge kw3;$ // *Финальное забеливание*
$D_1 = D_1 \wedge kw4;$
$C = (D_2 << 64) \mid D_1.$

Пример 9.8. Ключ шифрования K длинной *128* бит равен: $0123456789abcdeffedcba9876543210_{16}$. Шифруемое сообщение: $M = 0123456789abcdeffedcba9876543210_{16}$. Определить раундовые ключи и зашифрованное сообщение.

Решение: Раундовые ключи:

$k_1 = ae71c3d55ba6bf1d_{16}$; $k_2 = 169240a795f89256_{16}$;
$k_3 = a2b3c4d5e6f7ff6e_{16}$; $k_4 = 5d4c3b2a19080091_{16}$;
$k_5 = e1eaadd35f8e8b49_{16}$; $k_6 = 2053cafc492b5738_{16}$;
$k_7 = 79bdffdb97530eca_{16}$; $k_8 = 8642002468acf135_{16}$;
$k_9 = d7e3a2d24814f2bf_{16}$; $k_{10} = 00123456789abcde_{16}$;
$k_{11} = d169240a795f83df5_{16}$; $k_{12} = 6ae71c3d55ba6bf1_{16}$;
$k_{13} = 1d950c840048d159_{16}$; $k_{14} = e26af37bffb72ea6_{16}$;
$k_{15} = e57e2495ab9c70f5_{16}$; $k_{16} = 56e9afc745a49029_{16}$;

$kw_1 = 0123456789abcdef_{16}$; $kw_2 = fedcba9876543210_{16}$;
$kw_3 = 492b5738e1eaadd3_{16}$; $kw_4 = 5f8e8b492053cafc_{16}$;
$ke_1 = 56e9afc745a49029_{16}$; $ke_2 = e57e2495ab9c70f5_{16}$;
$ke_3 = 97530eca86420024_{16}$; $ke_4 = 68acf13579bdffdb_{16}$.

Зашифрованное сообщение: $C = 67673138549669730857065648 eabe43_{16}$.

9.3.5. Расшифрование данных в Camellia

Расшифрование данных алгоритмом *Camellia* выполняется полностью аналогично зашифрованию, но с использованием фрагментов расширенного ключа в обратной последовательности, т.е.:

- подключ kw_2 используется при входном отбеливании, и наоборот, подключи kw_1 используется при выходном отбеливании данных;
- в раундах шифрования ключи k_i применяются в обратном порядке: от k_N до k_1;
- в операциях *FL* и *FLI* четные фрагменты расширенного ключа используются при обработке левого субблока, нечетные – правого; и те, и другие берутся в обратной последовательности относительно зашифрования.

Отличия в использовании раундовых и вспомогательных ключах при зашифровании и расшифрования приведен в табл. 9.16.

Таблица 9.16

Отличия в использовании раундовых и вспомогательных ключах при зашифровании и расшифрования

Размер ключа шифрования			
128 бит		*192 или 256 бит*	
Зашифрование	*Расшифрование*	*Зашифрование*	*Расшифрование*
kw_1	kw_2	kw_1	kw_2
kw_2	kw_1	kw_2	kw_1
k_1	k_{18}	k_1	k_{24}
k_2	k_{17}	k_2	k_{23}
k_3	k_{16}	k_3	k_{22}
k_4	k_{15}	k_4	k_{21}
k_5	k_{14}	k_5	k_{20}
k_6	k_{13}	k_6	k_{19}
k_7	k_{12}	k_7	k_{18}

Окончание табл. 9.16

Размер ключа шифрования			
128 бит		*192 или 256 бит*	
Зашифрование	*Расшифрование*	*Зашифрование*	*Расшифрование*
k_8	k_{11}	k_8	k_{17}
k_9	k_{10}	k_9	k_{16}
k_{10}	k_9	k_{10}	k_{15}
k_{11}	k_8	k_{11}	k_{14}
k_{12}	k_7	k_{12}	k_{13}
k_{13}	k_6	k_{13}	k_{12}
k_{14}	k_5	k_{14}	k_{11}
k_{15}	k_4	k_{15}	k_{10}
k_{16}	k_3	k_{16}	k_9
k_{17}	k_2	k_{17}	k_8
k_{18}	k_1	k_{18}	k_7
		k_{19}	k_6
		k_{20}	k_5
		k_{21}	k_4
		k_{22}	k_3
		k_{23}	k_2
		k_{24}	k_1
ke_1	ke_3	ke_1	ke_6
ke_2	ke_4	ke_2	ke_5
		ke_3	Ke_4

9.3.6. Безопасность Camellia

Первичный анализ алгоритма *Camellia* был выполнен его разработчиками. В [20] показана стойкость алгоритма к линейному и дифференциальному криптоанализу, а также к использованию усеченных и невозможных дифференциалов, методу бумеранга, методу интерполяции, сдвиговым атакам и ряду других атак.

Первые отзывы известных экспертов об алгоритме *Camellia* также были крайне положительными. В различных работах отмечается исключительно высокая криптографическая стойкость *Camellia*. *Ларс Кнудсен* утверждает следующее:

- любые практически осуществимые атаки на *Camellia* будут возможны только после принципиальных прорывов в области криптоанализа;

- *Camellia* – один из наиболее стойких современных алгоритмов шифрования.

В оценке ресурсоемкости и быстродействия алгоритма *Camellia* эксперты были далеко не так единодушны:

- отмечается, что *Camellia* существенно проигрывает в скорости алгоритму *Rijndael*;

- *Camellia* предъявляет достаточно высокие требования к оперативной и энергонезависимой памяти;

Весьма много исследований было посвящено ослабленному варианту алгоритма *Camellia* – с уменьшенным количеством раундов, а также без входного/выходного отбеливания и операций *FL/FLI*. Стоит упомянуть следующие работы:

- в частности, описана атака на *9*-раундовый вариант *Camellia*, который вскрывается при наличии 2^{105} выбранных открытых текстов выполнением 2^{64} операций шифрования; там же отмечается, что линейный криптографический анализ неприменим для вскрытия алгоритма;

- найден *9*-раундовый усеченный дифференциал, с помощью которого возможна атака на *11*-раундовую версию *Camellia*;

- предложена атака методом поиска коллизий на *9*-раундовую версию *Camellia*, для которой требуется $2^{113,6}$ выбранных открытых текстов и 2^{121} операций шифрования (*128*-битный ключ) или 2^{13} выбранных открытых текстов и $2^{175,6}$ операций (*192*-битный ключ); там же предложена атака на *10*-раундовую версию с *256*-битным ключом, для которой необходимо 2^{14} выбранных открытых текстов и $2^{239,9}$ операций;

- применен метод невозможных дифференциалов, найден *8*-раундовый невозможный дифференциал, с помощью которого атакована *12*-раундовая версия; для атаки требуется 2^{120} выбранных открытых текстов и 2^{181} операций.

Работы, анализирующие варианты, более близкие к исходной версии *Camellia*, встречаются существенно реже. Примечательна работа, в которой применена *Square*-атака, вскрывающая *9*-раундовый алгоритм *Camellia* (включая операции *FL/FLI* после шестого раунда, но без входного и выходного отбеливания) при наличии $2^{60,5}$ выбранных открытых текстов выполнением $2^{202,2}$ операций шифрования.

Алгоритм *Camellia* был исследован и с точки зрения атак, использующих утечку информации по побочным каналам. В этой связи широко известны, в частности, следующие работы:

- авторы одной из работ утверждают, что *Camellia* – один из лучших алгоритмов, представленных во втором раунде конкурса *NESSIE*, с точки зрения защищенности от атак по потребляемой мощности;
- предложена атака по потребляемой мощности, которая использует особенности процедуры расширения ключа алгоритма *Camellia*.

Camellia – один из алгоритмов – победителей конкурса *NESSIE*. В рамках исследований, проведенных во время конкурса *NESSIE*, не было выявлено каких-либо проблем с криптографической стойкостью данного алгоритма. В настоящее время также не найдено каких-либо серьезных уязвимостей в алгоритме *Camellia*, однако видно, что активный анализ этого алгоритма будет продолжаться.

Контрольные вопросы и задания

1. Какой размер блока обрабатываемых данных, размер ключа шифра и размер ключей раунда в *IDEA*?
2. Какое количество раундов в *IDEA*?
3. Какие простые операции используются в раундах *IDEA*?
4. Какие простые операции используются на этапе трансформации данных в раундах *IDEA*?
5. Какие простые операции используются на этапе шифровании данных в раундах *IDEA*?
6. Поясните алгоритм формирования раундовых ключей при зашифровании данных в *IDEA*.
7. Поясните алгоритм формирования раундовых ключей при расшифровании данных в *IDEA*.
8. Начальный (сеансовый) ключ алгоритма *IDEA* – последовательность длиной 128 битов, которая равна $K = 3f424cdc105ca00d7b\ 3dbe8c96a2978e_{16}$. Сформировать раундовые ключи для второго раунда зашифрования.
9. Определить раундовые ключи расшифрования $k_2^{(i)}$ ($i = 1...9$) алгоритма *IDEA*, если раундовые ключи зашифрования равны: $k_2^{(1)} = 4cdc_{16}$, $k_2^{(2)} = 97be_{16}$, $k_2^{(3)} = 452f_{16}$, $k_2^{(4)} = 5a8a_{16}$,

$k_2^{(5)} = 64b5_{16}$, $k_2^{(6)} = 6bd9_{16}$, $k_2^{(7)} = 00d7_{16}$, $k_2^{(8)} = 9401_{16}$, $k_2^{(9)} = 1728_{16}$, $k_3^{(1)} = 105c_{16}$, $k_3^{(2)} = b820_{16}$, $k_3^{(3)} = 1c7e_{16}$, $k_3^{(4)} = 5e38_{16}$, $k_3^{(5)} = 14bc_{16}$, $k_3^{(6)} = 6a29_{16}$, $k_3^{(7)} = b3db_{16}$, $k_3^{(8)} = af67_{16}$, $k_3^{(9)} = 035e_{16}$.

10. Определить раундовые ключи расшифрования $k_3^{(i)}$ ($i = 1...9$) алгоритма *IDEA*, если раундовые ключи зашифрования равны: $k_2^{(1)} = 4cdc_{16}$, $k_2^{(2)} = 97be_{16}$, $k_2^{(3)} = 452f_{16}$, $k_2^{(4)} = 5a8a_{16}$, $k_2^{(5)} = 64b5_{16}$, $k_2^{(6)} = 6bd9_{16}$, $k_2^{(7)} = 00d7_{16}$, $k_2^{(8)} = 9401_{16}$, $k_2^{(9)} = 1728_{16}$, $k_3^{(1)} = 105c_{16}$, $k_3^{(2)} = b820_{16}$, $k_3^{(3)} = 1c7e_{16}$, $k_3^{(4)} = 5e38_{16}$, $k_3^{(5)} = 14bc_{16}$, $k_3^{(6)} = 6a29_{16}$, $k_3^{(7)} = b3db_{16}$, $k_3^{(8)} = af67_{16}$, $k_3^{(9)} = 035e_{16}$.

11. Определить раундовые ключи расшифрования $k_5^{(i)}$ ($i = 1...9$) алгоритма *IDEA*, если раундовые ключи зашифрования равны: $k_5^{(1)} = 7b3d_{16}$, $k_5^{(2)} = 1af6_{16}$, $k_5^{(3)} = 8035_{16}$, $k_5^{(4)} = 3370_{16}$, $k_5^{(5)} = = 1266_{16}$, $k_5^{(6)} = f424_{16}$, $k_5^{(7)} = c7e8_{16}$, $k_5^{(8)} = 92d4_{16}$.

12. Определить раундовые ключи расшифрования $k_1^{(i)}$ ($i = 1...9$) алгоритма *IDEA*, если раундовые ключи зашифрования равны: $k_1^{(1)} = 3f42_{16}$, $k_1^{(2)} = 96a2_{16}$, $k_1^{(3)} = 192d_{16}$, $k_1^{(4)} = fa32_{16}$, $k_1^{(5)} = edf4_{16}$, $k_1^{(6)} = e500_{16}$, $k_1^{(7)} = 05ca_{16}$, $k_1^{(8)} = 820b_{16}$, $k_1^{(9)} = 3704_{16}$.

13. Определить результат этапа перестановки (трансформации) первого раунда зашифрования алгоритмом *IDEA*, если входные данные равны: $M = 3d550f51d71ee0aa_{16}$. Раундовые ключи зашифрования равны: $k_1^{(1)} = 3f42_{16}$, $k_2^{(1)} = 4cdc_{16}$, $k_3^{(1)} = 105c_{16}$, $k_4^{(1)} = = a00d_{16}$.

14. Определить результат устройства *MA* первого раунда алгоритма *IDEA*, если данные на его входе равны: $f_1^{(1)} = 4cb9_{16}$, $f_2^{(1)} = = 4000_{16}$. Раундовые ключи зашифрования равны: $k_5^{(1)} = 7b3d_{16}$, $k_6^{(1)} = be8c_{16}$.

15. Определить результат зашифрования данных алгоритмом *IDEA*, если результат после *8*-го раунда равен: $D_1^{(8)} = e2fb_{16}$, $D_2^{(8)} = cd9d_{16}$, $D_3^{(8)} = cb10_{16}$, $D_4^{(8)} = 9936_{16}$. Раундовые ключи зашифрования равны: $k_1^{(9)} = 3704_{16}$, $k_2^{(9)} = 1728_{16}$, $k_3^{(9)} = 035e_{16}$, $k_4^{(9)} = cf6f_{16}$.

16. Определить входные данные устройства *MA* этапа зашифрования второго раунда алгоритма *IDEA*, если результаты перестановки данного раунда равны: $X_1^{(1)} = abc3_{16}$, $X_2^{(1)} = 5c2d_{16}$, $X_3^{(1)} = e77a_{16}$, $X_4^{(1)} = 1c2d_{16}$.

17. Какой размер блока обрабатываемых данных, размер ключа шифра и размер ключей раунда в *Camellia*?

18. Какое количество раундов в *Camellia*?

19. Какие операции используются в раундах *Camellia*?

20. Поясните алгоритм формирования раундовых ключей при шифровании данных в *Camellia*.

21. Поясните алгоритм зашифрования данных с помощью *Camellia*.

22. Поясните алгоритм расшифрования данных с помощью *Camellia*.

Раздел 10

АСИММЕТРИЧНЫЕ КРИПТОГРАФИЧЕСКИЕ СИСТЕМЫ

10.1. ПРЕДЫСТОРИЯ И ОСНОВНЫЕ ИДЕИ

Рассмотрим три задачи, решение которых поможет лучше понять идеи и методы криптографии с открытыми ключами (асимметричные криптографические системы). Все эти задачи имеют важное практическое значение.

Первая задача – хранение паролей в компьютере [10]. Известно, что каждый пользователь в сети имеет свой секретный пароль. При входе в сеть пользователь указывает свое имя (несекретное) и затем вводит пароль. Проблема состоит в следующем: если хранить пароль на диске компьютера, то злоумышленник может прочитать его, а затем использовать для несанкционированного доступа (особенно легко это сделать, если злоумышленник работает системным администратором этой сети). Поэтому необходимо организовать хранение паролей в компьютере так, чтобы такой "*взлом*" был невозможен.

Вторая задача возникла с появлением радиолокаторов и системы *противовоздушной обороны* (*ПВО*). При пересечении самолетом границы радиолокатор спрашивает пароль. Если пароль верный, то самолет "*свой*", в противном случае – "*чужой*". Здесь возникает такая проблема: так как пароль должен передаваться по открытому каналу (воздушной среде), то противник может прослушивать все переговоры и узнавать правильный пароль. Затем "*чужой*" самолет в случае запроса повторит перехваченный ранее "*правильный*" пароль в качестве ответа локатору и будет пропущен.

Третья задача похожа на предыдущую и возникает в компьютерных сетях с удаленным доступом, например, при взаимодействии банка и клиента. Обычно в начале сеанса банк запрашивает у клиента имя, а затем секретный пароль, но злоумышленник может узнать пароль, так как линия связи открытая.

Сегодня все эти проблемы решаются с использованием криптографических методов. Решение всех этих задач основано на важном понятии *односторонней функции* (*one-way function*).

Определение 10.1. Пусть дана функция

$$y = f(x), \tag{10.1}$$

определенная на конечном множестве X $(x \in X)$, для которой существует обратная функция

$$x = f^{-1}(y). \tag{10.2}$$

Функция называется о*дносторонней*, если вычисление по формуле (10.1) – простая задача, требующая немного времени, а вычисление по (10.2) – задача сложная, требующая привлечения массы вычислительных ресурсов, например, 10^6–10^{10} лет работы мощного суперкомпьютера.

Данное определение, безусловно, неформально. Строгое определение односторонней функции может быть найдено в [10, 18, 26], но для наших целей достаточно и выше приведенного.

В качестве примера односторонней функции рассмотрим следующую:

$$y = a^x \bmod p, \tag{10.3}$$

где p – некоторое простое число (т.е. такое, которое делится без остатка только на себя и на единицу); x – целое число из множества $\{1,2,\ldots p-1\}$.

Обратная функция обозначается

$$x = \log_a y \bmod p \tag{10.4}$$

и называется дискретным логарифмом.

Для того чтобы обеспечить трудность вычисления по (10.4) при использовании лучших современных компьютеров, в настоящее время используются числа размером более *512* бит. На практике часто применяются и другие односторонние функции, например, так называемые *хеш-функции*, оперирующие с существенно более короткими числами, порядка *128–512* бит.

Сначала мы покажем, что вычисление по (10.3) может быть выполнено достаточно быстро. Начнем с примера вычисления числа $a^{16} \bmod p$. Можно записать:

$$a^{16} \bmod p = (((a^2)^2)^2)^2 \bmod p,$$

т.е. значение данной функции вычисляется всего за четыре операции умножения вместо пятнадцати при "*наивном*" варианте $a \cdot a \cdot \ldots \cdot a$. На этом основан общий алгоритм.

Для описания алгоритма введем величину $t = [\log_2 x]$ – целую часть $\log_2 x$ (далее все логарифмы будут двоичные, поэтому в дальнейшем не будем указывать основание логарифма *2*). Вычисляем числа ряда:

$$a, a^2, a^4, a^8, \ldots, a^{2^t} \pmod p. \tag{10.5}$$

В ряду (10.5) каждое число получается путем умножения предыдущего числа самого на себя по модулю p. Запишем показатель степени x в двоичной системе счисления:

$$x = (x_t x_{t-1} \ldots x_1 x_o)_2.$$

Тогда число $y = a^x \bmod p$ может быть вычислено как

$$y = \prod_{i=0}^{t} a^{x_i \cdot 2^i} \bmod p. \tag{10.6}$$

Все вычисления проводятся по модулю p.

Пример 10.1. Пусть требуется вычислить $3^{100} \bmod 7$.

Имеем $t = [\log 100] = 6$. Вычисляем числа ряда (10.5):

a	a^2	a^4	a^8	a^{16}	a^{32}	a^{64}
3	*2*	*4*	*2*	*4*	*2*	*4*

(10.7)

В общем случае справедливо следующее.

Утверждение 10.1 (о сложности вычислений (10.3)). Количество операций умножения при вычислении (10.3) по описанному методу не превосходит $2 \cdot \log x$.

Доказательство. Для вычисления чисел ряда (10.5) требуется t умножений, для вычисления y по (10.6) не более, чем t умножений (см. пример 10.1). Из условия $t = [\log x]$, учитывая, что $[\log x] \le \log x$, делаем вывод о справедливости доказываемого утверждения.

Замечание. Как будет показано в дальнейшем, при возведении в степень по модулю p имеет смысл использовать только показатели $x < p$. В этом случае можно сказать, что количество операций умножения при вычислении (10.3) не превосходит $2 \cdot \log p$.

Важно отметить, что столь же эффективные алгоритмы вычисления обратной функции (10.4) неизвестны. Один из методов вычисления (10.4), называемый "*шаг младенца, шаг великана*" (будет подробно описан в подразделе 10.2). Этот метод требует порядка $2\sqrt{p}$ операций.

Покажем, что при больших p функция (10.3) действительно односторонняя, если для вычисления обратной функции используется метод "*шаг младенца, шаг великана*". Получаем следующий результат (табл. 10.1).

Таблица 10.1

Количество умножений для вычисления прямой и обратной функции

Количество десятичных знаков в записи p	Вычисление (10.3) ($2 \cdot \log p$ умножений)	Вычисление (10.4) ($2 \cdot \sqrt{p}$ умножений)
12	$2 \cdot 40 = 80$	$2 \cdot 10^{6}$
60	$2 \cdot 200 = 400$	$2 \cdot 10^{30}$
90	$2 \cdot 300 = 600$	$2 \cdot 10^{45}$

Из табл. 10.1 видно, что если использовать модули, состоящие из *50–100* десятичных цифр, то "*прямая*" функция вычисляется быстро, а обратная – практически не вычислима. Рассмотрим, например, суперкомпьютер, который умножает два *90*-значных числа за 10^{-14} сек (для современных компьютеров это пока не доступно). Для вычисления (10.3) такому компьютеру потребуется:

$$T_{ВЫЧ.ПР.} = 600 \cdot 10^{-14} = 6 \cdot 10^{-12} \text{ сек.},$$

а для вычисления (10.4)

$$T_{ВЫЧ.ОБР.} = 10^{45} \cdot 10^{-14} = 10^{31} \text{ сек.},$$

т.е. более 10^{22} лет.

Из этого следует, что вычисление обратных функций практически невозможно при длине чисел порядка *90* десятичных цифр, и использование параллельных вычислений и компьютерных сетей существенно не меняет ситуацию. В рассмотренном примере предполагалось, что обратная функция вычисляется за $2 \cdot \sqrt{p}$ операций. В настоящее время известны и более "*быстрые*" методы вычисления дискретного логарифма, однако общая картина та же – количе-

ство требуемых в них операций намного больше $2 \cdot \log p$. Таким образом, можно утверждать, что функция (10.3) действительно односторонняя, но с оговоркой. Никем не доказано, что обратная функция (10.4) не может быть вычислена столь же быстро, как и "*прямая*".

Используем одностороннюю функцию (10.3) для решения всех трех задач, описанных в начале данного раздела, не забывая, однако, что точно так же может быть использована и любая другая односторонняя функция.

Начнем с хранения паролей в памяти компьютера. Решение задачи основано на том, что пароли вообще не хранятся! Точнее, при регистрации в сети пользователь набирает свое имя и пароль; пусть, например, его имя – "*фрукт*", а пароль – "*абрикос*". Компьютер рассматривает слово "*абрикос*" как двоичную запись числа x и вычисляет (10.3), где a и p – два несекретные числа, возможно даже, всем известные. После этого в памяти компьютера заводится пара (*имя*, y), где y вычислено по (10.3) при $x = пароль$. При всех дальнейших входах этого пользователя после ввода пары ("*фрукт*" – "*абрикос*"), компьютер вычисляет по (10.3) новое значение $y_{НОВ}$ с x = "*абрикос*" и сравнивает с хранящимся в памяти ранее вычисленным значением y. Если $y_{НОВ}$ совпадает с хранящимся в памяти y, соответствующим данному имени, то это законный пользователь. В противном случае это злоумышленник.

Злоумышленник мог бы попытаться найти x по y. Однако, уже при *90*-значных числах для этого потребуется более чем 10^{22} лет.

Таким образом, представленная система хранения пароля весьма надежна и в настоящее время используется во многих реальных операционных системах.

Рассмотрим решение второй задачи (*ПВО* и самолет). Можно использовать следующий метод. Каждому "*своему*" самолету присваивается секретное имя, известное системе *ПВО* и летчику, точнее, бортовому компьютеру. Пусть, например, одному из самолетов присвоено секретное имя *СОКОЛ*, и этот самолет приближается к границе *01 февраля 2014 года в 12 час.45 мин*. Тогда перед приближением к границе бортовой компьютер самолета формирует слово:

СОКОЛ	*2014*	*02*	*01*	*12*	*45*
имя	*год*	*месяц*	*день*	*часы*	*минуты*

Другими словами, компьютер на самолете и станция *ПВО* прибавляют к секретному слову сведения о текущем времени и, рассматривая полученное слово как число x, вычисляют $y = a^x \bmod p$, где a и p не секретны. Затем самолет сообщает число y станции *ПВО*. Станция сравнивает вычисленное ею число y с полученным от самолета. Если вычисленное и полученное значения совпали, то самолет признается как "*свой*".

Противник не может взломать эту систему. Действительно, с одной стороны, он не знает секретного слова *СОКОЛ* и не может найти его по y, так как вычисление x по y занимает, скажем, 10^{22} лет. С другой стороны, он не может просто скопировать y и использовать его в качестве ответа в будущем, так как время пересечения границы никогда не повторяется и последующие значения y будут отличаться от первоначального.

Рассмотренный вариант решения задачи *ПВО* требует точной синхронизации часов в самолете и в локаторе. Эта проблема достаточно легко решается. Например, служба навигации постоянно передает метки времени в открытом виде (время не секретно), и все самолеты и локаторы используют эти метки для синхронизации своих часов. Но есть более тонкие проблемы. Метка времени добавляется в слово x для того, чтобы все вычисляемые значения y были различны и противник не мог их повторно использовать. Однако противник может попытаться мгновенно повторить y в пределах текущей минуты. Как предотвратить эту возможность? Это первый вопрос. Другое затруднение возникает в ситуации, когда самолет посылает число y в конце *45*-й минуты, а локатор принимает его в начале *46*-й. Предоставляем читателю возможность самостоятельно предложить вариант решения этих проблем.

Другой способ решения "задачи *ПВО*" возможен, если использовать дополнительный открытый канал передачи данных от локатора к самолету. Как и выше, каждый "*свой*" самолет и локатор знают секретное слово (типа *СОКОЛ*), которое не заменяется. Обнаружив цель, локатор посылает ей случайно сгенерированное число a ("*вызов*"). Самолет вычисляет $y = a^x \bmod p$, где x – секретное слово (*СОКОЛ*), и сообщает число y локатору. Локатор воспроизводит те же вычисления и сравнивает вычисленное y и принятое. В этой схеме не нужно синхронизировать часы, но, как и ранее, противник не может повторить число y, так как локатор всякий раз посылает разные вызовы (a). Интересно, что эта задача,

по-видимому, была исторически первой, при решении которой использовались односторонние функции.

Третья задача решается совершенно аналогично, и оба рассмотренных метода формирования пароля применимы и используются в реальных сетевых протоколах.

10.2. ПЕРВАЯ КРИПТОСИСТЕМА С ОТКРЫТЫМ КЛЮЧОМ – СИСТЕМА ДИФФИ-ХЕЛЛМАНА

Эта криптосистема была открыта в середине *70*-х годов американскими учеными *У. Диффи* (*Whitfield Diffie*) и *М. Хеллманом* (*Martin Hellman*) и привела к настоящей революции в криптографии и ее практических применениях. Это первая система, которая позволяла защищать информацию без использования секретных ключей, передаваемых по защищенным каналам. Для того чтобы продемонстрировать одну из схем применения таких систем, рассмотрим сеть связи с N пользователями, где N – большое число. Пусть необходимо организовать секретную связь для каждой пары из них. Если использовать обычную систему распределения секретных ключей, то каждая пара абонентов должна быть снабжена своим секретным ключом, т.е. всего потребуется

$$C_N^2 = \frac{N(N-1)}{2} \approx \frac{N^2}{2}$$

ключей.

Если абонентов *100*, то требуется *5000* ключей, если же абонентов 10^4, то ключей должно быть $5 \cdot 10^7$. Видно, что при большом числе абонентов система снабжения их секретными ключами становится очень громоздкой и дорогостоящей.

Диффи и *Хеллман* решили эту проблему за счет открытого распространения и вычисления ключей. Перейдем к описанию предложенной ими системы.

Пусть строится система связи для абонентов *A, B, C*, ... У каждого абонента есть своя секретная и открытая информация. Для организации этой системы выбирается большое простое число p и некоторое число g ($1<g<p-1$), такое, что все числа из множества $\{1,2,..., p-1\}$ могут быть представлены как различные степени $g \bmod p$ (известны различные подходы для нахождения таких чисел g, один из них будет представлен ниже). Числа p и g известны всем абонентам.

Абоненты выбирают большие числа d_A, d_B, d_C, которые хранят в секрете (обычно такой выбор рекомендуется проводить случайно, используя датчики случайных чисел). Каждый абонент вычисляет соответствующее число y, которая открыто передается другим абонентам,

$$\begin{cases} e_A = g^{d_A} \bmod p, \\ e_B = g^{d_B} \bmod p, \\ e_C = g^{d_C} \bmod p. \end{cases} \tag{10.9}$$

В результате получаем следующую таблицу.

Таблица 10.2

Ключи пользователей в системе Диффи–Хеллмана

Абонент	Секретный ключ	Открытый ключ
A	d_A	e_A
B	d_B	e_B
C	d_C	e_C

Допустим, абонент A решил организовать сеанс связи с B, при этом обоим абонентам доступна открытая информация из табл. 10.2. Абонент A сообщает B по открытому каналу, что он хочет передать ему сообщение. Затем абонент A вычисляет величину:

$$K_{AB} = (e_B)^{d_A} \bmod p. \tag{10.10}$$

Никто другой кроме A этого сделать не может, так как число x_A секретно.

В свою очередь, абонент B вычисляет число:

$$K_{BA} = (e_A)^{d_B} \bmod p. \tag{10.11}$$

Схема криптографической системы *Диффи–Хеллмана* представлена на рис. 10.1.

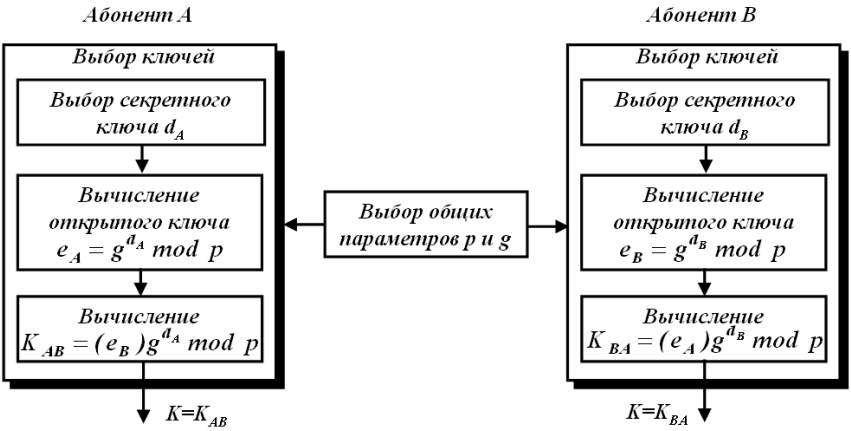


Рис. 10.1. Схема криптографической системы *Диффи–Хеллмана*

Утверждение 10.2. $K_{AB} = K_{BA}$.

Доказательство. Действительно,

$$K_{AB} = (e_B)^{d_A} \bmod p = (g^{d_B})^{d_A} \bmod p =$$
$$= g^{d_A d_B} \bmod p = (e_A)^{d_B} = K_{BA}.$$

Здесь первое равенство следует из (10.10), второе и четвертое – из (10.9), последнее – из (10.11).

Отметим главные свойства системы:

1) абоненты A и B получили одно и то же число $K = K_{AB} = K_{BA}$, которое не передавалось по открытой линии связи;

2) злоумышленник не знает секретных чисел d_A и d_B, поэтому не может вычислить число K_{AB} или K_{BA} (вообще говоря, он мог бы попытаться найти секретное число d_A по e_A (см. (10.9)), однако при больших p это практически невозможно (требуются миллионы лет)).

Абоненты A и B могут использовать $K = K_{AB} = K_{BA}$ в качестве секретного ключа для зашифрования и расшифрования данных. Таким же образом любая пара абонентов может вычислить секретный ключ, известный только им.

Остановимся теперь на упомянутой выше задаче выбора числа g. При произвольно заданном p она может оказаться трудной задачей, связанной с разложением на простые множители числа $p–1$. Дело в том, что для обеспечения высокой стойкости рассмотренной

системы число $p–1$ должно обязательно содержать большой простой множитель (в противном случае *алгоритм Полига-Хеллмана*, описанный, например, в [7, 14, 20], быстро вычисляет дискретный логарифм). Поэтому часто рекомендуют использовать следующий подход. Простое число p выбирается таким, чтобы выполнялось равенство

$$p = 2 \cdot q + 1,$$

где q – также простое число,
тогда в качестве g можно взять любое число, для которого справедливы неравенства

$$1 < g < p - 1 \quad \text{и} \quad g^q \bmod p \neq 1.$$

Пример 10.2. Пусть

$$p = 23 = 2 \cdot 11 + 1 \quad (q = 11).$$

Выберем параметр g. Попробуем взять $g = 3$. Проверим:

$$g^q \bmod p = 3^{11} \bmod 23 = 1,$$

и значит, такое g не подходит. Возьмем $g = 5$. Проверим:

$$g^q \bmod p = 5^{11} \bmod 23 = 22 \neq 1.$$

Итак, мы выбрали параметры $p = 23$, $g = 5$.

Теперь каждый абонент выбирает секретное число и вычисляет соответствующее ему открытое число. Пусть выбраны $d_A = 7$, $d_B = 13$. Вычисляем

$$e_A = g^{d_A} \bmod p = 5^7 \bmod 23 = 17;$$

$$e_B = g^{d_B} \bmod p = 5^{13} \bmod 23 = 21.$$

Пусть A и B решили сформировать общий секретный ключ. Для этого A вычисляет

$$K_{AB} = (e_B)^{d_A} \bmod p = 21^7 \bmod 23 = 10,$$

а B вычисляет

$$K_{BA} = (e_A)^{d_B} \bmod p = 17^{13} \bmod 23 = 10.$$

Теперь они имеют общий ключ $K = K_{AB} = K_{BA} = 10$, который не передавался по каналу связи.

10.3. ЭЛЕМЕНТЫ ТЕОРИИ ЧИСЕЛ

Многие криптографические алгоритмы базируются на результатах классической теории чисел. Рассмотрим необходимый минимум из этой теории. Классические *теоремы Ферма, Эйлера* и ряд других результатов из теории чисел будут даны без доказательств, которые могут быть найдены практически в любом учебнике по теории чисел (см., например, [14]).

Определение 10.2. Целое положительное число p называется *простым*, если оно не делится ни на какое другое число, кроме самого себя и единицы.

Пример 10.3. Числа *11* и *23* – простые; числа *27* и *33* – составные (*27* делится на *3* и на *9*, *33* делится на *3* и на *11*).

Теорема 10.3 (основная теорема арифметики). Любое целое положительное число может быть представлено в виде произведения простых чисел, причем единственным образом.

Пример 10.4. $27 = 3 \cdot 3 \cdot 3$, $33 = 3 \cdot 11$.

Определение 10.3. Два числа называются *взаимно простыми*, если они не имеют ни одного общего делителя кроме единицы.

Пример 10.5. Числа *27* и *28* взаимно просты (у них нет общих делителей кроме единицы), числа *27* и *33* – нет (у них есть общий делитель *3*).

Определение 10.4 (*функция Эйлера*). Пусть дано целое число $n \geq 1$. Значение функции *Эйлера* $\varphi(n)$ равно количеству чисел в ряду $1, 2, 3, \ldots, n-1$, взаимно простых с n.

Пример.10.6.

$\varphi(10)$	$\varphi(12)$
$1, \underline{2}, 3, \underline{4}, \underline{5}, \underline{6}, 7, \underline{8}, 9$	$1, \underline{2}, \underline{3}, \underline{4}, 5, \underline{6}, 7, \underline{8}, \underline{9}, \underline{\underline{10}}, 11$
$\varphi(10) = 4$	$\varphi(12) = 4$

Здесь подчеркнуты двойной линией числа, не взаимно простые с аргументом n.

Утверждение 10.3. Если p – простое число, то $\varphi(p) = p-1$.

Доказательство. В ряду $1, 2, 3, \ldots, p-1$ все числа взаимно просты с p, так как p – простое число и по определению не делится ни на какое другое число.

Утверждение 10.4. Пусть p и q – два различных простых числа ($p \neq q$). Тогда

$$\varphi(p \cdot q) = (p-1) \cdot (q-1).$$

Доказательство. В ряду $1, 2, \ldots, p \cdot g - 1$ не взаимно простыми с $p \cdot q$ будут числа

$$p, 2 \cdot p, 3 \cdot p, \ldots, (q-1) \cdot p \quad \text{и} \quad q, 2 \cdot q, 3 \cdot q, \ldots, (p-1) \cdot q.$$

Всего таких чисел будет $(q-1)+(p-1)$. Следовательно, количество чисел, взаимно простых с $p \cdot q$, будет [14]

$$p \cdot q - 1 - (p-1) - (q-1) = p \cdot q - q - p + 1 = (p-1) \cdot (q-1).$$

Теорема 10.6 (Теорема Ферма). Пусть p – простое число и $0 < a < p$. Тогда

$$a^{p-1} \bmod p = 1.$$

Примеры 10.7.

$p = 13, a = 2;$

$$2^{12} \bmod 13 = (2^2)^2 \cdot ((2^2)^2)^2 \bmod 13 = 3 \cdot 9 \bmod 13 = 1,$$

$p = 11, a = 10;$

$$10^{10} \bmod 11 = 10^2 \cdot ((10^2)^2)^2 \bmod 11 = 1 \cdot 1 \bmod 11 = 1.$$

$p = 10, a = 10;$

$$10^9 \bmod 11 = 10 \cdot ((10^2)^2)^2 \bmod 11 = 10 \cdot 1 \bmod 11 = 10 \neq 1.$$

Теорема 10.7 (Теорема Эйлера). Пусть a и b – взаимно простые числа. Тогда

$$a^{\varphi(b)} \bmod b = 1.$$

Теорема *Ферма* является частным случаем теоремы *Эйлера*, когда b – простое число.

Пример 10.8.

Пусть $a = 5$ и $b = 12$. Ранее (в примере 10.6) было определено

$$\varphi(b) = \varphi(12) = 4.$$

Тогда

$$a^{\varphi(b)} \bmod b = 5^4 \bmod 12 = (5^2)^2 \bmod 12 = (-1)^2 \bmod 12 = 1.$$

Пусть $a = 2$ и $b = 21$. Вычисляя $\varphi(b)$ получим

$$\varphi(b) = \varphi(21) = 12,$$

$$a^{\varphi(b)} \bmod b = 2^{12} \bmod 21 = 2^4 \cdot (2^4)^2 \bmod 21 = 16 \cdot 4 \bmod 21 = 1.$$

Нам понадобится еще одна теорема, близкая к теореме *Эйлера*.

Теорема 10.8. Если p и q – простые числа, $p \neq q$ и k – произвольное целое число, то

$$a^{k \cdot \varphi(p \cdot q)+1} \bmod (p \cdot q) = a. \qquad (10.12)$$

Пример 10.9. Возьмем $p = 5$, $q = 7$. Тогда $p \cdot q = 35$, а функция *Эйлера* – $\varphi(35) = 4 \cdot 6 = 24$. Рассмотрим случай $k = 2$, т.е. будем возводить числа в степень $k \cdot \varphi(p \cdot q)+1 = 2 \cdot 24 + 1 = 49$.

Пусть $a = 9$. В результате этого получим

$$a^{k \cdot \varphi(p \cdot q)+1} \bmod (p \cdot q) = 9^{49} \bmod 35 = 9.$$

Пусть $a = 23$. В результате этого получим

$$a^{k \cdot \varphi(p \cdot q)+1} \bmod (p \cdot q) = 23^{49} \bmod 35 = 23.$$

Это не удивительно, так как каждое из чисел *9* и *23* взаимно просто с модулем *35*, и по теореме *Эйлера* $9^{24} \bmod 35 = 1$, $23^{24} \bmod 35 = 1$. Однако теорема 10.8 остается верной и для следующих чисел:

$$10^{49} \bmod 35 = 10, \quad 28^{49} \bmod 35 = 28,$$

в то время как теорема *Эйлера* для них не применима (каждое из чисел *10* и *28* не взаимно просто с модулем *35* и $10^{24} \bmod 35 = 15$, $28^{24} \bmod 35 = 21$).

Определение 10.5. Пусть a и b – два целых положительных числа. *Наибольший общий делитель* чисел a и b есть наибольшее число c, которое делит и a и b:

$$c = gcd(a, b).$$

Обозначение *gcd* для наибольшего общего делителя происходит от английских слов *greatest common divisor* и принято в современной литературе.

Пример 10.10.

$$gcd(10,15) = 5; \quad gcd(8,28) = 4.$$

В данном примере значение числа *a* равно *10 (8)*, числа *b – 15 (28)*, а *c – 5 (4)*.

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

Алгоритм 10.1. Алгоритм *Евклида* (используется псевдокод):

ВХОД: Положительные целые числа *a*, *b*, $a \geq b$.

1. *while (b != 0)*
2. *{*

    r = a mod b; a = b; b = r;

3. *}*

ВЫХОД: Наибольший общий делитель *c = gcd(a, b)*.

Пример 10.11. Покажем, как с помощью алгоритма *Евклида* вычисляется *gcd(28,8)*:

a:	*28*	*8*	*4*
b:	*8*	*4*	*0*
r:	*4*	*0*	

Здесь каждый столбец представляет собой очередную итерацию алгоритма. Процесс продолжается до тех пор, пока *b* не станет равным нулю. Тогда в значении переменной *a* содержится ответ (*4*).

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

Теорема 10.9. Пусть *a* и *b* – два целых положительных числа. Тогда существуют целые (не обязательно положительные) числа *x* и *y*, такие, что

$$a \cdot x + b \cdot y = gcd(a, b). \qquad (10.13)$$

Обобщенный алгоритм *Евклида* служит для отыскания *gcd(a,b)* и *x*, *y*, удовлетворяющих (10.13). Введем три строки $U = (u_1, u_2, u_3)$, $V = (v_1, v_2, v_3)$ и $T = (t_1, t_2, t_3)$.

Тогда алгоритм записывается следующим образом.

Алгоритм 10.2. Обобщенный алгоритм *Евклида* (используется псевдокод):

ВХОД: Положительные целые числа a, b, $a \geq b$.

1. $U = \{ a, 1, 0 \}$; $V = \{ b, 0, 1 \}$.
2. *while* $(v_1 != 0)$
3. {
   $q = u_1 \; div \; v_1$;
   $T = \{ u_1 \; div \; v_1; \; u_2 - q \cdot v_2 \; ; \; u_3 - q \cdot v_3 \}$;
   $U = V$; $V = T$;
4. }
5. $U = \{ gcd(a, b); x; y \}$

ВЫХОД: $gcd(a, b)$; x; y удовлетворяющие (10.13).

Результат содержится в строке U.

Операция *div* в алгоритме – это целочисленное деление

$$a \; div \; b = [a / b].$$

Доказательство корректности алгоритма 10.2 может быть найдено в [2, 14].

Пример 10.12. Пусть $a = 28$, $b = 19$. Найдем числа x и y, удовлетворяющие (10.13).

U				28	1	0	
V	U			19	0	1	
T	V	U		9	1	-1	$q = 1$
	T	V	U	1	-2	3	$q = 2$
		T	V	0	19	-28	$q = 9$

Поясним представленную схему. Вначале в строку U записываются числа $(28,1,0)$, а в строку V – числа $(19,0,1)$ (это первые две строки на схеме). Вычисляется строка T (третья строка в схеме). После этого в качестве строки U берется вторая строка в схеме, а в качестве V – третья, и опять вычисляется строка T (четвертая строка в схеме). Этот процесс продолжается до тех пор, пока первый элемент строки V не станет равным нулю. Тогда предпоследняя строка в схеме содержит ответ. В нашем случае $gcd(28,19) = 1$, $x = -2$, $y = 3$. Выполним проверку: $28 \cdot (-2) + 19 \cdot 3 = 1$.

Рассмотрим одно важное применение обобщенного алгоритма Евклида. Во многих задачах криптографии для заданных чисел c и m требуется находить такое число $d < m$, что

$$c \cdot d \bmod m = 1. \qquad (10.14)$$

Отметим, что такое d существует тогда и только тогда, когда числа c и m взаимно простые.

Определение 10.6. Число d, удовлетворяющее (10.14), называется мультипликативной инверсией c по модулю m и часто обозначается $c^{-1} \bmod m$.

Данное обозначение для инверсии довольно естественно, так как мы можем теперь переписать (10.14) в виде

$$c \cdot c^{-1} \bmod m = 1.$$

Умножение на c^{-1} соответствует делению на c при вычислениях по модулю m. По аналогии можно ввести произвольные отрицательные степени при вычислениях по модулю m:

$$c^{-e} \bmod m = (c^{e})^{-1} \bmod m = (c^{-1})^{e} \pmod m.$$

Пример 10.13. $3 \cdot 4 \bmod 11 = 1$, поэтому число 4 – это мультипликативная инверсия числа 3 по модулю 11. Можно записать $3^{-1} \bmod 11 = 4$. $5^{-2} \bmod 11$ может быть найдено двумя способами:

$$5^{-2} \bmod 11 = (5^{2} \bmod 11)^{-1} \bmod 11 = 3^{-1} \bmod 11 = 4,$$

$$5^{-2} \bmod 11 = (5^{-1} \bmod 11)^{2} \bmod 11 = 9^{2} \bmod 11 = 4.$$

При вычислениях по второму способу мы использовали равенство $5^{-1} \bmod 11 = 9$. Действительно, $5 \cdot 9 \bmod 11 = 45 \bmod 11 = 1$.

Покажем, как можно вычислить инверсию c помощью обобщенного алгоритма *Евклида*. Равенство (10.14) означает, что для некоторого целого k

$$c \cdot d - k \cdot m = 1. \qquad (10.15)$$

Учитывая, что c и m взаимно простые, перепишем (10.15) в виде

$$m \cdot (-k) + c \cdot d = gcd(m, c) = 1, \qquad (10.16)$$

что полностью соответствует (10.13), здесь только по-другому обозначены переменные. Поэтому, чтобы вычислить $c^{-1} \bmod m$, т.е. найти число d, нужно просто использовать обобщенный алгоритма

Евклида для решения уравнения (10.16). Заметим, что значение переменной k нас не интересует, поэтому можно не вычислять вторые элементы строк U, V, T. Кроме того, если число d получается отрицательным, то нужно прибавить к нему m, так как по определению число $a \bmod m$ берется из множества $\{0,1, ..., m–1\}$.

Пример 10.14. Вычислим $7^{-1} \bmod 11$. Используем такую же схему записи вычислений, как в примере 10.12:

U					11	0	
V	U				7	1	
T	V	U			4	-1	$q = 1$
	T	V	U		3	2	$q = 1$
		T	V	U	1	-3	$q = 1$
			T	V	0	11	$q = 3$

Получаем $d = –3$ и $d \bmod 11 = (11 – 3) \bmod 11 = 8$, т.е. $7^{-1} \bmod 11 = 8$. Проверим результат: $(7 \cdot 8) \bmod 11 = 56 \bmod 11 = 1$.

Одной из важнейших операций в криптографии с открытыми ключами является операция возведения в степень по модулю. Идея построения эффективного алгоритма возведения в степень была ранее проиллюстрирована с помощью (10.5) и (10.6). Рассмотренный алгоритм можно реализовать и без хранения в памяти ряда чисел (10.5). Дадим описание этого алгоритма в форме, пригодной для непосредственной программной реализации. В названии алгоритма отражен тот факт, что биты показателя степени просматриваются справа-налево, т.е. от младшего к старшему.

Алгоритм 10.3. Возведение в степень (справа-налево)

ВХОД: Целые числа a, $x = (x_t x_{t-1}, ..., x_0)_2$, p.
ВЫХОД: Число $y = a^x \bmod p$.
1. $y \leftarrow 1$; $s \leftarrow a$
2. FOR $i = 0,1,...,t$ DO
3. IF $x_i = 1$ THEN $y \leftarrow y \cdot s \bmod p$;
4. $s \leftarrow s \cdot s \bmod p$
5. RETURN y

Чтобы показать, что по представленному алгоритму действительно вычисляется y согласно (10.6), запишем степени переменных после каждой итерации цикла. Пусть $x = 100 = (1100100)_2$, как в примере 10.1, тогда:

i:	0	1	2	3	4	5	6
x_i:	0	0	1	0	0	1	1
y:	1	1	a^4	a^4	a^4	a^{36}	a^{100}
s:	a^2	a^4	a^8	a^{16}	a^{32}	a^{64}	a^{128}.

В некоторых ситуациях более эффективным оказывается следующий алгоритм, в котором биты показателя степени просматриваются слева-направо, т.е. от старшего к младшему.

Алгоритм 10.4. Возведение в степень по модулю (слева-направо):

ВХОД: Целые числа a, $x = (x_t x_{t-1}, ..., x_0)_2$, p.
ВЫХОД: Число $y = a^x \bmod p$.
1. $y \leftarrow 1$;
2. FOR $i = t,t-1,...,0$ DO
3. $\quad y \leftarrow y \cdot y \bmod p$;
4. $\quad$ IF $x_i = 1$ THEN $y \leftarrow y \cdot a \bmod p$;
5. RETURN y.

Чтобы убедиться в том, что алгоритм 10.4 вычисляет то же самое, что и алгоритм 10.3, запишем степени переменной y после каждой итерации цикла для $x = 100$:

i:	6	5	4	3	2	1	0
x_i:	1	1	0	0	1	0	0
y:	a	a^3	a^6	a^{12}	a^{25}	a^{50}	a^{100}.

Приведенных в данном разделе сведений из теории чисел будет достаточно для описания основных криптографических алгоритмов и методов.

10.4. КРИПТОГРАФИЧЕСКАЯ СИСТЕМА ШАМИРА

Эта криптографическая система, предложенная *А. Шамиром* (*Adi Shamir*), была первой, позволяющей организовать обмен секретными сообщениями по открытой линии связи для лиц, которые не имеют никаких защищенных каналов и секретных ключей и, возможно, никогда не видели друг друга. Напомним, что криптографическая система *Диффи-Хеллмана* позволяет сформировать только секретное слово, а передача сообщения потребует использования некоторой криптосистемы, где это слово будет использоваться как ключ.

Перейдем к описанию системы. Пусть есть два абонента A и B, соединенные линией связи. A хочет передать сообщение M абоненту B так, чтобы никто не узнал его содержание. A выбирает случайное большое простое число p и открыто передает его B. Затем A выбирает два числа e_A и d_A, такие, что

$$e_A \cdot d_A \bmod (p-1) = 1. \qquad (10.17)$$

Эти числа A держит в секрете и передавать не будет. B тоже выбирает два числа e_B и d_B, такие, что

$$e_B \cdot d_B \bmod (p-1) = 1, \qquad (10.18)$$

и держит их в секрете.

После этого A передает свое сообщение M, используя трехступенчатый протокол. Если $M < p$ (M рассматривается как число), то сообщение M передается сразу, если же $M > p$, то сообщение представляется в виде $m_1, m_2, \ldots, m_t$, где все $m_i < p$, и затем передаются последовательно $m_1, m_2, \ldots, m_t$. При этом для зашифрования каждого m_i лучше выбирать случайно новые пары (e_A,d_A) и (e_B,d_B) – в противном случае надежность системы понижается. В настоящее время такая криптографическая система, как правило, используется для передачи чисел, например, секретных ключей, значения которых меньше p. Таким образом, мы будем рассматривать только случай $M < p$. Дадим описание протокола.

Шаг 1. A вычисляет число

$$c_1 = M^{e_A} \bmod p, \qquad (10.19)$$

где M – исходное сообщение,
и пересылает c_1 к B.

Шаг 2. B, получив c_1, вычисляет число

$$c_2 = c_1^{e_B} \bmod p \qquad (10.20)$$

и передает c_2 к A.

Шаг 3. A вычисляет число

$$c_3 = c_2^{d_A} \bmod p \qquad (10.21)$$

и передает его B.

Шаг 4. B, получив c_3, вычисляет число

$$c_4 = c_3^{d_B} \bmod p. \qquad (10.22)$$

Схема криптографической системы *Шамира*, которая показывает процессы шифрования данных при их передачи от абонента *A* к абоненту *B*, представлена на рис. 10.2.

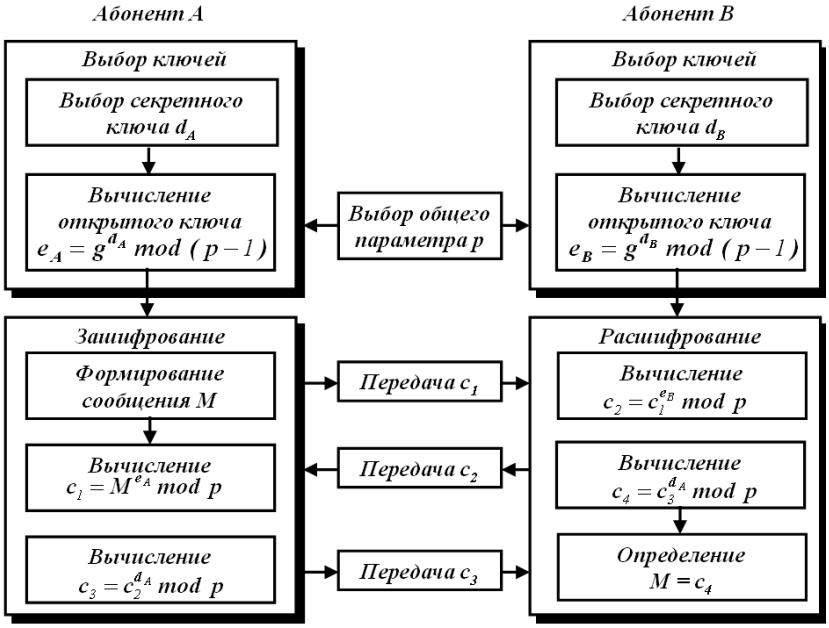


Рис. 10.2. Схема криптографической системы *Шамира*

Схема криптографической системы *Шамира*, которая показывает процессы шифрования данных при их передачи от абонента *B* к абоненту *A*, аналогична представленной, только в этом случае инициатором передачи сообщения будет абонент *B*.

Утверждение 10.5 (свойства протокола *Шамира*):

1) $c_4 = M$, т.е. в результате реализации протокола от *A* к *B* действительно передается исходное сообщение;

2) злоумышленник не может узнать, какое сообщение было передано.

Доказательство. Вначале заметим, что любое целое число $n \geq 0$ может быть представлено в виде

$$n = k \cdot (p-1) + r,$$

где $r = n \bmod (p-1)$.

Поэтому на основании теоремы *Ферма* ($x^{p-1} \bmod p = 1$)

$$x^n \bmod p = x^{k\cdot(p-1)+r} \bmod p = (1^k \cdot x^r) \bmod p = \\ = x^{n \bmod (p-1)} \bmod p. \tag{10.23}$$

Справедливость первого пункта утверждения вытекает из следующей цепочки равенств:

$$c_4 = c_3^{d_B} \bmod p = (c_2^{d_A})^{d_B} \bmod p = (c_1^{e_B})^{d_A \cdot d_B} \bmod p = \\ = (M^{e_A})^{e_B \cdot d_A \cdot d_B} \bmod p = M^{e_A \cdot e_B \cdot d_A \cdot d_B} \bmod p = \\ = M^{(e_A \cdot e_B \cdot d_A \cdot d_B) \bmod (p-1)} \bmod p = M.$$

Предпоследнее равенство следует из (10.23), а последнее выполняется в силу (10.17) и (10.18).

Доказательство второго пункта утверждения основано на предположении, что для злоумышленника, пытающегося определить *M*, не существует стратегии более эффективной, чем следующая. Вначале он вычисляет e_B из (10.20), затем находит d_B и, наконец, вычисляет $c_4 = M$ по (10.22). Но для осуществления этой стратегии злоумышленник должен решить задачу дискретного логарифмирования (10.20), что практически невозможно при больших *p*.

Опишем метод нахождения пар e_A, d_A и e_B, d_B удовлетворяющих (10.17) и (10.18). Достаточно описать только действия для абонента *A*, так как действия для *B* совершенно аналогичны. Число e_A выбираем случайно так, чтобы оно было взаимно простым с $p-1$ (поиск целесообразно вести среди нечетных чисел, так как $p-1$ четно). Затем вычисляем d_A с помощью обобщенного алгоритма Евклида, как это было объяснено в подразделе 10.3.

Пример 10.15. Пусть *A* хочет передать *B* сообщение $M = 10$. *A* выбирает $p = 23$, $e_A = 7$ ($gcd(7, 22) = 1$) и вычисляет $d_A = 19$. Аналогично, *B* выбирает параметры $e_B = 5$ (взаимно простое с 22) и $d_B = 9$. Переходим к протоколу Шамира.

Шаг 1. c_1 в соответствие с (10.19) равно $c_1 = 10^7 \bmod 23 = 14$.

Шаг 2. c_2 в соответствие с (10.20) равно $c_2 = 14^5 \bmod 23 = 15$.

Шаг 3. c_3 в соответствие с (10.21) равно $c_3 = 15^{19} \bmod 23 = 19$.

Шаг 4. c_4 в соответствие с (10.22) равно $c_4 = 19^9 \bmod 23 = 10$.

Таким образом, *B* получил передаваемое сообщение $M = 10$.

10.5. КРИПТОГРАФИЧЕСКАЯ СИСТЕМА ЭЛЬ-ГАМАЛЯ

Пусть имеются абоненты *A*, *B*, *C*, ..., которые хотят передавать друг другу зашифрованные сообщения, не имея никаких защищенных каналов связи. В этом подразделе рассмотрим криптографическую систему, предложенную *Эль-Гамалем* (*Taher ElGamal*), которая решает эту задачу, используя, в отличие от криптографической системы *Шамира*, только одну пересылку сообщения. Фактически здесь используется схема *Диффи-Хеллмана*, чтобы сформировать общий секретный ключ для двух абонентов, передающих друг другу сообщение, и затем сообщение зашифровывается путем умножения его на этот ключ. Для каждого следующего сообщения секретный ключ вычисляется заново. Перейдем к точному описанию криптографической системы.

Для всей группы абонентов выбираются некоторое большое простое число p и число g, такие, что различные степени g суть различные числа по модулю p (см. подраздел 10.2). Числа p и g передаются абонентам в открытом виде (они могут использоваться всеми абонентами сети).

Затем каждый абонент группы выбирает свое секретное число d_i, которое удовлетворяет требованию

$$1 < d_i < p-1$$

и вычисляет соответствующее ему открытое число e_i

$$e_i = g^{d_i} \bmod p. \qquad (10.24)$$

В результате получаем таблицу 10.3.

Таблица 10.3

Ключи пользователей в системе *Эль-Гамаля*

Абонент	Секретный ключ	Открытый ключ
A	d_A	e_A
B	d_B	e_B
C	d_C	e_C

Покажем теперь, как *A* передает сообщение *M* абоненту *B*. Будем предполагать, как и при описании криптографической системы *Шамира*, что сообщение представлено в виде числа $M < p$.

Шаг 1. А формирует случайное число k, $1 < k < p–2$, вычисляет числа

$$r = g^k \bmod p; \tag{10.25}$$

$$c = (M \cdot e_B^{\,k}) \bmod p \tag{10.26}$$

и передает пару чисел (r, c) абоненту *B*.

Шаг 2. B, получив (r, c), вычисляет

$$M' = (c \cdot r^{p-1-d_B}) \bmod p. \tag{10.27}$$

Схема криптографической системы *Эль-Гамаля*, которая показывает процессы шифрования данных при их передачи от абонента *A* к абоненту *B*, представлена на рис. 10.3.

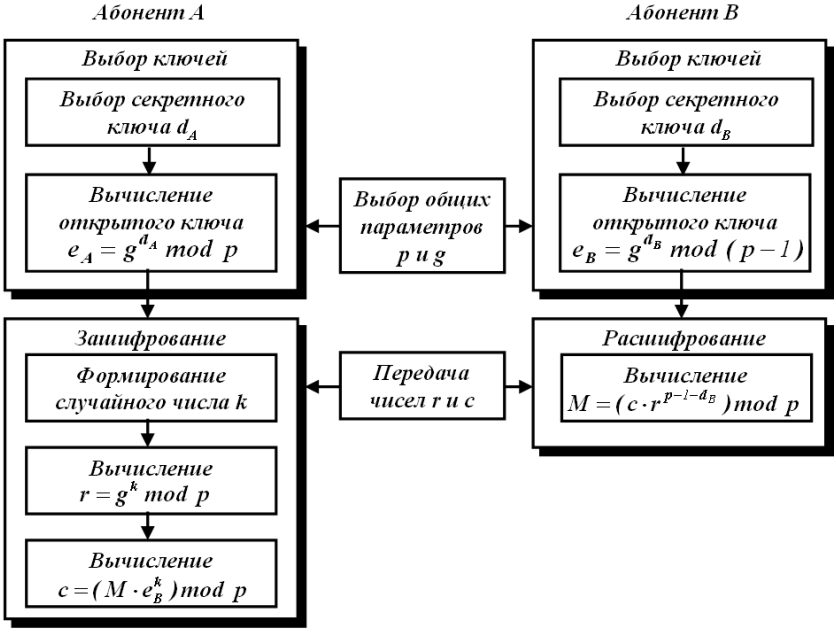


Рис. 10.3. Схема криптографической системы *Эль-Гамаля*

Схема криптографической системы *Эль-Гамаля*, которая показывает процессы шифрования данных при их передачи от абонента *B* к абоненту *A*, аналогична представленной, только в этом случае инициатором передачи сообщения будет абонент *B*.

Утверждение 10.6 (свойства криптографической системы *Эль-Гамаля*):

1) абонент B получил сообщение, т.е. $M' = M$;

2) противник, зная p, g, e_B, r і c, не может вычислить M.

Доказательство. Подставим в (10.27) значение c из (10.26):

$$M' = (M \cdot e_B^{\,k} \cdot r^{p-1-d_B}) \bmod p.$$

Теперь вместо r подставим (10.25), а вместо e_B – (10.24):

$$M' = (M(g^{d_B})^k (g^k)^{p-1-d_B}) \bmod p =$$

$$= (M \cdot g^{d_B \cdot k + k \cdot (p-1) - k \cdot d_B}) \bmod p = (M \cdot g^{k \cdot (p-1)}) \bmod p.$$

По теореме *Ферма*

$$g^{k \cdot (p-1)} \bmod p = 1^k \bmod p = 1,$$

и, таким образом, получаем первую часть утверждения.

Для доказательства второй части заметим, что злоумышленник не может вычислить k в равенстве (10.25), так как это задача дискретного логарифмирования. Следовательно, он не может вычислить M в равенстве (10.26), так как M было умножено на неизвестное ему число. Противник также не может воспроизвести действия законного получателя сообщения (абонента B), так как ему не известно секретное число d_B (вычисление d_B на основании (10.24) – также задача дискретного логарифмирования).

Пример 10.16. Передадим сообщение $M = 15$ от абонента A к абоненту B. Выберем параметры аналогично тому, как это было сделано в примере 10.2. Возьмем $p = 23$, $g = 5$. Пусть абонент B выбрал для себя секретное число $d_B = 13$ и вычислил по (10.24)

$$e_B = g^{d_B} \bmod p = 5^{13} \bmod 23 = 21.$$

Абонент A выбирает случайно число k, например $k = 7$, и вычисляет по (10.25), (10.26):

$$r = g^k \bmod p = 5^7 \bmod 23 = 17,$$

$$c = (M \cdot e_B^{\,k}) \bmod p = (15 \cdot 21^7) \bmod 23 = (15 \cdot 10) \bmod 23 = 12.$$

Теперь *A* посылает к *B* зашифрованное сообщение в виде пары чисел (*r*, *c*) = (*17, 12*).

Абонент *B* вычисляет по (10.27)

$$M' = (c \cdot r^{p-1-d_B}) \bmod p = (12 \cdot 17^{23-1-13}) \bmod 23 =$$

$$= (12 \cdot 17^{9}) \bmod 23 = (12 \cdot 7) \bmod 23 = 15.$$

Из этого видно, что *B* смог расшифровать переданное сообщение.

Ясно, что по аналогичной схеме могут передавать сообщения все абоненты в сети. Заметим, что любой абонент, знающий открытый ключ абонента *B*, может посылать ему сообщения, зашифрованные с помощью открытого ключа e_B. Но только абонент *B*, и никто другой, может расшифровать эти сообщения, используя известный только ему секретный ключ d_B. Отметим также, что объем шифротекста в два раза превышает объем сообщения, но требуется только одна передача данных (при условии, что таблица с открытыми ключами заранее известна всем абонентам).

10.6. КРИПТОГРАФИЧЕСКАЯ СИСТЕМА RSA

Криптографическая система *RSA* названа в честь его разработчиков *Ривеста, Шамира* и *Адлемана*. Эта криптографическая система до сих пор является одной из наиболее широко используемых [17, 18, 24, 35].

Как указано выше, что криптографическая система *Шамира* полностью решает задачу обмена сообщениями, закрытыми для прочтения, в случае, когда абоненты могут пользоваться только открытыми линиями связи. Однако при этом сообщение пересылается три раза от одного абонента к другому, что является недостатком. Криптографическая система *Эль-Гамаля* позволяет решить ту же задачу за одну пересылку данных, но объем передаваемых зашифрованных данных в два раза превышает объем передаваемого сообщения. Система *RSA* лишена подобных недостатков. Интересно то, что она базируется на другой односторонней функции, отличной от дискретного логарифма. Кроме того, в криптографической системе *RSA* появляется изобретение современной криптографии – *односторонней функцией* с “*лазейкой*” (*trapdoor function*).

Эта система базируется на следующих двух фактах из теории чисел:

- задача проверки числа на простоту является сравнительно легкой;

- задача разложения чисел вида $n = p \cdot q$ (p и q – простые числа) на множители является очень трудной, если мы знаем только n, а p и q – большие числа (это так называемая *задача факторизации*).

Пусть в криптографической системе *RSA* есть абоненты *A*, *B*, *C*, ... Каждый абонент выбирает случайно два больших простых числа p и q. Затем он вычисляет число

$$n = p \cdot q. \tag{10.28}$$

Число n является открытой информацией, доступной другим абонентам.

После этого абонент вычисляет число $\varphi(n) = (p-1) \cdot (q-1)$ и выбирает некоторое число $e < \varphi(n)$, взаимно простое с $\varphi(n)$, и по обобщенному алгоритму *Евклида* находит число d, такое, что

$$e \cdot d \bmod \varphi(n) = 1. \tag{10.29}$$

Вся информация, связанная с абонентами и являющаяся их открытыми и секретными ключами, представлена в табл. 10.4.

Опишем протокол *RSA*. Пусть абонент *A* хочет передать сообщение *M* абоненту *B*, причем сообщение *M* рассматривается как число, удовлетворяющее неравенству $M < n_B$ (далее индекс *B* указывает на то, что соответствующие параметры принадлежат абоненту *B*).

Таблица 10.4

Ключи пользователей в системе *RSA*

Абонент	Открытый ключ	Секретный ключ
A	e_A, n_A	d_A
B	e_B, n_B	d_B
C	e_C, n_C	d_C

Шаг 1. Абонент *A* зашифровывает сообщение по формуле

$$C = (M^{e_B}) \bmod n_B, \tag{10.30}$$

используя открытые параметры абонента *B*, и пересылает e по открытой линии.

Шаг 2. Абонент *B*, получивший зашифрованное сообщение, вычисляет

$$M' = (C^{d_B}) \bmod n_B . \qquad (10.31)$$

Схема криптографической системы *RSA*, которая показывает процессы шифрования данных при их передачи от абонента *A* к абоненту *B*, представлена на рис. 10.4.

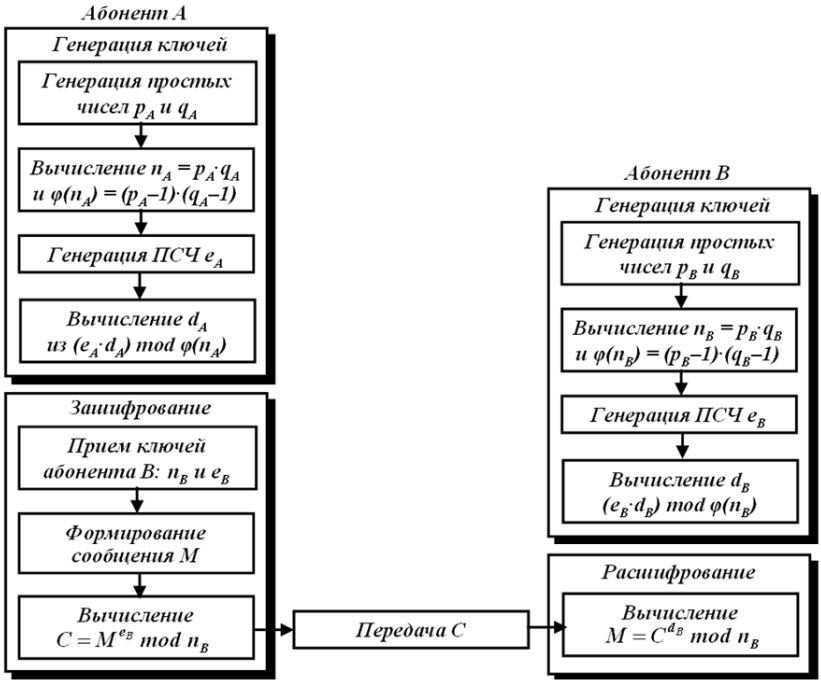


Рис. 10.4. Схема криптографической системы *RSA*

Схема криптографической системы *RSA*, которая показывает процессы шифрования данных при их передачи от абонента *B* к абоненту *A*, аналогична представленной, только в этом случае инициатором передачи сообщения будет абонент *B*.

Утверждение 10.7. Для описанного протокола $M' = M$, т.е. абонент *B* получает исходящее от абонента *A* сообщение.

Доказательство. По построению протокола

$$M' = (C^{d_B}) \bmod n_B = (M^{e_B \cdot d_B}) \bmod n_B .$$

Равенство (10.29) означает, что для некоторого k

$$e_B \cdot d_B = k \cdot \varphi_B + 1.$$

Согласно утверждению 10.5

$$\varphi_B = (p_B - 1) \cdot (q_B - 1) = \varphi(n_B),$$

где $\varphi(n_B)$ – функция *Эйлера*.

Отсюда и из теоремы 10.8 следует

$$M' = (M^{e_B \cdot d_B}) \bmod n_B = (M^{k \cdot \varphi(n_B)+1}) \bmod n_B = M .$$

Утверждение 10.8 (свойства протокола *RSA*).

1) протокол обеспечивает зашифрование и расшифрование информации корректно;

2) злоумышленник, перехватывающий все сообщения и знающий всю открытую информацию, не сможет найти исходное сообщение при больших p и q.

Доказательство. Первое свойство протокола следует из утверждения 10.8. Для доказательства второго свойства заметим, что злоумышленник знает только открытые параметры n и e. Для того чтобы найти d, он должен знать значение $\varphi(n) = (p-1) \cdot (q-1)$, а для этого, в свою очередь, ему требуется знать p и q. Вообще говоря, он может найти p и q, разложив n на множители, однако это трудная задача (см. подразделе 10.2). Отметим, что выбор больших случайных p и q возможен за приемлемое время, так как справедлив пункт 1.

Односторонняя функция $y = x^d \bmod n$, применяемая в системе *RSA*, обладает так называемой "*лазейкой*", позволяющей легко вычислить обратную функцию $x = \sqrt[d]{y} \cdot \bmod n$, если известно разложение n на простые множители. Действительно, легко вычислить $\varphi(n) = (p-1) \cdot (q-1)$, а затем $d = e^{-1} \bmod \varphi(n)$. Если p и q неизвестны, то вычисление значения обратной функции практически невозможно, а найти p и q по n очень трудно, т.е. знание p и q – это "*лазейка*" или "*потайной ход*". Такие односторонние функции с лазейкой находят применение и в других разделах криптографии.

Отметим, что для схемы *RSA* важно, чтобы каждый абонент выбирал собственную пару простых чисел p и q, т.е. все модули n_A, n_B, n_C, … должны быть различны (в противном случае один абонент мог бы читать зашифрованные сообщения, предназначенные для

другого абонента). Однако этого не требуется от второго открытого параметра e. Параметр e может быть одинаковым у всех абонентов. Часто рекомендуется выбирать $e = 3$ (при соответствующем выборе p и q, см. [7, 17]). Тогда зашифрование выполняется максимально быстро, всего за два умножения.

Пример 9.17. Допустим, абонент A хочет передать абоненту B сообщение $M = 15$. Пусть абонент B выбрал следующие параметры:

$$p_B = 3,\ q_B = 11,\ n_B = 33,\ e_B = 3,$$

e_B взаимно просто с $\varphi(33) = 20$.

Найдем d_B с помощью обобщенного алгоритма *Евклида*:

Секретный ключ абонента B – d_B с помощью обобщенного алгоритма Евклида будет равняться: $d_B = 7$.

Проверим: $e_B \cdot d_B \bmod \varphi(n) = 3 \cdot 7 \bmod 20 = 1$.

Зашифровываем M по формуле (10.30):

$$C = M^{e_B} \bmod n_B = 15^3 \bmod 33 = 15^2 \cdot 15 \bmod 33 =$$

$$= 27 \cdot 15 \bmod 33 = 9.$$

Число $C = 9$ абонент A передает абоненту B по открытому каналу связи. Только абоненту B знает $d_B = 7$, поэтому он расшифровывает принятое сообщение, используя (10.31):

$$M' = C^{d_B} \bmod n_B = 9^7 \bmod 33 = (9^2)^2 \cdot 9^2 \cdot 9 \bmod 33 =$$

$$= 15^2 \cdot 15 \cdot 9 \bmod 33 = 15.$$

Таким образом, абонент B расшифровал сообщение абонента A.

Рассмотренная система невскрываема при больших p и q, но обладает следующим недостатком: абонент A передает сообщение абоненту B, используя открытую информацию абонента B (числа n_B и c_B). Злоумышленник не может читать сообщения, предназначенные для B, однако он может передать сообщение к B от имени A. Избежать этого можно, используя более сложные протоколы, например, следующий.

Абонент A хочет передать абоненту B сообщение M. Сначала абонент A вычисляет число $C = M^{d_A} \bmod n_A$. Злоумышленник не может этого сделать, так как d_A секретно. Затем A вычисляет число

$F = C^{e_B} \bmod n_B$ и передает F к B. B получает F и вычисляет последовательно числа

$$U = F^{d_B} \bmod n_B \quad \text{и} \quad W = U^{e_A} \bmod n_A .$$

В результате абонент B получает сообщение W Как и в исходной схеме *RSA*, злоумышленник не может прочитать переданное сообщение, но здесь, в отличие от *RSA*, он не может также послать сообщение от имени A (поскольку не знает секретного d_A).

Здесь возникает новая ситуация. B знает, что сообщение пришло от A, т.е. A как бы "*подписал*" его, зашифровав своим секретным d_A. Это пример так называемой электронной или цифровой подписи. Она – одно из широко используемых на практике изобретений современной криптографии.

10.7. КРИПТОГРАФИЧЕСКАЯ СИСТЕМА РАБИНА

Безопасность *криптографической системы Рабина* опирается на сложность поиска квадратных корней по модулю составного числа. С позиции теории чисел – эта задача аналогична задаче факторизации модуля. Рассмотрим одну из реализаций системы. Представим, что пользователи A и B желают обмениваться зашифрованными сообщениями между собой. Тогда по схеме *Рабина*:

1. Пользователи A и B генерируют для себя открытые и закрытые ключи, для чего:

- каждый из пользователей выбирает по два больших простых числа p_A (p_B) и q_A (q_B) (далее алгоритм расшифрования будет особенно простым, если эти числа сравнимы с числом *3* по модулю *4*, то есть $p_A \bmod 4 = 3$, $q_A \bmod 4 = 3$, $p_B \bmod 4 = 3$ и $q_B \bmod 4 = 3$). Числа p_A (p_B) и q_A (q_B) будут закрытыми ключами системы (пользователей A и B);
- каждый из пользователей определяет свои открытые ключи: $n_A = p_A \cdot q_A$ и $n_B = p_B \cdot q_B$;
- пользователи A и B обмениваются открытыми ключами.

2. Перед зашифрованием сообщений абоненты разбивают их на блоки m_i, длина которых меньше открытого ключа n_A (если идет передача сообщения от пользователя B к пользователю A) или n_B (если идет передача сообщения от пользователя A к пользователю B). Будем полагать, что $M < n_A$ или $M < n_B$.

3. Для зашифрования сообщений абоненты *A* или *B* осуществляют преобразование данных согласно уравнению зашифрования

$$C = E_n(M) = M^2 \bmod n \qquad (10.32)$$

и после этого пересылают *C* другому абоненту *B* или *A*.

4. Из выражения (10.32) следует, что *M* – квадратный корень из числа *C* по модулю *n*. Если $gcd(C,n) = 1$ (числа *C* и *n* взаимно простые), то каждый квадратичный остаток имеет в мультипликативной группе остатков Z_n^* ровно четыре разных корня [7, 24]. Поскольку получатель знает множители *p* и *q* числа *n*, то он решает два сравнения. Прежде всего, он вычисляет

$$\begin{aligned} r_1 &= C^{(p+1)/4} \bmod p; \quad r_2 = -C^{(p+1)/4} \bmod p; \\ r_3 &= C^{(q+1)/4} \bmod q; \quad r_4 = -C^{(q+1)/4} \bmod q; \end{aligned} \qquad (10.33)$$

$$a = q\cdot(q^{-1} \bmod p); \quad b = p\cdot(p^{-1} \bmod q). \qquad (10.34)$$

Четыре возможных корня из числа *C* по модулю *n* – это

$$\begin{aligned} M_1 &= (a\cdot r_1 + b\cdot r_3) \bmod n; \quad M_2 = (a\cdot r_2 + b\cdot r_4) \bmod n; \\ M_3 &= (a\cdot r_1 + b\cdot r_4) \bmod n; \quad M_4 = (a\cdot r_2 + b\cdot r_3) \bmod n. \end{aligned} \qquad (10.35)$$

Схема криптографической системы *Рабина*, которая показывает процессы шифрования данных при их передачи от абонента *B* к абоненту *A*, представлена на рис. 10.5.

Схема криптографической системы *Рабина*, которая показывает процессы шифрования данных при их передачи от абонента *A* к абоненту *B*, аналогична представленной, только в этом случае инициатором передачи сообщения будет абонент *A*.

Несложно заметить, что в криптографической системе *Рабина* шифрующее отражение не инъективное (не взаимно однозначное). После вычисления всех четырех корней из них выбирают тот, который является числовым эквивалентом передаваемого сообщения. Если сообщение написано обычным языком, то выбрать правильное *M* нетрудно. Когда же шифровалась совокупность случайных битов, способа определения правильного *M*, не существует.

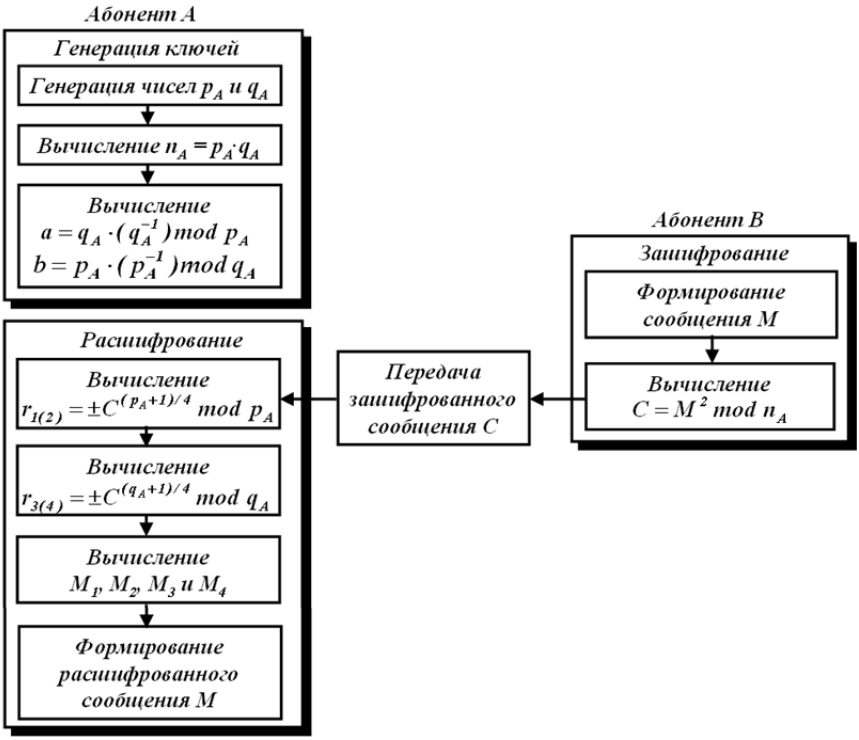


Рис. 10.5. Схема криптографической системы *Рабина*

Пример 10.18. Сгенерировать ключи и зашифровать с помощью криптографической системы *Рабина* сообщение "*протокол*" при передачи его от пользователя *B* к пользователю *A*.

Решение. Пусть закрытый ключ пользователя *A* образует пара простых чисел $p_A = 79$ и $q_A = 83$, тогда открытый ключ – число $n_A = p_A \cdot q_A = 6557$. Каждую букву открытого сообщения заменим ее номером в русском алфавите

"*протокол*" => 15 16 14 18 14 10 14 11.

Приведенная числовая последовательность – это открытое сообщение, которое записано в цифровой форме. Разобьем последовательность на четыре блока m_i, каждый из которых – некоторое натуральное число, которое должно быть меньше, чем число $n_A = 6557$:

$$M => 1516 - 1418 - 1410 - 1411.$$

Зашифруем открытое сообщение *M* по формуле (10.32):

$$C_1 = E_{n_A}(m_1) \bmod n_A = 1516^2 \bmod 6557 = 3306;$$

$$C_2 = E_{n_A}(m_2) \bmod n_A = 1418^2 \bmod 6557 = 4282;$$

$$C_3 = E_{n_A}(m_3) \bmod n_A = 1410^2 \bmod 6557 = 1329;$$

$$C_4 = E_{n_A}(m_4) \bmod n_A = 1411^2 \bmod 6557 = 4150.$$

В результате получаем зашифрованное сообщение

$$C \Rightarrow 3306 - 4282 - 1329 - 4150.$$

Пример 10.19. Пользователь *A* сгенерировал собственную пару ключей криптографической системы *Рабина*: закрытый ключ – простые числа $p_A = 59$ и $q_A = 67$, которые он хранит в тайне, и открытый ключ – общедоступное число $n_A = p_A \cdot q_A = 3953$. Расшифровать принятое зашифрованное сообщение $C \Rightarrow 1108 - 1756$ от пользователя *B*.

Решение. Расшифрование – это не что иное, как получение квадратного корня из чисел $C_1 = 1108$ и $C_2 = 1756$ по модулю $n_A = 3953$ и сводится к получению корней с простыми модулями $p_A = 59$ и $q_A = 67$.

Вычислим

$$r_1 = C_1^{(p_A+1)/4} \bmod p_A = 1108^{(59+1)/4} \bmod 59 = 20;$$

$$r_2 = -C_1^{(p_A+1)/4} \bmod p_A = -1108^{(59+1)/4} \bmod 59 = -20 \bmod 79 = 39;$$

$$r_3 = C_1^{(q_A+1)/4} \bmod q_A = 1108^{(67+1)/4} \bmod 67 = 6;$$

$$r_4 = -C_1^{(q_A+1)/4} \bmod q_A = -1108^{(67+1)/4} \bmod 67 = -6 \bmod 79 = 61.$$

Полученные решения скомбинируем между собой в системы и по китайской теореме об остатках определим значение квадратных корней. $gcd(p_A, q_A) = 1$ и по расширенному алгоритму *Евклида*:

$$1 = 37 \cdot 67 + 25 \cdot 59,$$

т.е. $(q_A^{-1} \cdot q_A) \bmod p_A = 1$, откуда $q_A^{-1} = 37$; $(p_A^{-1} \cdot p_A) \bmod q_A = 1$, откуда $p_A^{-1} = 25$.

В соответствии со значениями q_A, q_A^{-1}, p_A и p_A^{-1}:

$$a = q_A(q_A^{-1} \bmod p_A) = 67 \cdot 37 = 2479;$$

$$b = p_A(p_A^{-1} \bmod q_A) = 59 \cdot 25 = 1475.$$

Первый корень M найдем из (10.33), (10.34) и (10.35). Откуда следует:

$$M_1 = (a \cdot r_1 + b \cdot r_3) \bmod n_A = 3088.$$

Второй корень, противоположный до найденного первого корня $M_1 = 3088$ найдем также из (10.33), (10.34) и (10.35). Откуда следует:

$$M_2 = (a \cdot r_2 + b \cdot r_4) \bmod n_A = 0865.$$

Можно убедиться, что корни M_1 и M_2 противоположны, т.е. $(M_1 + M_2) \bmod n_A = 0$:

$$(M_1 + M_2) \bmod n_A = (3088 + 0865) \bmod 3953 = 3953 \bmod 3953 = 0.$$

Третий корень M найдем из (10.33) и (10.34). Откуда следует:

$$M_3 = (a \cdot r_1 + b \cdot r_4) \bmod n_A = 1200.$$

Четвертый корень, противоположный до найденного третьего корня $M_3 = 1200$ найдем также из (10.33) и (10.34). Откуда следует:

$$M_4 = (a \cdot r_2 + b \cdot r_3) \bmod n_A = 2753.$$

Можно убедиться, что корни M_3 и M_4 противоположны, т.е. $(M_3 + M_4) \bmod n_A = 0$:

$$(M_3 + M_4) \bmod n_A = (1200 + 2753) \bmod 3953 = 3953 \bmod 3953 = 0.$$

Следовательно, из числа $C_1 = 1108$ получено четыре корня по модулю *3953*: *3088*, *0865*, *1200* и *2753*. Аналогично получим четыре корня из второго числа $C_2 = 1756$ полученного зашифрованного сообщения. Это будет: *1215*, *2738*, *2336*, *1617*. В этом случае корни *1215* и *1617* – числовые эквиваленты сообщения русского алфавита, мощность которого *32* буквы.

Поскольку числу *12* русского алфавита соответствует буква “*м*”, *00* – “*а*”, *15* – “*п*”, *16* – “*р*”, *17* – “*с*”, то первые две буквы расшифро-

ванного сообщения будут – “*ма*”, а вторые две буквы – либо “*мп*”, либо “*рс*”. Следовательно, расшифрованное сообщение соответствует либо “*мамп*”, либо “*марс*”. Смысловую нагрузку в русском языке несет только слово “*марс*”.

10.8. МЕТОДЫ ВЗЛОМА КРИПТОГРАФИЧЕСКИХ СИСТЕМ, ОСНОВАННЫХ НА ДИСКРЕТНОМ ЛОГАРИФМИРОВАНИИ

10.8.1. Постановка задачи

Для построения надежной криптографической системы необходимо принимать во внимание те методы взлома, которые может применить злоумышленник, и выбирать параметры криптографической системы (в частности, размерности чисел) так, чтобы сделать эти методы практически нереализуемыми. В данном подразделе рассмотрим два таких метода, для того чтобы дать читателю некоторое представление об этой “*таинственной*” области.

Как уже было отмечено, многие рассматриваемые криптографические системы основываются на односторонней функции

$$y = a^x \bmod p. \qquad (10.36)$$

y что можно вычислить, если даны a, x и p, затратив не более, чем $2 \cdot \log x$ операций (утверждение 10.1). Однако отыскание x по известным a, y и p, т.е. вычисление дискретного логарифма, – задача намного более трудная.

Как уже было показано при рассмотрении криптосистемы *Шамира* (см. (10.23)), на основании теоремы *Ферма* при возведении в степень по простому модулю p показатели степени приводятся по модулю $p–1$. Поэтому достаточно рассматривать только показатели x, удовлетворяющие неравенству $0 \le x \le p–1$.

Обозначим через t_y число операций умножения, необходимых для вычисления y в (10.1) по a, x и p, и будем для краткости называть t_y временем вычисления. Время возведения в степень по алгоритмам из подраздела 10.1 не больше $2 \cdot \log x$, причем $x < p$. Отсюда

$$t_y \le 2 \cdot \log x \qquad (10.37)$$

при любом показателе степени x.

Теперь перейдем к задаче отыскания x в (10.36) по данным a, y и p. Сначала оценим сложность прямого перебора. Для этого можно было бы сначала вычислить a^1 и проверить, верно ли равенство $a^1 \bmod p = y$. Если нет, то проверяем $a^2 \bmod p = y$, если нет, то $a^3 \bmod p = y$ и т.д. до $a^{p-1} \bmod p = y$. В среднем потребуется $(p-1)/2$ раз умножать на a и проверять равенство. Таким образом, время прямого перебора

$$t_{п.п.} \approx p/2.$$

В описываемом ниже методе "*шаг младенца, шаг великана*" время отыскания x существенно меньше

$$t_{ш.м\ ш.в.} \approx 2 \cdot \sqrt{p}\ ,$$

а в методе исчисления порядка это время еще меньше:

$$t_{и.п.} \approx c_1 \cdot 2^{c_2 \sqrt{\log p \cdot \log(\log p)}}\ ,$$

где c_1, c_2 – некоторые положительные константы.

Чтобы сделать сравнение более наглядным, выразим время вычисления через длину числа p в (10.36). Обозначим эту длину в битах через n. При вычислениях по модулю p имеем $n \approx \log p$. Поэтому порядок трудоемкости (в смысле количества операций) упомянутых алгоритмов будет следующий:

$$t_y \approx n, \quad t_{п.п.} \approx 2^{n-1}, \quad t_{ш.м\ ш.в.} \approx 2^{n/2}, \quad t_{и.п.} \approx 2^{c_2 \sqrt{n \cdot \log n}}\ ,$$

где $\approx$ означает "*пропорционально*".

Как видим, что количество операций при возведении в степень растет линейно с ростом длины числа n, а время решения обратной задачи различными методами растет экспоненциально либо субэкспоненциально (для метода исчисления порядка). Вопрос о существовании более быстрых алгоритмов для вычисления дискретных логарифмов, как и для решения других обратных задач, возникающих в криптографическом анализе, остается открытым.

10.8.2. Метод "*Шаг младенца, шаг великана*"

В открытой литературе этот метод был впервые описан *Д. Шенксом* (*Daniel Shanks*); ссылки на него известны с 1973 года [20]. Это был один из первых методов, который показал, что задача вычисления дискретного логарифма может быть решена значительно быстрее, чем методом перебора. Перейдем к описанию этого метода отыскания x в (10.36).

Шаг 1. Сначала берем два целых числа m и k, такие, что

$$m \cdot k > p. \tag{10.38}$$

Шаг 2. Вычислим два ряда чисел

$$y,\ a \cdot y,\ a^2 \cdot y,\ \ldots,\ a^{m-1} \cdot y \bmod p; \tag{10.39}$$

$$a^m,\ a^{2 \cdot m},\ \ldots,\ a^{k \cdot m} \bmod p. \tag{10.40}$$

Все вычисления проводятся по модулю p.

Шаг 3. Найдем такие i и j, для которых выполняется равенство

$$a^{i \cdot m} = a^j \cdot y. \tag{10.41}$$

Утверждение 10.9. Число

$$x = i \cdot m - j \tag{10.42}$$

является решением уравнения (10.36). Кроме того, целые числа i и j, удовлетворяющие (10.41), существуют.

Доказательство. Справедливость (10.42) следует из приводимой ниже символической цепочки равенств, где все вычисления даны по модулю p, а деление соответствует умножению на обратный элемент:

$$a^x = a^{i\ m-j} = \frac{a^{i\ m}}{a^j} = \frac{a^{i\ m} y}{a^j y} = \frac{a^{i\ m} y}{a^{i\ m}} = y\,.$$

Докажем теперь, что числа i и j, удовлетворяющие (10.41), существуют. Для этого сведем все числа вида (10.42) в табл. 10.5.

Видим, что в таблице содержатся все числа от *1* до $k \cdot m$. Значит, из (10.42) следует, что в таблице содержатся все числа от *1* до p. Таким образом, любой показатель степени $x < p$ будет содержаться

в таблице, т.е число x, удовлетворяющее (10.36), может быть представлено в виде (10.42) и всегда найдется в таблице, поэтому уравнение (10.41) всегда имеет решение.

Таблица 10.5

Распределение чисел вида $i \cdot m - j$

	$j = 0$	$j = 1$	$j = 2$	...	$j = m - 1$
$i = 1$	m	$m - 1$	$m - 2$	...	1
$i = 2$	$2 \cdot m$	$2 \cdot m - 1$	$2 \cdot m - 2$	...	$m + 1$
...	...	...	...	...	...
$i = k$	$k \cdot m$	$k \cdot m - 1$	$k \cdot m - 2$	...	$(k - 1) \cdot m + 1$

Пример 10.20. Найдем решение уравнения $2^x \bmod 23 = 9$, используя метод "*шаг младенца, шаг великана*".

Выберем m и k. Пусть $m = 6$ и $k = 4$. Видим, что (10.38) выполняется. Вычислим числа (10.39) и (10.40)

9, 18, 13, 3, 6, 12;
18.

Дальнейшие вычисления не проводим, так как уже нашлись одинаковые числа в (10.39) и (10.40) при $i = 1, j = 1$. По (10.42) получаем

$$x = i \cdot m - j = 1 \cdot 6 - 1 = 5.$$

Проверим: $2^5 \bmod 23 = 9$. Действительно, $x = 5$ есть решение.

Объясним происхождение названия рассмотренного метода. Известно, что в криптографии p – большое простое число, значит m и k тоже большие. В ряду (10.39) степень увеличивается на 1 (шаг младенца), а в ряду (10.40) степень увеличивается на m (шаг великана). Оценим сложность этого метода.

Утверждение 10.10. Время вычислений по данному методу при больших p удовлетворяет неравенству [34]

$$t_{ш.м.ш.в.} \le const \cdot \sqrt{p} \cdot \log^2 p\,. \qquad (10.43)$$

Здесь речь идет о полном времени вычислений, а не о числе умножений.

Доказательство. Можем взять

$$k = m = \left\lfloor \sqrt{p} \right\rfloor + 1\,, \qquad (10.44)$$

так что, очевидно, (10.38) выполняется. Тогда в (10.39) и (10.40) потребуется не более $2\cdot\sqrt{p}$ операций умножения. Известно, что для *обычных* (*школьных*) методов умножения и деления время вычисления результата для двух *r*-значных чисел пропорционально r^2. У нас все числа берутся из множества $\{ 1, \ldots, p \}$, значит, $r < \log p$, и время вычисления пропорционально $\log^2 p$. Отсюда сразу получаем время, затраченное на вычисление рядов (10.39) и (10.40). Однако не учтены все этапы алгоритма и время, требуемое для нахождения равных чисел в этих рядах. При больших k и m это далеко не простая задача. Она может быть решена следующим образом: сначала каждому числу припишем его номер в ряду и еще один бит, в котором указана принадлежность к ряду (10.39) или (10.40), затем преобразуем обе последовательности в список и отсортируем (упорядочим по величине). Длина общего ряда равна $k + m \approx 2\cdot\sqrt{p}$. Для лучших методов сортировки требуется $S\cdot\log S$ операций сравнения, где S – число элементов в списке (см., например, [34]). В нашем случае $S = 2\cdot\sqrt{p}$ и, следовательно, требуется

$$2\sqrt{p}\cdot\log(2\cdot\sqrt{p}) \approx \sqrt{p}\cdot\log p$$

операций сравнения над словами длины $\log p$ бит, т.е. всего требуется порядка $\sqrt{p}\cdot\log^2 p$ операций. После сортировки объединенного ряда его надо просмотреть и найти два равных числа из разных рядов (10.39), (10.40), используя битовый признак. Таким образом, суммируя время вычисления на всех этапах, получаем (10.43).

10.8.3. Алгоритм исчисления порядка

Основные идеи *алгоритма исчисления порядка* (*index-calculus algorithm*) были известны в теории чисел еще с 20-х годов XX века. Однако только в 1979 году *Адлеман*, один из создателей *RSA*, указал на этот алгоритм как на средство решения уравнения (10.36) и исследовал его трудоемкость. В настоящее время алгоритм исчисления порядка и его улучшенные варианты дают наиболее быстрый способ вычисления дискретных логарифмов в уравнениях типа (10.36).

Для удобства описания алгоритма введем следующее понятие.

Определение 10.7. Число n называется p-гладким, если оно разлагается только на простые множители, меньшие либо равные p.

Пример 10.21. Числа *15, 36, 45, 270, 2025* являются *5*-гладкими (в их разложении участвуют только множители *2, 3* и *5*).

Перейдем непосредственно к описанию алгоритма.

Шаг 1. Формируем множество базовых множителей

$$S = \{ p_1, p_2, \ldots, p_t \},$$

состоящее из первых t простых чисел (замечание о выборе значения t будет дано ниже).

Шаг 2. Задавая последовательно значения $k = 1,2,3,\ldots$, находим $t + \varepsilon$ (ε – небольшое целое число, см. ниже) p^t-гладких чисел вида $a^k \bmod p$, проверяя гладкость путем деления на элементы множества S. Каждое из найденных p^t-гладких чисел записывается через произведение базовых множителей:

$$a^k \bmod p = \prod_{i=1}^{t} p_i^{c_i}, \quad c_i \geq 0. \tag{10.45}$$

Для каждого значения k получаем свой набор чисел c_i.

Шаг 3. Переходим к логарифмам в (10.45):

$$k = \sum_{i=1}^{t} c_i \cdot \log_a p_i, \tag{10.46}$$

для каждого p^t-гладкого числа, найденного на шаге 2. Получена система из $t+\varepsilon$ уравнений вида (10.46) с t неизвестными. В качестве неизвестных здесь выступают величины $\log_a p_i$, при этом число уравнений на ε больше числа неизвестных, что повышает вероятность получения решения системы в случае, если некоторые из уравнений окажутся линейно зависимыми. Решаем систему методами линейной алгебры, проводя все вычисления по модулю $p–1$ (напомним, что показатели степени, а следовательно и логарифмы, приводятся по модулю $p–1$). В результате получаем значения логарифмов чисел из множества S: $\log_a p_1, \log_a p_2, \ldots, \log_a p_t$.

Шаг 4. Случайным образом выбирая r, находим p^t-гладкое число вида $(y \cdot a^r)$:

$$y \cdot a^r \bmod p = \prod_{i=1}^{t} p_i^{\varepsilon_i}, \quad \varepsilon_i \geq 0. \tag{10.47}$$

Шаг 5. Логарифмируя (10.47), получаем конечный результат

$$x = \log_a y = \left(\sum_{i=1}^{t} e_i \log_a p_i - r \right) \bmod (p-1), \tag{10.48}$$

величина r вычитается из всей суммы, а не из каждого слагаемого.

Справедливость описанного метода довольно очевидна из построения алгоритма, а его эффективность связана со следующим наблюдением. Если выбирать случайно наугад число из бесконечного множества целых чисел, то с вероятностью *1/2* оно делится на *2*, с вероятностью *1/3* – на *3*, с вероятностью *1/5* – на *5* и т.д. Поэтому можно ожидать, что в промежутке от *1* до $p-1$ существует достаточно много чисел, в разложении которых участвуют только маленькие простые множители из множества S. Именно такие числа отыскиваются на шагах 2 и 4 алгоритма. Чем больше t, т.е. количество простых множителей в S, тем меньше неудач при поиске гладких чисел происходит на шагах 2 и 4, т.е. эти шаги выполняются быстрее. Однако при больших t резко увеличивается трудоемкость шага 3, когда приходится решать систему из $t+\varepsilon$ уравнений. Нахождение значения дающего минимальное общее время вычислений, обычно может быть выполнено с использованием численных методов. Аналитические выражения получить довольно трудно. Параметр ε принимается равным небольшому целому числу для того, чтобы увеличить вероятность существования решения системы уравнений на шаге 3. Дело в том, что полученная система может содержать линейно зависимые уравнения (как это будет показано в приводимом ниже примере). Считается, что при больших p значение ε порядка *10* гарантирует существование единственного решения системы с высокой вероятностью (см. [7, 34]).

Если же все-таки полученная система имеет бесконечно много решений, то необходимо вернуться к шагу 2 и использовать другие значения k.

Адлеман показал, что при оптимальном значении t для трудоемкости алгоритма имеем

$$t_{u.n.} < c_1 \cdot 2^{(c_2 + o(1)) \cdot \sqrt{\log p \cdot \log(\log p)}},$$

где c_1, c_2 – некоторые положительные константы.

Пример 10.22. Решим с помощью алгоритма исчисления порядка уравнение

$$37 = 10^x \bmod 47. \tag{10.49}$$

Имеем $y = 37$, $a = 10$, $p = 47$. Возьмем множество базовых множителей $S = \{ 2, 3, 5 \}$, $t = 3$, и примем $\varepsilon = 1$, т.е. будем строить систему из четырех уравнений. Обозначим логарифмы чисел из S через u_1, u_2 и u_3 соответственно, например, $u_3 = log_{10}5 \bmod 47$. Выполнен первый шаг алгоритма, перейдем ко второму.

Проведем поиск четырех 5-гладких чисел:

$$\begin{aligned}
10^1 \bmod 47 &= 10 = 2\cdot 5, && \vee\\
10^2 \bmod 47 &= 6 = 2\cdot 3, && \vee\\
10^3 \bmod 47 &= 13 = 13, &&\\
10^4 \bmod 47 &= 36 = 2\cdot 2\cdot 3\cdot 3, && \vee\\
10^5 \bmod 47 &= 31 = 31, &&\\
10^6 \bmod 47 &= 28 = 2\cdot 2\cdot 7, &&\\
10^7 \bmod 47 &= 45 = 3\cdot 3\cdot 5. && \vee
\end{aligned}$$

Найдено четыре 5-гладких числа, соответствующих степеням *1*, *2*, *4* и *7*.

Начинаем третий шаг алгоритма. Перейдем к логарифмам и составим систему уравнений из равенств, отмеченных на предыдущем шаге символом $\vee$:

$$1 = u_1 + u_3, \tag{10.50}$$

$$2 = u_1 + u_2, \tag{10.51}$$

$$4 = 2\cdot u_1 + 2\cdot u_2, \tag{10.52}$$

$$7 = 2\cdot u_2 + u_3. \tag{10.53}$$

Видно, что в полученной системе уравнения (10.51) и (10.52) линейно зависимы, так что мы не зря нашли четвертое гладкое число. Чтобы решить систему, вычтем (10.50) из (10.51). Получим

$$1 = u_2 - u_3. \tag{10.54}$$

Прибавим (10.54) к (10.53). Получим

$$8 = 3\cdot u_2 - u_3. \tag{10.55}$$

Из (10.50) непосредственно находим u_2:

$$u_2 = (8/3) \bmod 46 = 8 \cdot 3^{-1} \bmod 46 = 8 \cdot 31 \bmod 46 = 18.$$

Можно сделать проверку, вычислив $10^{18} \bmod 47 = 3$, таким образом, u_2 – действительно логарифм числа 3. Теперь из (10.54) находим u_3:

$$u_3 = u_2 - 1 = 18 - 1 = 17.$$

Действительно, $10^{17} \bmod 47 = 5$.

Наконец, из (10.51) находим u_1:

$$u_1 = 2 - u_2 = (2 - 18) \bmod 46 = -16 \bmod 46 = 30, \; 10^{30} \bmod 47 = 2.$$

Итак, теперь известны логарифмы чисел из S. Самый трудоемкий этап алгоритма позади. Переходим к четвертому шагу. Начнем с $k = 3$:

$$37 \cdot 10^3 \bmod 47 = 37 \cdot 13 \bmod 47 = 11,$$
$$37 \cdot 10^4 \bmod 47 = 37 \cdot 36 \bmod 47 = 16 = 2 \cdot 2 \cdot 2 \cdot 2. \quad \vee$$

Переходим в последнем равенстве к логарифмам (это пятый шаг) и получаем конечный результат:

$$\log_{10} 37 = 4 \log_{10} 2 - 4 = (4 \cdot 30 - 4) \bmod 46 = 24.$$

Таким образом найдено решение уравнения (10.49) $x = 24$. Можно сделать проверку: $10^{24} \bmod 47 = 37$.

Самым быстрым на данное время считается вариант рассмотренного алгоритма исчисления порядка, называемый *Number Field Sieve*. *Number Field Sieve* (*NFS* – общий метод решета числового поля — метод факторизации целых чисел).Этот метод использует тонкие алгебраические конструкции и довольно сложен для описания. Его трудоемкость дается оценкой

$$t_{u.n.} < c_1 \cdot 2^{(c_2 + o(1)) \cdot \sqrt[3]{\log p \log(\log p)^2}}, \qquad (10.56)$$

где c_1 и c_2 - некоторые положительные константы.

Именно этот метод диктует сегодня условия для выбора длин модулей криптосистем, стойкость которых основана на трудности вычисления дискретных логарифмов (это система *Диффи-Хеллмана*, *Шамира* и *Эль-Гамаля*). Для достижения долговременной стойкости

этих криптосистем рекомендуется брать модули длиной не менее *1024* бит [7, 20, 43].

В заключение отметим, что в учебнике не рассматриваются методы взлома криптографических систем, основанных на факторизации чисел (таких как *RSA*). Дело в том, что описание современных алгоритмов разложения числа на множители потребовало бы введения дополнительных понятий и алгоритмов из теории чисел, нигде больше в книге не используемых. Однако на сегодня [7, 34] самые быстрые методы разложения чисел на множители характеризуются такой же оценкой времени, которую дает выражение (10.56). Как следствие, для обеспечения стойкости системы *RSA* длина модуля должна также быть не менее *1024* бит (т.е. простые числа, дающие в произведении модуль *RSA*, должны быть длиной минимум по *512* бит).

Контрольные вопросы и задания

1. Дайте определение простого и составного числа. Приведите по три примера простых и составных чисел.

2. Дать определение понятия "*взаимно простые чис*ла". Привести примеры взаимно простых чисел и чисел, которые не являются взаимно простыми. Что такое инверсия по модулю *n*?

3. В чем заключается задача факторизации? Дайте определение наибольшего общего делителя.

4. С какой целью может применяться алгоритм *RSA*?

5. Опишите процесс шифрования с использованием алгоритма *RSA*.

6. Для чего может применяться алгоритм *Диффи-Хеллмана*?

7. Опишите последовательность действий при использовании алгоритма *Диффи-Хеллмана*.

8. Для каких целей может применяться алгоритм *Эль-Гамаля*?

9. Опишите последовательность действий при использовании алгоритма *Эль-Гамаля*.

10. Для каких целей может применяться алгоритм *Рабина*?

11. Опишите последовательность действий при использовании алгоритма *Рабина*.

12. Какие атаки возможны при использовании алгоритмов шифрования с открытым ключом?

13. Привести результат выражений *5*, *16*, *27*, *–4*, *–13*, *3 + 8*, *3 – 8*, *3·8* і *3·8·5*: а) по модулю *10*; б) по модулю *11*.

14. Вычислить, используя быстрые алгоритмы возведение в степень: $2^8 \bmod 10$, $2^7 \bmod 10$, $7^{19} \bmod 100$ и $7^{57} \bmod 100$.

15. Разложить на простые множители числа: *108*, *77*, *65*, *30* и *159*.

16. Определить, какие из пар чисел (*25*,*12*), (*25*,*15*), (*13*,*39*) и (*40*,*27*) взаимно просты.

17. Найти значение функции *Эйлера* $\varphi(14)$ и $\varphi(15)$.

18. Используя свойства функции *Эйлера*, вычислить $\varphi(53)$, $\varphi(21)$ и $\varphi(159)$.

19. Используя теорему *Ферма*, вычислить $3^{13} \bmod 13$, $5^{22} \bmod 11$ и $3^{17} \bmod 5$.

20. Используя теорему *Эйлера*, вычислить $3^9 \bmod 20$, $2^{14} \bmod 21$ и $2^{107} \bmod 159$.

21. С помощью алгоритма *Евклида* найти $gcd(21,12)$, $gcd(30,12)$, $gcd(24,40)$ и $gcd(33,16)$.

22. С помощью обобщенного алгоритма *Евклида* найти значения *x* и *y* в уравнениях:

а) $21 \cdot x + 12 \cdot y = gcd(21,12)$; б) $30 \cdot x + 12 \cdot y = gcd(30,12)$;

в) $24 \cdot x + 40 \cdot y = gcd(24,40)$; г) $33 \cdot x + 16 \cdot y = gcd(33,16)$.

23. Вычислить $3^{-1} \bmod 7$, $5^{-1} \bmod 8$, $3^{-1} \bmod 53$ и $10^{-1} \bmod 53$.

24. Выписать все простые числа, меньшие *100*. Какие из них соответствуют виду $p = 2 \cdot q + 1$, где *q* также простое число?

25. Найти все допустимые варианты выбора параметра g в системе *Диффи-Хеллмана* при $p = 11$.

26. Вычислить открытые ключи e_A, e_B и общий ключ K_{AB} для криптографической системы *Диффи-Хеллмана* с параметрами:

а) $p = 23$, $g = 5$, $d_A = 5$, $d_B = 7$;

б) $p = 19$, $g = 2$, $d_A = 5$, $d_B = 7$;

в) $p = 23$, $g = 7$, $d_A = 3$, $d_B = 4$;

г) $p = 17$, $g = 3$, $d_A = 10$, $x_B = 5$;

д) $p = 19$, $g = 10$, $d_A = 4$, $d_B = 8$.

27. Для криптографической системы *Шамира* с заданными параметрами *p*, e_A и d_A найти недостающие параметры и описать процесс передачи сообщения *M* от *A* к *B*:

а) $p = 19$, $e_A = 5$, $e_B = 7$, $M = 4$;

б) $p = 23$, $e_A = 15$, $e_B = 7$, $M = 6$;

в) $p = 19$, $e_A = 11$, $e_B = 5$, $M = 10$;

г) $p = 23$, $e_A = 9$, $e_B = 3$, $M = 17$;

д) $p = 17$, $e_A = 3$, $e_B = 13$, $M = 9$.

28. Для криптографической системы *Эль-Гамаля* с заданными параметрами p, g, d_B и k найти недостающие параметры и описать процесс передачи сообщения M пользователю B:

а) $p = 19$, $g = 2$, $d_B = 5$, $k = 7$, $M = 5$;

б) $p = 23$, $g = 5$, $d_B = 8$, $k = 10$, $M = 10$;

в) $p = 19$, $g = 2$, $d_B = 11$, $k = 4$, $M = 10$;

г) $p = 23$, $g = 7$, $d_B = 3$, $k = 15$, $M = 5$;

д) $p = 17$, $g = 3$, $d_B = 10$, $k = 5$, $M = 10$.

29. В системе *RSA* с заданными параметрами p_A, q_A и e_A найти недостающие параметры и описать процесс передачи сообщения M пользователю A:

а) $p_A = 5$, $q_A = 11$, $e_A = 3$, $M = 12$;

б) $p_A = 5$, $q_A = 13$, $e_A = 5$, $M = 20$;

в) $p_A = 7$, $q_A = 11$, $e_A = 7$, $M = 17$;

г) $p_A = 7$, $q_A = 13$, $e_A = 5$, $M = 30$;

д) $p_A = 3$, $q_A = 11$, $e_A = 3$, $M = 15$.

30. Пользователю A системы *RSA* с параметрами $n_A = 187$ і $e_A = 3$ передано зашифрованное сообщение $C = 100$. Злоумышленник перехватил это сообщение. Пояснить процесс взламывания данной системы *RSA* и определить открытое сообщение, которое передавалось пользователю A.

31. Сгенерировать открытый ключ и, применив криптографическую систему *Рабина*, зашифровать открытый текст: "*криптография*", $p_A = 53$, $q_A = 71$. При переходе к цифровой записи текста каждую букву заменить ее двухцифровым десятичным номером в русском алфавите (нумерацию начинать с *00*, пробел заменить числом *33*) и разбить текст на блоки из четырех цифр.

32. Пользователь A сгенерировал открытый ключ $n_A = 4189$ и с помощью криптографической системы *Рабина* получил в свой адрес зашифрованное сообщение: $C = 4076\ 2375\ 1903$. Восстановить закрытые ключи и расшифровать зашифрованное сообщение (в цифровой записи расшифрованного сообщения каждое двузнач-

ное десятичное число является номером соответствующей буквы в русском алфавите, нумерация начинается с *00* и заканчивается числом *31*).

33. Используя метод "*шаг младенца, шаг великана*", решить следующие уравнения:

а) $2^x \bmod 29 = 21$; б) $3^x \bmod 31 = 25$; в) $2^x \bmod 37 = 12$;

г) $6^x \bmod 41 = 21$; д) $3^x \bmod 43 = 11$.

34. Используя алгоритм вычисления порядка, решить такие уравнение:

а) $2^x \bmod 53 = 24$; ….б) $2^x \bmod 59 = 13$; в) $2^x \bmod 61 = 45$;

г) $2^x \bmod 67 = 41$; ….д) $7^x \bmod 71 = 41$.

ИМЕННОЙ УКАЗАТЕЛЬ

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ

Л

М

Н

О

П

Р

Э

Ю

СПИСОК ЛИТЕРАТУРЫ

1. Аграновский А.В. Практическая криптография: алгоритмы и их программирование / А.В. Аграновский, Р.А. Хади. – М.: СОЛОН-Пресс, 2009. – 256 с.: ил.

2. Бабаш А.В. Криптография / А.В. Бабаш, Г.П. Шанкин; под редакцией В.П. Шерстюка, Э.А. Применко. – М.: СОЛОН-Пресс, 2007. – 512 с.: ил.

3. Блінцов В.С. Математичні основи криптології + CD : Навчальний посібник для студ. вищих навч. закл. / В.С. Блінцов, Ю.Л. Гальчевський. – Миколаїв: Національний ун-т кораблебудування ім. адмірала Макарова, 2006. – 232 с.: іл.

4. Безпека інформаційних систем і технологій: Навч. посібник / В. І. Єсін, О. О. Кузнецов, Л. С. Сорока. – Х. : ХНУ імені В. Н. Каразіна, 2013. – 632 с. : іл.

5. Богуш В.М. Криптографічні застосування елементарної теорії чисел : Навч. посібник / В.М. Богуш, В.А. Мухачов. – К.: Державний ун-т інформаційно-комунікаційних технологій, 2006. – 126 с.: іл.

6. Введение в криптографию / Н.П. Варновский, Ю.В. Нестеренко, Г.А. Кабатянский и др.; под ред. В.В. Ященко. – М.: МЦНМО-ЧеРо, 1998. – 272 с.: ил.

7. Горбенко І.Д. Прикладна криптологія. Теорія. Практика. Застосування : монографія / І.Д. Горбенко, Ю.І. Горбенко. – Харків: Видавництво "Форт", 2012. – 880 с.: іл.

8. Горбенко І.Д. Захист інформації в інформаційно-телекомунікаційних системах : Навч. посіб. для студ. Ч. 1. Криптографічний захист інформації / І. Д. Горбенко, Т. О. Гріненко. – Х. : Харк. нац. ун-т радіоелектрон., 2004. – 368 с.: іл.

9. Грайворонський М.В. Безпека інформаційно-комунікаційних систем : Підручник / М. В. Грайворонський, О. М. Новіков. – К. : Видавнича група BHV, 2009. – 608 с.: іл.

10. Задірака В.К. Комп'ютерна криптологія : Підручник / В.К. Задірака, О.С. Олексюк. – К.: Тернопільська академія народного господарства; НАН України; Інститут кібернетики ім. В.М. Глушкова, 2002. – 504 с.: іл.

11. Захист інформації в мережах передачі даних / Юдін О.К., Корченко О.Г., Конахович Г.Ф. – К.: Вид-во ТОВ «НВП» ІНТЕРСЕРВІС», 2009. – 716 с.: іл.

12. Защита информации в компьютерных системах и сетях / Ю.В. Романец, П.А. Тимофеев, В.Ф. Шаньгин; под ред. В.Ф. Шаньгина. – М.: Радио и связь, 2001. – 376 с.: ил.

13. Иванов М.А. Криптографические методы защиты информации в компьютерных системах и сетях / М.А. Иванов – М.: КУДИЦ-ОБРАЗ, 2001. – 363 с.: ил.

14. Коблиц Н. Курс теории чисел и криптографии / Н. Коблиц. – М.: Научное издательство ТВП, 2001. – 272 с.: ил.

15. Конеев И.Р. Информационная безопасность предприятия / И.Р. Конеев, А.В. Беляев. – СПб, БХВ-Петербург, 2003. – 752 с.: ил.

16. Корченко О.Г. Охорона конфіденційної інформації підприємства : Навч. посіб. / О. Г. Корченко, Ю. О. Дрейс. – Житомир: ЖВІ НАУ, 2011. – 172 с.: іл.

17. Коутинхо С. Введение в теорию чисел. Алгоритм RSA / С. Коутинхо. – М.: Постмаркет, 2001. – 328 с.: ил.

18. Мао В. Современная криптография: теория и практика / В. Мао; пер. с англ. – М.: Издательский дом “Вильямс”, 2005. – 768 с.: ил.

19. Математичні основи криптографії : навч. посібник / Г.В. Кузнєцов, В.В. Фомічов, С.О. Сушко, Л.Я. Фомічова. – Дніпропетровськ: Національний гірничий університет, 2004. – 391 с.: іл.

20. Математичні основи криптоаналізу : Навч. посіб. / С.О. Сушко, Г.В. Кузнецов, Л.Я. Фомичова, А.В. Корабльов. – Д. : Національний гірничий університет, 2010. – 465 с.: іл.

21. Математические и компьютерные основы криптологии : учебное пособие / Ю.С. Харин, В.И. Берник, Г.В. Матвеев, С.В. Агиевич. – Минск: Новое издание, 2003. – 382 с.: ил.

22. Математические основы криптологии : учебное пособие / Ю.С. Харин, В.И. Берник, Г.В. Матвеев. – Минск: БГУ, 1999. – 319 с.: ил.

23. Методи та алгоритми симетричної криптографії: навч. пос. / Кузнецов О.О., Євсеев С.П., Смірнов О.А., Мелешко Є.В., Король О.Г. – Кіровоград: Вид. КНТУ, 2012. – 316 с.: іл.

24. Молдовян Н.А. Криптография с открытым ключом / Н.А. Молдовян, А.А. Молдовян. – СПб.: БХВ-Петербург, 2005. – 288 с.: ил.

25. Мукачев В.А. Методы практической криптографии / В.А. Мукачев, А.А. Хорошко. – К.: ООО "Полиграф-Консалтинг", 2005. – 215 с.: ил.

26. Ожиганов А.А. Основы криптоанализа симметричных шифров : учебное пособие / Ожиганов А.А.– СПб: СПбГУ ИТМО, 2008. – 44 с.: ил.

27. Основы криптографии : учебное пособие / А.П. Алферов, А.Ю. Зубов, А.С. Кузьмин, А.В. Черемушкин. – М.: Гелиос АРВ, 2002. – 480 с.: ил.

28. Основы современной криптографии / С.Г. Баричев, В.В. Гончаров, В.Е. Серов. – М.: "Горячая линия-Телеком", 2001. – 154 с.: ил.

29. Панасенко С.П. Алгоритмы шифрования. Специальный справочник / С.П. Панасенко. – СПб: БХВ-Петербург, 2009. – 576 с.: ил.

30. Петров А.А. Компьютерная безопасность. Криптографические методы защиты / А.А. Петров. – М.: Издательство ДМК, 2000. – 448 с.: ил.

31. Поповский В.В. Защита информации в телекоммуникационных системах: учебник / В.В. Поповский, А.В. Персиков. – Харьков: ООО "Компания СМИТ", Т. 1. – 2006. – 238 с.: ил.

32. Поточные шифры / А.В. Асосков, М.А. Иванов, А.А. Мирский и др. — М.: КУДИЦ-ОБРАЗ, 2003. – 336 с.: ил.

33. Ростовцев А.Г. Теоретическая криптография / А.Г. Ростовцев, Е.Б. Маховенко. – СПб.: АНО НПО Профессионал, 2005. – 480 с.: ил.

34. Рябко Б.Я. Криптографические методы защиты информации: учебное пособие для вузов / Б.Я. Рябко, А.Н. Фионов. – М.: Горячая линия – Телеком, 2005. – 229 с.: ил.

35. Саломаа А. Криптография с открытым ключом : пер. с англ / А. Саломаа. – М.: Мир, 1996. – 304 с.: ил.

36. Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования ГОСТ 28147-

89. Гос. Ком. СССР по стандартам. - М., 1989. ftp://ftp.wtc-ural.ru/pub/ru.crypt/ГОСТ28147

37. Смарт Н. Криптография : пер. с англ / Н. Смарт Н. – М.: Техносфера, 2005. – 528 с.: ил.

38. Стандарт криптографической защиты – AES. Конечные поля / А.С. Зензин, М.А. Иванов ; под ред. М.А. Иванова. – М.: КУДИЦ-ОБРАЗ, 2002. – 176 с.: ил.

39. Стеганографія: Навч. посіб. / О.О. Кузнецов, С.П. Євсеєв, О.Г. Король. – Х. : Вид. ХНЕУ, 2011. – 232 с.: ил.

40. Тилборг ван Х.К.А. Основы криптологии. Профессиональное руководство и интерактивный учебник : пер. с англ / Тилборг ван Х.К.А. – М.: Мир, 2006. – 471 с.: ил.

41. Фергюссон Н. Практическая криптография : пер. с англ. / Н. Фергюссон, Б. Шнайер. – М.: издательский дом “Вильямс”, 2005. – 424 с.: ил.

42. Фомичев В.М. Дискретная математика и криптология / Фомичев В.М. – М.: Диалог-МИФИ, 2003. – 400 с.: ил.

43. Фороузан Б.А. Криптография и безопасность сетей: Учебное пособие / Б.А. Фороузан; пер. с англ. Под ред. А.Н. Берлина. – М.: Интернет-Университет Информационных Технологий: БИНОМ. Лаборатория знаний, 2010. – 784 с.: ил.

44. Черемушкин А.В. Криптографические протоколы. Основные свойства и уязвимости : учеб. пособие для студ. учреждений высш. проф. образования / А.В. Черемушкин. – М.: Издательский центр "Академия", 2009. – 272 с.: ил.

45. Шеннон К. Теория связи в секретных системах // В кн. Работы по теории информации и кибернетики / К. Шеннон. – М.: ИЛ, 1963. – 830 с.: ил.

46. Щербаков А.Ю. Прикладная криптография. Использование и синтез криптографических интерфейсов / А.Ю. Щербаков, А.В. Домашев. – М.: Издательско-торговый дом “Русская редакция”, 2003. – 416 с.: ил.

47. Шнайер Б. Прикладная криптография. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайер Б. – М.: Триумф, 2002. – 797 с.: ил.

ОТВЕТЫ НА КОНТРОЛЬНЫЕ ЗАДАНИЯ

Раздел 2

18. “НСПРГХГ”.
19. “растение”.
20. K = 7.
21. “ЖЪЩУЩХЪКЖ”.
22. “криптография”.
23. “ЗЧЦЖГАИБ”.
24. “космонавт”.
25. “РСЮФТЯГО”.
26. “осторожность”.
27. “ЭЬЙУЛДШЗЦЮЗФ”.
28. “вероятность_”.
29. “ФЦЧЫМЧСЪ”.
30. “зашифрование”.
31. “OLYMVQGAJDGAR”.
32. “WANT TO TRAVEL”.

Раздел 3

10. а) *32* бита; б) *188* блоков.
11. а) *24*; б) *5* бит.
12. а) *20 922 789 888 000*; б) *43* бита.
13. $(11011100)_2$.
14. $(11111010)_2$.
15. а) $(00000000)_2$; б) $(11111111)_2$; в) $(01001101)_2$; г) $(10110010)_2$.
16. $(00111)_2$.
17. $(011)_2$.
18. $(11110)_2$.
19. Прямим *P*-блоком.
20. P-блоком расширения.
21. P-блоком сжатия.
22. Прямим *P*-блоком.

23.

Вход: левый бит \ Вход: правый бит	0	1
0	10	00
1	11	01

24. а) *10*; б) *00*.

25. а) *110*; б) *011*.

Раздел 4

10. а) *7*; *8*; *13*; *4*; *10*; *6*; *3*; *5*; *15*; *14*; б) *3*; *21*; *14*; *18*; *19*; *2*; *15*; *1*; *9*; *11*.

11. а) $k_3 = 0{,}1$; $k_4 = 0{,}2$; $k_5 = 0{,}3$; $k_6 = 0{,}8$; $k_7 = 0{,}4$; $k_8 = 0{,}9$; $k_9 = 0{,}9$; $k_{10} = 0{,}5$; $k_{11} = 0{,}4$; $k_{12} = 0{,}5$; б) $k_4 = 0{,}4$; $k_5 = 0{,}4$; $k_6 = 0{,}1$; $k_7 = 0{,}5$; $k_8 = 0{,}3$; $k_9 = 0{,}9$; $k_{10} = 0{,}8$; $k_{11} = 0{,}6$; $k_{12} = 0{,}5$; $k_{12} = 0{,}3$.

12. $a = 5$; $b = 3$; $c = 11$.

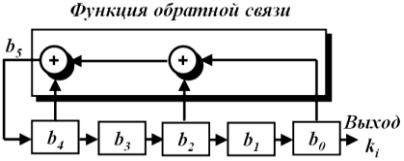
13. а) $x_{11} = 35$; б) $x_{11} = 679$.

14. а) $(110001011001)_2$; б) $(110111100011)_2$.

15. а) $x_1 = 81$; $x_2 = 6$; $x_3 = 36$; $x_4 = 422$; $x_5 = 225$; $x_6 = 370$; $x_7 = 119$; $x_8 = 177$; б) $x_1 = 81$; $x_2 = 144$; $x_3 = 59$; $x_4 = 629$; $x_5 = 639$; $x_6 = 485$; $x_7 = 648$; $x_8 = 660$.

16. 18, 4, 232, 61, 37, 201, 252, 166, 151, 162, 244, 66, 137, 130, 28, 248, 127, 77, 205, 226.

17. а)



б) *31* бит.

18. а) $x^4 + x^3 + x^2 + 1$; б) *15* біт.

19. $(11110001000101111000)_2$.

20. 5 разрядов.

21. Максимальный период двоичной последовательности линейных регистров сдвига: R_1 – *524287* битов; R_2 – *4194303* бита; R_3 – *8388607* битов.

4.22. а) $f(x, y, z) = f(1, 0, 0) = 0$; б) $f(x, y, z) = f(0, 1, 1) = 1$; синхронизируются линейные регистры сдвига R_2 и R_3; в) $f(x, y, z) = f(0, 0, 0) = 0$; г) $f(x, y, z) = f(1, 1, 1) = 1$; синхронизируются линейные регистры сдвига R_1, R_2 и R_3.

Раздел 5

11. $L_0||R_0 = cc1fc6e0f0aae8a5_{16}$.

12. $R_0^E = fa155575150b_{16}$.

13. $R_1^E = 8a2b57d029a6_{16}$.

14. $R_1^{\oplus} = cf430fca9568_{16}$.

15. $R_{12}^{\oplus} = 5dfd4bd5b555_{16}$.

16. а) 5_{16}; б) e_{16}; в) f_{16} и г) 0_{16}.

17. $R_{13}^{S} = 75140940eb51_{16}$.

18. $R_{2}^{P} = 1680e976_{16}$.

19. $R_{3}^{L} = 48a3638f_{16}$.

20. $L_5 = 48a3638f_{16}$ и $R_5 = 4178632b_{16}$.

21. $L_6 = 4178632b_{16}$ и $R_6 = 48a9b329_{16}$.

22. $C_0 = c3c033a_{16}$ и $D_0 = 33f0cfa_{16}$.

23. а) $C_2 = 0f00ceb_{16}$ и $D_2 = cfc33e8_{16}$; б) $C_3 = 3c033ac_{16}$ и $D_3 = = 3f0cfa3_{16}$; в) $C_7 = 033ac3c_{16}$ и $D_7 = 0cfa33f_{16}$; г) $C_{12} = 7587806_{16}$ и $D_{12} = f467e19_{16}$.

24. $k_2 = 4568581abcce_{16}$.

25. а) $k_4 = da2d032b6ee3_{16}$; б) $k_9 = 84bb4473dccc_{16}$; в) $k_{13} = 99c31397c91f_{16}$; г) $k_{16} = 181c5d75c66d_{16}$.

26. $f07704d0741eb2c2_{16}$.

Раздел 6

14. Первым эффектом будет сбой единого бита блока расшифрованных данных. Вторым эффектом будут разрушены следующие *n/l* блоков расшифрованных данных.

15. В первом блоке расшифрованных данных будет разрушено примерно половина битов.

16. В первом блоке расшифрованных данных будет разрушено примерно половина битов; а во втором – один (*5*) бит.

17. Ошибка в открытых данных повлияет на все дальнейшие зашифрованные данные, но самоустранится в ходе расшифрования.

18. Первым эффектом будет сбой второго бита первого блока расшифрованных данных. Вторым эффектом будет разрушение блоков расшифрованных данных со второго по девятый.

19. Будет сбой второго бита первого блока расшифрованных данных. Дальнейшая последовательность блоков будет расшифрована корректно.

20. Будет только сбой второго бита второго блока расшифрованных данных.

Раздел 7

11. $N_1 – af0e1516_{16}$, $N_2 – 191a2ab8_{16}$.

12. $N_1 – 23627c0f_{16}$, $N_2 – 434665b2_{16}$.

13. $N_1 – 10d3b61a_{16}$, $N_2 – 79682f2f_{16}$.

14. $N_1 – 0d3b61a5_{16}$, $N_2 – 29929f76_{16}$.

15. Длина имитовставки должна быть не менее *32* бит.

16. Вероятность навязывания ложных помех $P_{ЛП} \approx 5{,}96 \cdot 10^{-10}$.

Раздел 8

11. $x^7 + x^6 + x^2 + x$.

12. Частное от деления: $x^7 + x^5 + x^4 + x^2 + 1$. Остаток от деления: $x^5 + x^3 + x$.

13. $\{f2\}\cdot x^7 + \{1b\}\cdot x^5 + \{03\}\cdot x^3 + \{17\}\cdot x + \{1d\}$.

14. $\{06\}\cdot x^6 + \{05\}\cdot x^5 + \{19\}\cdot x^4 + \{15\}\cdot x^3 + \{1d\}\cdot x^2 + \{13\}\cdot x + \{04\}$.

15. $\{19\}\cdot x^3 + \{11\}\cdot x^2 + \{1c\}\cdot x + \{0a\}$.

16. $x^7 + x^6 + x^3 + x^2 + x$.

17. $x^5 + x^2 + 1$.

18. $x^7 + x^6 + x^5 + x + 1$.

19. $49ded28945db96f17f39871a7702533b_{16}$.

20. $2b359f6849506a2f27f9ca443ea5b6b_{16}$.

21. $2b359f6849506a2f27f9ca443ea5b6b_{16}$.

22. $f187de773b9639498953db7f1ad2245_{16}$.

23. $584dcaf11b4b5aacdbe7caa81b6bb0e5_{16}$.

24. $72dbe546102a2bf01ad3ef5836ab483d_{16}$.

25. $aa8f5f0361dde3ef82d24ad26832469a_{16}$.

26. $d014f9a8c9ee2589e13f0cc8b6630ca6_{16}$.

27. $w[6] = 5846f2f9_{16}$; $w[7] = 5c43f4fe_{16}$; $w[8] = 544afef5_{16}$; $w[9] = 5847f0fa_{16}$; $w[10] = 4856e2e9_{16}$; $w[11] = 5c43f4fe_{16}$.

28. $w[8] = a573c29f_{16}$; $w[9] = a176c498_{16}$; $w[10] = a97fce93_{16}$; $w[11] = a572c09c_{16}$; $w[12] = 1651a8cd_{16}$; $w[13] = 0244beda_{16}$; $w[14] = 1a5da4c1_{16}$; $w[15] = 0640bade_{16}$.

Раздел 9

8. $k_2^{(1)} = 96a2_{16}$, $k_2^{(2)} = 978e_{16}$, $k_2^{(3)} = b820_{16}$, $k_2^{(4)} = b940_{16}$, $k_2^{(5)} = 1af6_{16}$, $k_2^{(6)} = 7b7d_{16}$.

9. $k_2^{(1)} = e8d8_{16}$, $k_2^{(2)} = 5099_{16}$, $k_2^{(3)} = 4c25_{16}$, $k_2^{(4)} = 95d7_{16}$, $k_2^{(5)} = eb43_{16}$, $k_2^{(6)} = a108_{16}$, $k_2^{(7)} = 8382_{16}$, $k_2^{(8)} = 47e0_{16}$, $k_2^{(9)} = b324_{16}$.

10. $k_3^{(1)} = fca2_{16}$, $k_3^{(2)} = 6bff_{16}$, $k_3^{(3)} = ff29_{16}$, $k_3^{(4)} = 9427_{16}$, $k_3^{(5)} = 9b4b_{16}$, $k_3^{(6)} = a576_{16}$, $k_3^{(7)} = bad1_{16}$, $k_3^{(8)} = 6842_{16}$, $k_3^{(9)} = efa4_{16}$.

11. $k_5^{(1)} = 92d4_{16}$, $k_5^{(2)} = c7e8_{16}$, $k_5^{(3)} = f424_{16}$, $k_5^{(4)} = 1266_{16}$, $k_5^{(5)} = 3370_{16}$, $k_5^{(6)} = 8035_{16}$, $k_5^{(7)} = 1af6_{16}$, $k_5^{(8)} = 7b3d_{16}$.

12. $k_1^{(1)} = 5773_{16}$, $k_1^{(2)} = 7f25_{16}$, $k_1^{(3)} = ab30_{16}$, $k_1^{(4)} = e2ef_{16}$, $k_1^{(5)} = 529a_{16}$, $k_1^{(6)} = 20e2_{16}$, $k_1^{(7)} = c77b_{16}$, $k_1^{(8)} = 9a70_{16}$, $k_1^{(9)} = 7bcc_{16}$.

13. $X_1^{(1)} = abc3_{16}$, $X_2^{(1)} = 5c2d_{16}$, $X_3^{(1)} = e77a_{16}$, $X_4^{(1)} = 1c2d_{16}$.

14. $y_1^{(1)} = 5c92_{16}$, $y_2^{(1)} = 62b8_{16}$.

15. $C = 4825e238d0fb9c46_{16}$.

16. $f_1^{(1)} = 4cb9_{16}$, $f_2^{(1)} = 4000_{16}$.

Раздел 10

13. а) 5 mod 10 = 5, 16 mod 10 = 6, 27 mod 10 = 7, -4 mod 10 = 6, -13 mod 10 = -3 mod 10 = 7, (3 + 8) mod 10 = 1, (3 – 8) mod 10 = 5, (3·8) mod 10 = 4, (3·8·5) mod 10 = (4·5) mod 10= 0. б) 5 mod 11 = 5, 16 mod 11 = 5, 27 mod 11 = 5, -4 mod 11 = 7, -13 mod 11 = -2 mod 11 = 9, (3 + 8) mod 11 = 0, (3 – 8) mod 11 = 6, (3·8) mod 11 = 2, (3·8·5) mod 11= (2·5) mod 11= 10.

14. 2^8 mod 10 = 6, 2^7 mod 10 = 7, 7^{19} mod 100 = 43, 7^{57} mod 100 = 7.

15. 108 = 2·2·3·3·3, 77 = 7·11, 65 = 5·13, 30 = 3·3·5, 159 = 3·53.

16. Пары (*25,12*) и (40,27) взаимно просты, остальные – нет (числа (*25,15*) делятся на 5, (*13,39*) делятся на *13*).

17. φ(14) = 6, φ(15) = 8.

18. φ(53) = 52, φ(21) = φ(7)·φ(3) = 6·2 = 12, φ(159) = φ(3)·φ(53) = 2·52 = 104.

19. 3^{13} mod 13 = 3·3^{12} mod 13 = 3, 5^{22} mod 11 = 5^2·5^{10}·5^{10} mod 11 = 25 mod 11 = 3, 3^{17} mod 5 = 3.

20. 3^9 mod 20 = 3·3^8 mod 20 = 3, 2^{14} mod 21 = 2^2·2^{12} mod 21 = 4, 2^{107} mod 159 = 2^3·2^{104} mod 159 = 8.

21. gcd(21,12) = 3, gcd(30,12) = 6, gcd(24,40) = gcd(40,24) = 8, gcd(33,16) = 1.

22. а) *x = -1, y = 2.* б) *x = 1, y = -2.* в) *x = 2, y = -1.* г) *x = 1, y = -2.*

23. 3^{-1} mod 7 = 5, 5^{-1} mod 8 = 5, 3^{-1} mod 53 = 18, 10^{-1} mod 53 = 16.

24. Простые числа, которые меньше *100*: *2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 73, 79, 83, 89, 97.* Из них числа *5, 7, 11, 23, 47, 59* и *83* соответствуют виду *p = 2·q + 1*.

25. При *p = 11* в качестве параметра *g* могут быть выбраны числа *2, 6, 7* и *8*.

26. а) $e_A = 20$, $e_B = 17$, $K_{AB} = 21$; б) $e_A = 13$, $e_B = 14$, $K_{AB} = 10$; в) $e_A = 21$, $e_B = 9$, $K_{AB} = 16$; г) $e_A = 8$, $e_B = 5$, $K_{AB} = 9$; д) $e_A = 6$, $e_B = 17$, $K_{AB} = 16$.

27. а) $d_A = 11$, $d_B = 13$, $x_1 = 17$, $x_2 = 5$, $x_3 = 6$, $x_4 = 4$; б) $d_A = 3$, $d_B = 19$, $x_1 = 8$, $x_2 = 12$, $x_3 = 3$, $x_4 = 6$; в) $d_A = 5$, $d_B = 11$, $x_1 = 14$, $x_2 = 10$, $x_3 = 3$, $x_4 = 10$; г) $d_A = 5$, $d_B = 15$, $x_1 = 7$, $x_2 = 21$, $x_3 = 14$, $x_4 = 17$; д) $d_A = 11$, $d_B = 5$, $x_1 = 15$, $x_2 = 2$, $x_3 = 8$, $x_4 = 9$.

28. а) $e_B = 13$, $r = 14$, $s = 12$, $M' = 5$; б) $e_B = 16$, $r = 9$, $s = 15$, $M' = 10$; в) $e_B = 15$, $r = 16$, $s = 14$, $M' = 10$; г) $e_B = 21$, $r = 14$, $s = 12$, $M' = 5$; д) $e_B = 8$, $r = 5$, $s = 12$, $M' = 10$.

29. а) $n_A = 55$, $\varphi(n_A) = 40$, $d_A = 27$, $C = 23$, $M' = 12$; б) $n_A = 65$, $\varphi(n_A) = 48$, $d_A = 29$, $C = 50$, $M' = 20$; в) $n_A = 77$, $\varphi(n_A) = 60$, $d_A = 43$, $C = 52$, $M' = 17$; г) $n_A = 91$, $\varphi(n_A) = 72$, $d_A = 29$, $C = 88$, $M' = 30$; д) $n_A = 33$, $\varphi(n_A) = 20$, $d_A = 7$, $C = 9$, $M' = 18$.

30. M = 111.

31. Открытый ключ $n_A = 3763$, зашифрованное сообщение *C = 1194 1937 1734 2018 0400 1932 0831.*

32. $p_A = 59$, $q_A = 71$; “*защита*”.

33. а) *x = 17*; б) *x = 10*; в) *x = 28*; г) *x = 14*; д) *x = 30.*

34. а) *x = 10000*; б) *x = 20000*; в) *x = 1000*; г) *x = 12345*; д) *x = 25000.*

Учебное издание

Ахметов Бахытжан Сражатдинович
Корченко Александр Григорьевич
Сиденко Владимир Павлович
Дрейс Юрий Александрович
Сейлова Нургуль Абадуллаевна

ПРИКЛАДНАЯ КРИПТОЛОГИЯ:

методы шифрования

Учебное пособие