УДК 681.3(075)
ББК 32.973-01я7
М29

Рецензенты: Шишкин В.В. – декан ФИСТ УлГТУ, канд. техн. наук., проф. каф. ИВК УлГТУ
Негода Д.В. – канд. техн. наук., старший инженер-программист ООО «Креативная разработка»

Утверждено редакционно-издательским советом университета в качестве учебного пособия

Мартынов А.И.
М29 Методы и задачи криптографической защиты информации : учебное пособие для студентов специальности «Вычислительные машины, комплексы, системы и сети»/ А. И. Мартынов. – Ульяновск : УлГТУ, 2007. – 92 с.

ISBN 978–5–9795–0

Пособие написано в соответствии с рабочей программой дисциплины «Методы и средства защиты компьютерной информации» для студентов третьего курса специальности «Вычислительные машины, комплексы, системы и сети». Представлены основные теоретические понятия и определения, а также подробно рассмотрены основные методы и алгоритмы криптографических преобразований, используемых в современных системах защиты.

Подготовлено на кафедре «Вычислительная техника».

УДК 681.3(075)
ББК 32.973-01я7

ISBN 978–5–9795–0

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Проблема защиты информации – это вечная проблема человечества. На разных этапах своего развития она решалась по-разному, с присущей для данной эпохи характерностью.

Появление и бурное развитие информационных технологий в конце XX века возвело проблему защиты информации в ранг первоочередных задач, от успешного решения которых часто зависит не только процветание предприятия, но и безопасность нации.

Однако очевидна сложность проблемы информационной безопасности, проистекающая как из сложности и разнородности современных информационных систем, так и из необходимости применения комплексного подхода к безопасности с привлечением законодательных, административных и программно-технических мер. Находясь на стыке нескольких разнородных дисциплин, таких как: «Математика», «Криптография», «Аппаратное и программное обеспечение ЭВМ», «Программирование на языках высокого и низкого уровней», «Сетевые технологии», «Юриспруденция», «Психология» сама дисциплина «Методы и средства защиты компьютерной информации» является синтезированной и требует от инженера по информационной безопасности глубоких теоретических знаний и практических навыков в каждой из вышеперечисленных областей.

На сегодняшний день под *защитой компьютерной информации* понимается совокупность мероприятий, методов и средств, обеспечивающих решение задач проверки целостности информации, исключения несанкционированного доступа к ресурсам ЭВМ и хранящимся в ней программам и данным, а также исключения несанкционированного использования программных продуктов.

Традиционно выделяют следующие направления защиты компьютерной информации:

- *Криптография* – наука о защите информации от прочтения ее посторонними лицами. Защита достигается путем некоторого преобразования исходных данных, которое делает их трудно раскрываемыми без знания специальной информации (криптографического ключа).
- *Сетевая безопасность* – рассматривает способы защиты (как аппаратные, так и программные) от несанкционированного доступа к удаленной ЭВМ посредством сетевых атак.
- *Защита от несанкционированного копирования* – предотвращает использование нелицензионных копий программного обеспечения (ПО) и защищает права разработчиков.
- *Антивирусология* – наука о способах борьбы с компьютерными вирусами и прочими самораспространяющимися программами,

направленная на обеспечение и поддержание целостности хранимых данных.

- *Системная защита* – комплекс аппаратных и программных средств, направленных на обеспечение целостности и недоступности данных в случаях отказа техники, ошибочных действий и прочих причин стихийного характера.

Современные системы защиты информации строятся по многоуровневой схеме, которая позволяет комплексно использовать различные средства и методы защиты, и за счет этого повысить общую эффективность системы при снижении расходов на её организацию и обслуживание. Каждый уровень системы защиты может в свою очередь делиться на ряд рубежей (подуровней), тесно взаимодействующих между собой. При этом отдельный подуровень строится на основе одного или совокупности различных приёмов защиты, направленных на защиту от конкретной угрозы.

В первой главе данного учебного пособия рассматриваются теоретические основы предмета «Методы и средства защиты компьютерной информации», даются базовые определения и понятия, рассматриваются международные стандарты и требования к современным системам защиты.

Во второй главе учебного пособия идет речь о криптографических методах, алгоритмах и средствах, составляющих базу современных систем защиты компьютерной информации. Особый упор делается на алгоритмической составляющей приведенных решений, поскольку пособие рассчитано, прежде всего, на студентов специальности «Вычислительные машины, системы, комплексы и сети».

Все теоретические материалы подкреплены примерами выполнения лабораторных работ.

ГЛАВА 1. ОБЩИЕ ПРИНЦИПЫ ПРОЕКТИРОВАНИЯ СИСТЕМ ЗАЩИТЫ

С массовым внедрением компьютерной техники во все сферы человеческой деятельности резко возрос объем и концентрация информации, хранимой в электронном виде. Это позволило злоумышленникам значительно сократить временные и материальные затраты на то, чтобы получить доступ к этой информации. Кроме того, усложнение вычислительного процесса и недостаточная квалификация персонала часто становятся причинами непреднамеренного разрушения и уничтожения информационных ресурсов.

К сожалению, во многих источниках понятие *защиты компьютерной информации* раскрывается не полностью, а дается лишь частичное его определение, как защита от умышленных попыток человека получить доступ к этой информации либо модифицировать ее.

Как показывают исследования, проведенные американским центром DataPro Research в 1998 году, основные причины повреждений электронной информации распределились следующим образом (Рисунок 1.1):

- 52% – ошибочные действия пользователя;
- 25% – стихийные бедствия (затопления, пожары и т. п.);
- 10% – умышленные действия человека;
- 10% – отказ техники;
- 3% – прочие непредвиденные обстоятельства.

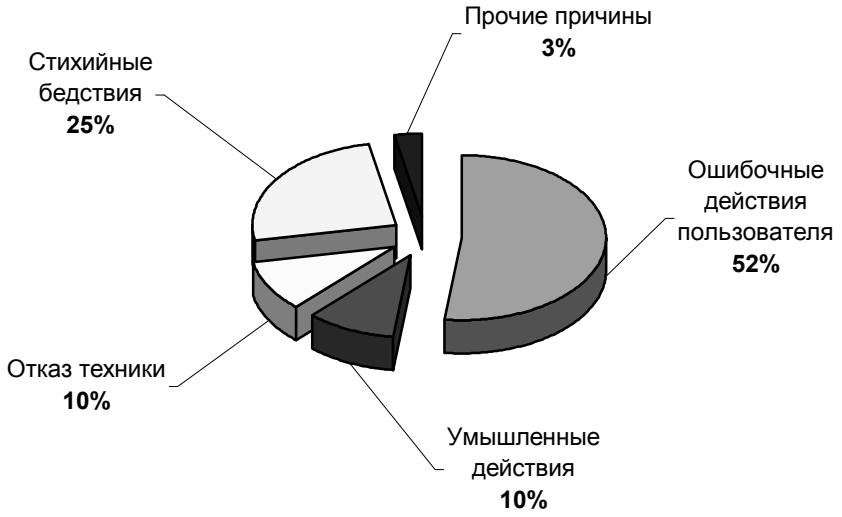


Рисунок 1.1. Причины повреждения электронной информации (DataPro Research, 1998 г.)

Как видно из этой схемы, в каждом втором случае причиной повреждения информации являются ошибочные действия пользователей, а умышленное действие человека, составляет лишь 10%. Но и эти 10% – уже вполне настораживающий факт. Нельзя забывать о том, что конфиденциальная информация, полученная злоумышленником, и используемая им в своих корыстных целях, может принести гораздо больший ущерб организации, нежели непреднамеренное уничтожение этой информации. Попытка

постороннего лица получить доступ к информации обычно называется *атакой на информацию*. Более точное определение дается в работе [1]:

При хранении, поддержании и предоставлении доступа к любому информационному ресурсу его владелец, либо уполномоченное им лицо, накладывает явно либо самоочевидно набор правил по работе с ней. Умышленное их нарушение классифицируется как атака на информацию.

Тогда, исходя из этого определения, будем называть *злоумышленником* лицо, которое совершает атаку на информацию.

По исследованиям центра DataPro Research, действия злоумышленников, получивших доступ к закрытой информации, распределились так (Рисунок 1.2):

- 44% – кражи денег с электронных счетов;
- 16% – вывод из строя программного обеспечения;
- 16% – кража информации с различными последствиями;
- 12% – фальсификация информации;
- 10% – заказ различного рода услуг.

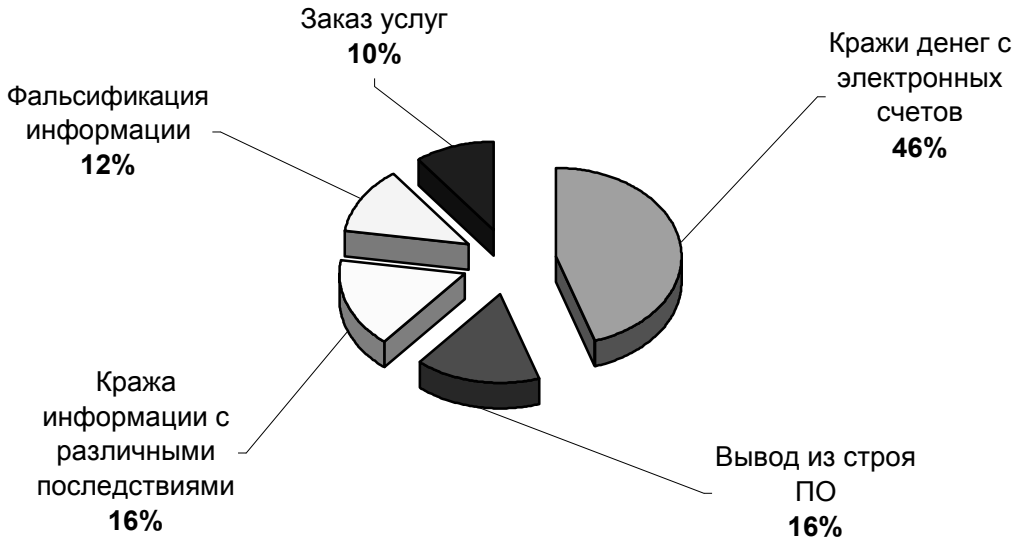


Рисунок 1.2. Действия злоумышленников (DataPro Research, 1998 г.)

Это означает, что современные информационные системы хранения, накопления и обработки информации должны быть защищены не только от атак на информацию, но и от возможных причин повреждения и оснащены комплексной системой защиты, которая бы обеспечивала:

1. Проверку целостности информации и возможность ее восстановления после повреждения.
2. Исключение несанкционированного доступа к ресурсам ЭВМ и/или хранящимся в ней программам и данным.
3. Исключение несанкционированного использования программ (защита от копирования).

1. Возможные каналы утечки информации

Под возможным *каналом утечки информации* будем понимать способ, позволяющий нарушителю получить доступ к хранящейся или обрабатываемой информации. По каждому возможному каналу утечка информации происходит с помощью одного из трех типов средств:

- *Человек*: хищение носителей информации, чтение информации с экрана посторонними лицами, чтение информации из оставленных без присмотра распечаток и т.п.
- *Аппаратура*: подключение к ЭВМ специально разработанных аппаратных средств, обеспечивающих доступ к информации.
- *Программы*: несанкционированный доступ к информации, расшифровка программой зашифрованных данных, незаконное копирование информации, вредоносное действие (вирусы, черви и т. п.)

Прежде чем переходить к вопросам, касающимся непосредственно защиты информации, рассмотрим наиболее распространенные методы взлома, проанализировав которые можно получить общую картину современной ситуации в области информационной безопасности.

2. Обзор наиболее распространенных методов взлома

К наиболее распространенным методам взлома можно отнести следующие [1]:

- доступ к информации через терминалы защищенной информационной системы;
- получение пароля на основе ошибок администратора и пользователей;
- получение пароля на основе ошибок в реализации системы;
- социальная психология и иные способы получения ключа;
- комплексный поиск возможных методов доступа.

Рассмотрим более подробно каждый из этих методов:

Доступ к информации через терминалы защищенной системы

Под *терминалом* понимается точка входа пользователей в систему (в простейшем случае это log-in запрос). Доступ к терминалам может быть *физическим*, когда терминал – это ЭВМ с клавиатурой и дисплеем, либо *удаленным* – чаще всего по телефонной линии связи (здесь терминалом является модем).

При использовании терминалов с физическим доступом необходимо соблюдать следующие требования:

1. Наличие имени пользователя и личного пароля для каждого пользователя, имеющего доступ к терминалу. В том случае, если доступ к терминалу имеет только один человек или группа лиц с

одинаковыми привилегиями, то допускается отсутствие регистрационного имени пользователя.
2. Наличие административных мер, обеспечивающих контроль за доступом в помещение, в котором установлен терминал.
3. Если терминал установлен в местах скопления массы людей (банкомат, удаленный терминал), то на нем должно быть предусмотрено наличие устройств, позволяющих видеть информацию только работающему в данный момент клиенту (пластмассовые защитные ограждения, шторки и т. п.)

При использовании удаленного терминала необходимо учитывать современные возможности аппаратуры установленной на автоматизированной телефонной станции (АТС). Все дело в том, что при наличии специального программного обеспечения и тонового набора для одного звонка достаточно около 4 секунд. За это время можно определить существует ли на этом телефонном номере модем, и узнать его шестизначный номер. Это означает, что за 1 минуту можно перебрать 15 номеров телефонной станции, за час – 1000 номеров, а за рабочий день – всю АТС (около 10 000 номеров). Если учесть, что в Ульяновске около 15 АТС, то за две недели можно проверить все телефоны и узнать номера подключенных к ним модемов. Поэтому основными требованиями по безопасности при доступе к удаленным терминалам являются следующие:

1. Любой удаленный терминал должен запрашивать имя пользователя и пароль. Того, что якобы никто не знает шестизначного номера Вашего модема, отнюдь не достаточно для конфиденциальности.
2. Своевременное отключение всех модемов, не требующихся в данный момент в фирме.
3. По возможности рекомендуется использовать схему возвратного звонка от модема, поскольку она гарантирует с уровнем надежности АТС то, что удаленный клиент получил доступ с определенного телефонного номера.
4. Из логина и пароля рекомендуется исключить любую информацию, касающуюся непосредственно фирмы (её названия, логотипа и т.п.).
5. При входе в систему на экран рекомендуется выводить предупреждение о том, что вход в систему без полномочий преследуется по закону. Во-первых, это может отпугнуть начинающих злоумышленников, а во-вторых, является надежным аргументом в пользу атакованной фирмы в судебном разбирательстве, если таковое будет производиться.

Независимо от типа терминала определенные требования должны соблюдаться при работе с коммуникационным оборудованием. Зона ядра информационной системы должна быть защищена от прослушивания, либо

весь информационный канал должен быть защищен по конфиденциальной схеме идентификации и надежной схеме аутентификации клиента. Этим занимаются криптосистемы, речь о которых пойдет позже.

Получение пароля на основе ошибок администратора и пользователей

Одним из самых эффективных методов получения пароля является перебор паролей по словарю. Технология перебора паролей родилась достаточно давно, но до сих пор используется и очень успешно. Например, программа ShadowScan позволяет методом перебора паролей взломать практически любой FTP или HTTP сервер или вскрыть зашифрованный файл за приемлемый промежуток времени. Встроенный генератор паролей избавляет взломщика от необходимости создавать словарь вручную. Для того чтобы добиться успеха в 60% случаев обычно достаточно словаря размером в 50 000 существительных. Огромное число инцидентов со взломами систем заставило пользователей добавлять к словам 1–2 цифры с конца, записывать первую и/или последнюю букву в верхнем регистре, использовать «транслит». Но как показали исследования, даже составление двух совершенно не связанных осмысленных слов подряд не дает сколь нибудь реальной надежности паролю. К этому времени широкое распространение получили языки составления паролей, записывающие в абстрактной форме основные принципы составления паролей среднестатистическими пользователями ЭВМ.

Еще одной модификацией подбора паролей является проверка паролей, устанавливаемых в системе по умолчанию. В некоторых случаях после инсталляции администратор программного обеспечения не удосуживается проверить, из чего состоит система безопасности, и какие там используются пароли. Следовательно, тот пароль, который был установлен в системе по умолчанию, так и остается основным действующим паролем. В сети Интернет можно найти огромные списки паролей по умолчанию практически ко всем версиям программного обеспечения, если они устанавливаются на нем производителем.

Основные требования к информационной безопасности, основанные на анализе данного метода, следующие:

1. Вход всех пользователей в систему должен подтверждаться вводом уникального для клиента пароля.
2. Пароль должен тщательно подбираться так, чтобы его информационная емкость соответствовала времени полного перебора пароля. Для этого необходимо детально инструктировать клиентов о понятии «простой к подбору пароль», либо передать операцию выбора пароля в ведение инженера по информационной безопасности.
3. Пароли, используемые по умолчанию, должны быть сменены до официального запуска системы.

4. Все ошибочные попытки войти в систему должны учитываться, записываться в файл журнала событий и анализироваться через «разумный» промежуток времени. Если в системе предусмотрена возможность блокирования клиента либо всей системы после определенного количества неудачных попыток войти в систему, этой возможностью необходимо воспользоваться. Разумно блокировать клиента после 3-й подряд неправильной попытки набора пароля и блокировать систему после $K = \max(\text{int}(N \cdot 0.1 \cdot 3) + 1,\ 3)$ неудачных попыток входа за некоторый период времени.

   В данной формуле
   - N – среднее количество подключившихся за этот период времени к системе клиентов;
   - *0.1* – 10%-ый предел «забывчивости пароля»;
   - *3* – три попытки на вспоминание пароля.
5. Все действительные в системе пароли желательно проверять современными программами подбора паролей, либо оценивать лично инженеру по безопасности.
6. Через определенные промежутки времени необходима принудительная смена паролей у клиентов. В зависимости от уровня конфиденциальности интервалами смены полей могут быть год, месяц, неделя, день или даже час.
7. Все неиспользуемые в течение долгого времени имена регистрации должны переводиться в закрытое состояние. Это относится к сотрудникам, находящимся в отпуске, на больничном, в командировке. Это также относится и именам регистрации, которые были созданы для тестов и испытаний системы.
8. От сотрудников и всех операторов терминала необходимо требовать строгое неразглашение паролей, отсутствие каких-либо взаимосвязей с широкоизвестными фактами и данными, и отсутствия бумажных записей пароля.

Получение пароля на основе ошибок в реализации

Следующей по частоте использования является методика получения паролей из самой системы. Основными двумя возможностями выяснения пароля являются несанкционированный доступ к носителю информации, на котором они содержатся, либо использование недокументированных возможностей и ошибок в реализации системы.

Первая группа методов основана на том, что в любой системе приходится где-либо хранить подлинники паролей всех клиентов для того, чтобы сверять их в момент регистрации. При этом пароли могут храниться в открытом виде, как это имеет место во многих клонах UNIX, так и представленные в виде малозначащих контрольных сумм, как это реализовано в ОС Windows, Novell

NetWare и многих других. Дело в том, что при хранении паролей на носителе не может быть использована основная методика защиты данных – шифрование, так как при этом необходимо также хранить и ключи к зашифрованным паролям, для того чтобы система автоматически могла производить идентификацию клиента. Получив доступ к такому носителю, злоумышленник может либо восстановить пароль в читабельном виде, что бывает очень редко, либо отправлять запросы, подтвержденные контрольной суммой, не раскодируя сам пароль.

Все рекомендации по предотвращению хищения паролей состоят в проверке: не доступен ли файл с паролями или таблица в базе данных, в которой хранятся эти пароли, кому-то ещё, кроме администраторов системы.

Второй способ получения паролей на основе ошибок в реализации сегодня встречается достаточно редко. Ранее эта методика использовалась разработчикам намного чаще в основном в целях отладки либо для экстренного восстановления работоспособности системы. Но постепенно с развитием как технологий обратной компиляции, так и информационной связанности мира, она постепенно стала исчезать. Любые недокументированные возможности рано или поздно становятся известными, после чего эта новость с головокружительной быстротой облетает мир, и разработчикам приходится рассылать всем пользователям скомпрометированной системы «программные заплатки» либо новые версии программного продукта. Единственной мерой профилактики данного метода является постоянный поиск на серверах, посвященных компьютерной безопасности, объявлений обо всех неприятностях с ПО, установленном в Вашем учреждении.

Еще одной распространенной технологией получения паролей является копирование буфера клавиатуры в момент набора пароля на терминале. Этот метод используется редко, так как для него необходим доступ к терминальной машине с возможностью запуска программ. Но если все-таки злоумышленник получает такой доступ, то эффективность этого метода достаточно высока.

Двумя основными методами борьбы с копированием паролей являются:

1. Адекватная защита рабочих станций от запуска сторонних программ:
   - отключение сменных носителей информации (гибких дисков);
   - специальные драйверы, блокирующие запуск исполнимых файлов без ведома оператора;
   - мониторы, уведомляющие о любых изменениях системных настроек и списка автоматически запускаемых программ.
2. Очень мощная, но неудобная мера – система единовременных паролей (при каждой регистрации в системе клиентам с очень высоким уровнем ответственности самой системой генерируется новый пароль).

Следующий метод получения паролей относится к сетевому программному обеспечению. Проблема заключается в том, что во многих программах не учитывается возможность перехвата любой информации,

идущей по сети, – так называемого сетевого трафика. Первоначально с внедрением компьютерных сетей так оно и было. Сеть располагалась в пределах 2–3 кабинетов, либо здания с ограниченным физическим доступом к кабелям. Однако стремительное развитие глобальных сетей затребовало на общий рынок те же версии ПО без какого-либо промедления для усиления безопасности. Более половины протоколов сети Интернет передают пароли в нешифрованном виде – открытым текстом. К ним относятся протоколы передачи электронной почты SMTP и POP3, протокол передачи файлов FTP.

Современное программное и аппаратное обеспечение позволяют получать всю информацию, проходящую по сегменту сети, к которому подключен конкретный компьютер, и анализировать ее в реальном масштабе времени. Это может сделать служащий компании со своего рабочего компьютера или злоумышленник, подключившийся к сегменту с помощью портативной ЭВМ. Наконец трафик, идущий от Вас к Вашему партнеру или в другой офис сети Интернет, технически может прослушиваться со стороны Вашего непосредственного провайдера или со стороны любой организации, предоставляющей транспортные услуги для сети Интернет.

Для комплексной защиты от подобной возможности необходимо выполнять следующие меры:

1. Физический доступ к сетевым кабелям должен соответствовать уровню доступа к информации.
2. При определении топологии сети следует избегать широковещательных топологий.
3. Оптимальной единицей сегментирования является группа операторов с равными правами доступа, либо если в группу входит не более 10 человек, то комната или отдел внутри группы.
4. Ни в коем случае на одном кабеле не должны находится операторы с разными уровнями доступа, если только весь передаваемый трафик не шифруется, а идентификация не производится по скрытой схеме без открытой передачи пароля.
5. Ко всем информационным потокам, выходящим за пределы фирмы, должны применяться те же правила для объединения разноуровневых терминалов.

Социальная психология и иные способы получения паролей

Воздействовать на ум и поведение человека можно различными путями, одни требуют лишь специфичной подготовленности специалиста (убеждение, внушение, обман, подкуп, шантаж и т. п.), а другие – еще и специальной аппаратуры (технотронные приемы, зомбирование).

Выбор применяемой методики зависит от многих факторов, таких как:

- реальная уязвимость объекта (черты его характера, эпизоды биографии, наличная ситуация);

- цель намеченного воздействия (изменение мышления, привлечение к сотрудничеству, получение информации, одноразовое содействие);
- собственные возможности (обладание временем, умением, знанием, техаппаратурой, компетентными помощниками);
- персональные установки исполнителя (уровень его моральной устойчивости).

На практике чаще всего используются следующие методы социальной психологии:

1. *Звонок администратору от лица клиента*. Злоумышленник выбирает из списка сотрудников тех, кто не использовал пароль в течение нескольких дней и кого администратор не знает по голосу. Затем следует звонок администратору с объяснением ситуации о забытом пароле, искренние извинения, просьба зачитать пароль или сменить его на новый. Больше чем в половине случаев просьба будет удовлетворена, а факт подмены будет зафиксирован только после первой неудачной попытки регистрации истинного сотрудника.
2. *Звонок клиенту от лица администратора.* Почти такая же схема, но только в обратную сторону может быть разыграна злоумышленником в адрес сотрудника фирмы. В этом случае он уже представляется сотрудником службы информационной безопасности и просит назвать пароль из-за происшедшего сбоя в базе данных или другой причины. Фантазия в этом случае может придумывать самые правдоподобнее причины, по которым сотруднику «просто необходимо» вслух назвать пароль.

Оба метода относятся к группе «атака по социальной психологии» и могут принимать самые различные модификации. Их профилактикой может быть только тщательное разъяснение всем сотрудникам правил работы с защищенной информацией.

Большое внимание следует уделять любым носителям информации, покидающим пределы фирмы. Наиболее частыми причинами этого бывают ремонт аппаратуры и списание технологически устаревшей техники. На сегодняшний день не существует разумных по критерию «цена/надежность» носителей информации, не доступных к взлому. Практически невскрываемым может быть только энергонезависимый носитель, автоматически разрушающий информацию при попытке несанкционированного подключения к любым точкам, кроме разрешенных разъемов. Однако все это из области сумасшедших цен и военных технологий.

Для бизнес-класса и частной переписки данная проблема решается гораздо проще и дешевле при помощи криптографии. Против самых новейших технологий и миллионных расходов здесь стоит математика, и этот барьер до сих пор невозможно преодолеть.

Комплексный поиск возможных методов доступа

Злоумышленники исключительно тщательно изучают системы безопасности перед проникновением в нее. Очень часто они находят очевидные и простые методы «взлома» системы, которые создатели просто «проглядели», создавая, возможно, очень хорошую систему идентификации или шифрования.

3. Критерии оценки безопасности компьютерных систем

Анализ возможных угроз и анализ рисков, несомненно, помогает выбору мер безопасности, которые должны быть осуществлены, чтобы уменьшить риск до приемлемого уровня. Эти меры можно обеспечить через соответствующие комбинации информационных технологий (ИТ), реализующих функции системы, и/или через внешние меры. Однако для того чтобы оценить безопасность создаваемой системы, необходимо иметь какой-то эталон, позволяющий проводить сравнения по некоторым общим критериям, зафиксированным в этом эталоне.

Общие критерии позволяют сравнивать результаты независимых оценок безопасности ИТ. Чтобы достигнуть большей сравнимости между результатами оценок, оценки должны быть выполнены в пределах структуры авторитетной схемы оценки, которая устанавливает стандарты и контролирует качество оценок. Такие схемы оценки в настоящее время существуют в нескольких странах и основаны на различных (хотя и сопоставимых) критериях оценки. Они разработаны с учетом совместимости с этими существующими критериями, чтобы таким образом сохранить преемственность оценок безопасности.

Критерии оценки надежных компьютерных систем были впервые опубликованы в 1983 году Министерством обороны США в издании называемым, чаще всего по цвету обложки «Оранжевой книгой» [2]. Оранжевая книга поясняет понятие безопасной системы, которая «управляет, посредством соответствующих средств, доступом к информации, так что только должным образом авторизированные лица или процессы, действующие от их имени, получают право читать, писать, создавать и удалять информацию». Очевидно, что абсолютно безопасных систем не существует, что это абстракция. Любую систему можно «взломать», если располагать достаточно большими материальными и временными ресурсами. Есть смысл оценивать лишь степень доверия, которое разумно оказать той или иной системе.

В «Оранжевой книге» дается следующее определение надежной системы:

Надежной называется система, использующая достаточные программные и аппаратные средства, чтобы обеспечить одновременную обработку информации разной степени секретности группой пользователей без нарушения прав доступа.

Степень доверия, или надежность системы, оценивается по двум основным критериям:

Политика безопасности – набор законов, правил и норм поведения, определяющих, как организация обрабатывает, защищает и распространяет информацию. Чем надежнее система, тем строже и многообразнее должна быть политика безопасности. В зависимости от сформулированной политики можно выбирать конкретные механизмы, обеспечивающие безопасность системы. Политика безопасности – это активный компонент защиты, включающий в себя анализ возможных угроз и выбор мер противодействия.

Гарантированность – мера доверия, которая может быть оказана архитектуре и реализации системы. Гарантированность может проистекать как из тестирования, так и из проверки общего замысла и исполнения системы в целом и ее компонентов. Гарантированность показывает, насколько корректны механизмы, отвечающие за проведение в жизнь политики безопасности. Гарантированность можно считать пассивным компонентом защиты, надзирающим над самими защитниками.

Основные элементы политики безопасности

Согласно Оранжевой книге, политика безопасности должна включать в себя следующие элементы:

- *Произвольное управление доступом* – это метод ограничения доступа к объектам, основанный на учете личности субъекта или группы, в которую субъект входит. Произвольность управления состоит в том, что некоторое лицо (обычно владелец объекта) может по своему усмотрению давать другим субъектам или отбирать у них права доступа к объектам. Текущее состояние прав доступа описывается матрицей, в строках которой перечислены субъекты, а в столбцах – объекты. В клетках, расположенных на пересечении, записываются способы доступа, допустимые для субъекта относительно объекта, например чтение, запись, выполнение, возможность передачи прав другим субъектам и т. п.
- *Безопасность повторного использования объектов* – важное на практике дополнение средств управления доступом, предохраняющее от случайного или преднамеренного извлечения секретной информации из «мусора». Безопасность повторного использования должна гарантироваться для областей оперативной памяти, для дисковых блоков и магнитных носителей в целом. Важно обратить внимание на следующий момент. Поскольку информация о субъектах также представляет собой объект, необходимо позаботиться о безопасности «повторного использования субъектов». Когда пользователь покидает организацию, следует не только лишить его возможности входа в систему, но и запретить доступ ко всем объектам. В противном случае новый сотрудник может получить ранее

использовавшийся идентификатор, а с ним и все права своего предшественника.

- *Метки безопасности* – для реализации принудительного управления доступом с субъектами и объектами ассоциируются метки безопасности. Метка субъекта описывает его благонадежность, метка объекта – степень закрытости содержащейся в нем информации. Согласно Оранжевой книге, метки безопасности состоят из двух частей – уровня секретности и списка категорий. Уровни секретности образуют упорядоченное множество (совершенно секретно, секретно, конфиденциально, несекретно). Категории образуют неупорядоченный набор. Их назначение описать предметную область, к которой относятся данные. Механизм категорий позволяет разделить информацию по отсекам, что способствует лучшей защищенности. То есть определенный субъект не может получить доступ к «чужим» категориям, даже если он обладает уровнем «Совершенно секретно».
- *Принудительное управление доступом* – основано на сопоставлении меток безопасности субъекта и объекта. Субъект может читать информацию из объекта, если уровень секретности субъекта не ниже, чем у объекта, а все категории, перечисленные в метке безопасности объекта, присутствуют в метке субъекта. В таком случае говорят, что метка субъекта *доминирует* над меткой объекта. Субъект может записывать информацию в объект, если метка безопасности объекта доминирует над меткой субъекта. В частности, субъект с уровнем благонадежности «секретно» может писать в совершенно секретные файлы, но не может в несекретные (при этом также учитывается набор категорий). На первый взгляд подобное ограничение может оказаться странным, однако оно вполне разумно. Ни при каких операциях уровень секретности информации не должен понижаться, хотя обратный процесс вполне возможен. Посторонний человек может случайно узнать секретные сведения и сообщить их куда следует, однако лицо, допущенное к работе с секретными документами, не имеет право раскрывать их содержание. Описанный способ управления называется принудительным, поскольку он не зависит от воли субъектов (даже системных администраторов). После того как зафиксированы метки безопасности субъектов и объектов, оказываются зафиксированными и права доступа.

Пример: пусть у нас имеются три субъекта: «Субъект А», «Субъект В» и «Субъект С», каждый из которых имеет уровни благонадежности: «Секретно», «Конфиденциально» и «Несекретно» соответственно (Рисунок 1.3).

Определим взаимное доминирование субъектов и объектов по их меткам безопасности.

Сначала определим, кто из субъектов может читать информацию, и из каких объектов.

«Субъект А» доминирует над объектами «Объект В», «Объект D» и «Объект Е», так как уровни секретности этих объектов не ниже, чем у субъекта «Субъект А», а все категории, перечисленные в метках безопасности объектов «Объект В», «Объект D» и «Объект Е», присутствуют в метке субъекта «Субъект А».

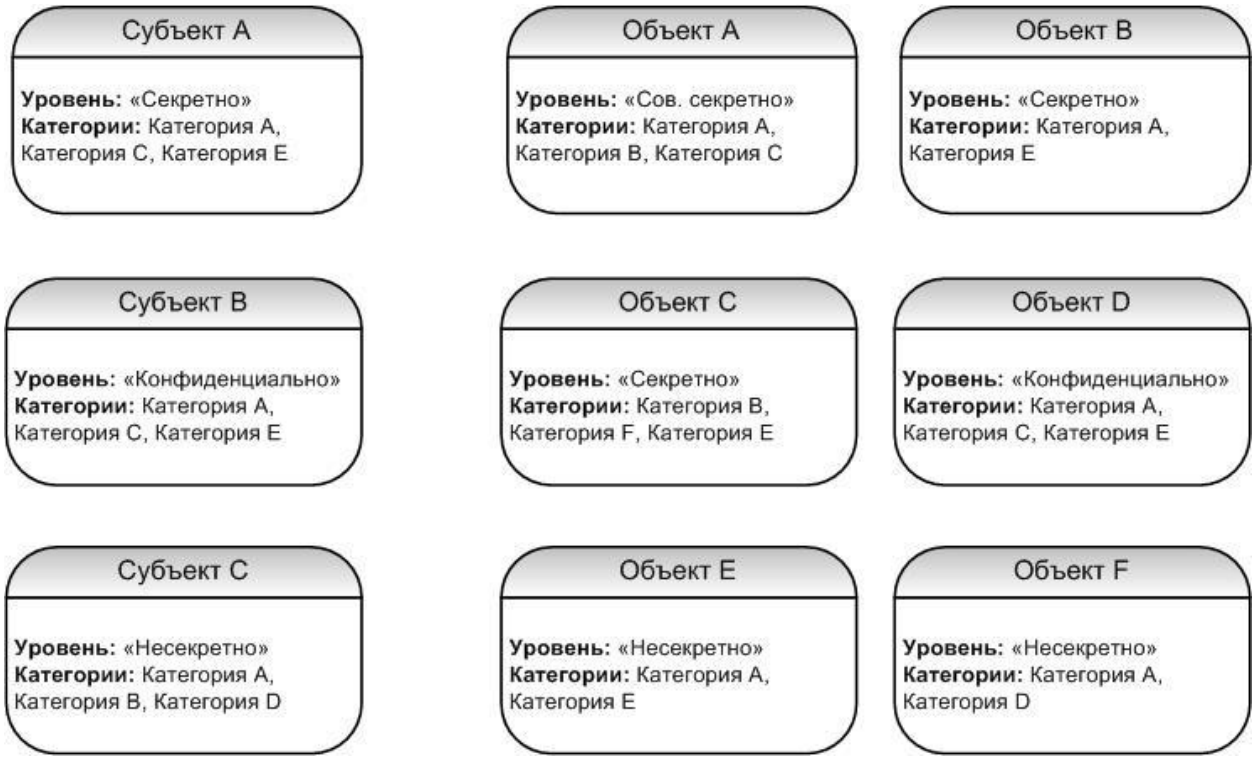


Рисунок 1.3. Пример описания меток безопасности для субъектов и объектов

Аналогично «Субъект В» доминирует над объектами «Объект В» и «Объект Е», а «Субъект С» доминирует над объектом «Объект F».

Это означает, что «Субъект А» может производить чтение информации из объектов «Объект В», «Объект D» и «Объект Е», «Субъект В» из объектов «Объект В» и «Объект Е», а «Субъект С» из объекта «Объект F».

Теперь определим доминирование объектов над субъектами, для того чтобы определить, кто из субъектов может производить запись информации, и в какой объект.

Здесь только «Объект D» доминирует над субъектом «Субъект В», так как уровень секретности этого объекта не ниже, чем у субъекта «Субъект В», а все категории, перечисленные в метке безопасности субъекта, присутствуют в метке объекта. Это означает, что при данной политике безопасности только «Субъект В» может производить запись информации и только в «Объект D».

Принудительное управление доступом реализовано во многих вариантах операционных систем и СУБД, отличающихся повышенными мерами безопасности. Независимо от практического использования принципы принудительного управления являются удобным методологическим базисом для начальной классификации информации и распределения прав доступа.

Удобнее мыслить в терминах уровней секретности и категорий, чем заполнять неструктурированную матрицу доступа.

Однако такая политика безопасности является достаточно жесткой. Она рассчитана на статичные, замкнутые системы, которые, вероятно, доминируют в военной среде, но крайне редки в среде коммерческой. Поэтому в реальной практике иногда лучше сочетать сильные стороны как принудительного, так и произвольного управления доступом.

Гарантированность

Гарантированность – это мера уверенности, с которой можно утверждать, что для проведения в жизнь сформулированной политики безопасности выбран подходящий набор средств, и что каждое из этих средств правильно исполняет отведенную ему роль.

В Оранжевой книге рассматривается два вида гарантированности – *операционная* и *технологическая*. Операционная гарантированность относится к архитектурным и реализационным аспектам системы, в то время как технологическая – к методам построения и сопровождения.

Операционная гарантированность включает в себя проверку следующих элементов:

- Архитектура системы
- Целостность системы
- Анализ тайных каналов передачи информации
- Надежное администрирование
- Надежное восстановление после сбоев.

Операционная гарантированность – это способ убедиться в том, что архитектура системы и ее реализация действительно проводят в жизнь избранную политику безопасности.

Архитектура системы должна способствовать реализации мер безопасности или прямо поддерживать их. В принципе меры безопасности не обязательно должны быть заранее встроены в систему – достаточно принципиальной возможности дополнительной установки защитных продуктов. Например, сугубо ненадежная система MS-DOS может быть улучшена за счет средств проверки паролей доступа к компьютеру и/или жесткому диску, за счет борьбы с вирусами путем отслеживания попыток записи в загрузочный сектор CMOS-средствами и т. п.

Среди аппаратных решений, предусматриваемых Оранжевой книгой, можно выделить следующие:

- Деление аппаратных и системных функций по уровням привилегированности и контроль обмена информацией между уровнями.
- Защита различных процессов от взаимного влияния за счет механизма виртуальной памяти.

- Наличие средств управления доступом.
- Структурированность системы, явное выделение надежной вычислительной базы, обеспечение компактности этой базы.
- Следование принципу минимизации привилегий – каждому компоненту дается ровно столько привилегий, сколько необходимо для выполнения своих функций.
- Сегментация (в частности сегментация адресного пространства процессоров) как средство повышения надежности компонентов.

Целостность системы в данном контексте означает, что аппаратные и программные компоненты надежной вычислительной базы работают должным образом и что имеется аппаратное и программное обеспечение для периодической проверки целостности.

Анализ тайных каналов передачи информации – тема специфичная для режимных систем, когда главное – обеспечить конфиденциальность информации. Тайным называется канал передачи информации, не предназначенный для обычного использования.

Надежное администрирование означает, что должны быть логически выделены три роли – системного администратора, системного оператора и администратора безопасности. Физически эти обязанности может выполнять один человек, но, в соответствии с принципом минимизации привилегий, в каждый момент времени он должен выполнять только одну из трех ролей.

Надежное восстановление после сбоев – вещь необходимая, однако, ее реализация может быть сопряжена с серьезными техническими трудностями. Прежде всего должна быть сохранена целостность меток безопасности. В принципе возможна ситуация, когда в момент записи файла с секретной информацией происходит сбой. В результате, если файл окажется с неправильной меткой, то информация может быть скомпрометирована. Нельзя допускать промежуточных состояний, когда защитные механизмы полностью или частично отключены, а доступ пользователей разрешен.

Несмотря на важный методологический недостаток Оранжевой книги – явная ориентация на производителя и оценщика, а не покупателя систем, она стала эпохальным событием в области защиты коммерческих систем. Появился общепринятый понятийный базис, без которого даже обсуждение проблем безопасности было бы затруднительным.

Огромный идейный потенциал Оранжевой книги пока во многом остается невостребованным. Прежде всего это касается концепции технологической гарантированности, охватывающей весь жизненный цикл системы – от выработки спецификаций до фазы эксплуатации.

Следуя по пути интеграции, европейские страны приняли согласованные критерии оценки безопасности информационных технологий [2].

Принципиально новой чертой Европейских критериев является отсутствие априорных требований к условиям, в которых должна работать информационная система. В Оранжевой книге очевидна привязка к условиям

правительственной системы. Организация, запрашивающая сертификационные услуги, формулирует лишь оценки, то есть описывает условия, в которых должна работать система, возможные угрозы ее безопасности и предоставляемые ею защитные функции. Задача органа сертификации – оценить, насколько полно достигаются поставленные цели, то есть насколько корректны и эффективны архитектура и реализация механизмов безопасности в описанных условиях.

Европейские критерии рассматривают следующие составляющие информационной безопасности:

- Конфиденциальность, то есть защиту от несанкционированного получения информации.
- Целостность, то есть защиту от несанкционированного изменения информации.
- Доступность, то есть защиту от несанкционированного удержания информации и ресурсов.

В 1992 году Гостехкомиссия при Президенте РФ опубликовала пять Руководящих документов, посвященных проблеме защиты от несанкционированного доступа (НСД) к информации [3].

Идейной основой набора руководящих документов является «Концепция защиты средств вычислительной техники (СВТ) и автоматизированных систем (АС) от несанкционированного доступа к информации (НСД)».

В Концепции различаются понятия СВТ и АС. Более точно Концепция предусматривает существование двух относительно самостоятельных и, следовательно, имеющих отличие направлений в проблеме защиты информации от НСД. Это – направление, связанное с СВТ, и направление, связанное с АС.

Необходимость в различии понятий СВТ и АС связана с некоторыми принципиальными отличиями с точки зрения информационной безопасности. СВТ – это конкретная аппаратно-программная конфигурация, построенная с вполне определенными целями и функционирующая в известном окружении. АС – это аппаратно-программный пакет, который можно купить и по своему усмотрению встроить в ту или иную систему СВТ. То есть СВТ имеют конкретное окружение, которое можно определить и изучить сколь угодно детально, а АС должна быть рассчитана на использование в различных условиях. Угрозы для СВТ носят вполне конкретный характер, относительно угроз АС можно лишь строить предположения.

4. Общая схема абстрактной модели защиты информации

Любая модель защиты информации не может претендовать на полную гарантию от взлома. Это лишь некий абстракт, цель которого описать общую терминологию и критерии системы безопасности. Модель не дает ответа на вопрос, как безопасным образом строить систему, как наращивать отдельные компоненты и конфигурацию в целом.

Начиная с 1977 года, было предложено огромное количество абстрактных моделей защиты информации [1]. Самые популярные из них: модель Биба (1977 г.), модель Гогена-Мезигера(1982 г.), Сазерлендская модель (1986 г.), модель Кларка-Вилсона (1987 г. и 1989 г.).

Общую схему абстрактной модели защиты информации можно представить следующим образом (Рисунок 1.4):

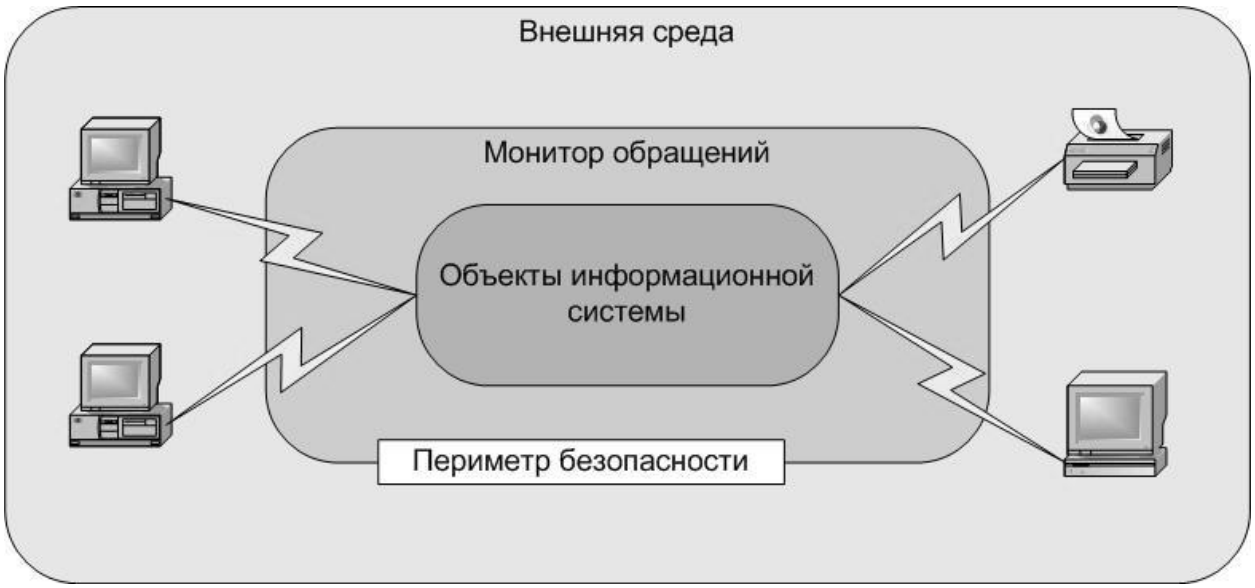


Рисунок 1.4. Общая схема абстрактной модели защиты информации.

Концепция надежной вычислительной базы является центральной при оценке степени гарантированности, с которой систему можно считать надежной. *Надежная вычислительная база* – это совокупность защитных механизмов компьютерной системы (включая аппаратное и программное обеспечение), отвечающих за проведение в жизнь политики безопасности. Надежность вычислительной базы определяется исключительно ее реализацией и корректностью исходных данных, которые вводит административный персонал.

Вообще говоря, компоненты вне вычислительной базы могут не быть надежными, однако это не должно влиять на безопасность системы в целом. Основное назначение надежной вычислительной базы – выполнять функции монитора обращений, то есть контролировать допустимость выполнения субъектами определенных операций над объектами. Монитор проверяет каждое обращение пользователя или процесса, запущенного от его имени, к программам и данным на предмет согласованности со списком действий, допустимых для пользователя.

От монитора обращений требуется выполнение трех свойств:

- *Изолированность* – монитор должен быть защищен от отслеживания своей работы.
- *Полнота* – монитор должен вызываться при каждом обращении и не должно быть способов его обхода.

- *Верифицируемость* – монитор должен быть компактным, чтобы его можно было проанализировать и протестировать, будучи уверенным в полноте его тестирования.

Реализация монитора обращений называется *ядром безопасности.* Ядро безопасности – это основа, на которой строятся все защитные механизмы. Помимо перечисленных выше свойств монитора обращений, ядро должно гарантировать собственную неизменность.

Границу надежной вычислительной базы называют *периметром безопасности*. От компонентов, лежащих вне периметра безопасности, не требуется надежности. То, что внутри ядра безопасности, считается надежным, а то, что вне – нет. Связь между внутренним и внешним мирами осуществляется посредством шлюзовой системы, которая по идее способна противостоять потенциально ненадежному или даже враждебному окружению.

5. Влияние систем защиты на потребительские свойства программного обеспечения

Описанная в предыдущем параграфе стратегия оценки критериев надежных систем, несомненно, играет положительную роль для разработчиков таких систем защиты. Она позволяет при помощи своего мощного теоретического базиса строить абстрактную модель для того, чтобы в дальнейшем снабдить ее конкретными средствами защиты. Но нельзя забывать и о потребительских свойствах создаваемой системы. Ведь какой бы надежной она не была, если такая система приносит неудобства, то рано или поздно от нее откажутся. Поэтому оценка критериев, отражающих удобство работы с системой безопасности, играет очень важную роль.

Можно выделить следующий набор критериев оценки средств защиты с точки зрения пользователя [4,5]:

- *Совместимость* – это отсутствие конфликтов с системным ПО, отсутствие конфликтов с прикладным ПО, отсутствие конфликтов с существующим аппаратным обеспечением, а также максимальная совместимость с будущим программным и аппаратным обеспечением.
- *Неудобство для конечного пользователя* – необходимость и сложность дополнительной настройки системы защиты, доступность документации, доступность информации об обновлении модулей системы защиты (из-за ошибок, несовместимости, нестойкости), доступность сервисных пакетов, безопасность сетевой передачи пароля или ключа, замедление работы основных функций системы.
- *Побочные эффекты* – перегрузка трафика, отказ в обслуживании, замедление работы операционной системы, захват системных ресурсов, перегрузка ОЗУ, нарушение стабильности работы системы.

- *Доброкачественность* – правдивая реклама, доступность результатов независимой экспертизы, доступность информации о побочных эффектах, доступность полной информации о системе защиты для конечного пользователя.

Важным фактором применимости системы защиты является ее *экономическая эффективность*, под которой в простейшем случае можно понимать абсолютную разницу либо соотношение потерь до и после установки системы, а также отношение затрат на разработку или приобретение защиты к приросту прибыли.

Кроме экономического показателя применимости можно выделить также *технические* и *организационные* показатели применимости. Под *техническим показателем применимости* следует понимать соответствие системы защиты функциональным требованиям по стойкости, системные требования ПО, функциональную направленность, а также наличие и тип систем защиты у аналогов ПО – конкурентов.

Организационный показатель применимости – это распространенность и популярность ПО, условия его распространения и использования, уникальность, вероятность превращения пользователя в злоумышленника, роль документации и поддержки при использовании ПО.

6. Принципы проектирования систем защиты

Исходя из вышесказанного, можно попытаться сформулировать основные принципы проектирования систем защиты:

1. Простота механизма защиты.
2. В механизме защиты при работе в нормальных условиях доступ должен разрешаться, а не запрещаться.
3. Все возможные каналы утечки информации должны быть перекрыты, то есть предполагается проверка полномочий любого обращения к любому объекту.
4. Сам механизм защиты можно не засекречивать, засекречивается только какая-то его часть, например списки паролей.
5. Установление для любого пользователя только тех полномочий, которые ему необходимы, то есть круг полномочий должен быть минимальным.
6. Обособленность или сведение к минимуму числа общих для нескольких пользователей параметров и характеристик защиты.
7. Психологическая привлекательность и простота использования системы.

Контрольные вопросы к первой главе

1. Дайте определение понятия «Защита компьютерной информации».
2. Укажите традиционные направления защиты компьютерной информации.
3. Что является объектом защиты информации?
4. Что является предметом защиты в компьютерных системах?
5. Дайте определение понятия «Надежная система».
6. Перечислите основные критерии, по которым оценивают надежность системы согласно «Оранжевой книге».

ГЛАВА 2. КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ЗАЩИТЫ ИНФОРМАЦИИ

О важности сохранения информации в тайне знали уже в древние времена, когда с появлением письменности появилась и опасность прочтения ее нежелательными лицами.

Существовали три основных способа защиты информации. Первый из них предполагал защиту чисто силовыми методами: охрана документа физическими лицами, передача его специальным курьером и т. п. Второй способ получил название «Стеганография» латино-греческое сочетание слов, означающих «тайнопись». Он заключался в сокрытии самого факта наличия информации. Например, использование симпатических чернил, которые становятся видимыми лишь при определенном воздействии на бумагу – яркий пример тайнописи. Но он появился несколько позже. Первые способы сокрытия информации были приведены в трудах древнегреческого историка Геродота. На голове раба, которая брилась наголо, записывалось нужное сообщение. И когда волосы его отрастали, раба отправляли к адресату, который снова брил его голову и считывал полученное сообщение.

Третий способ защиты информации заключался в преобразовании смыслового текста в некий набор хаотических знаков (или букв алфавита). Получатель данного донесения имел возможность преобразовать его в то же самое осмысленное сообщение, если обладал ключом к его построению. Этот способ защиты информации называется криптографическим (от греческого слова crypto – шифрую и graf – пишу). По утверждению ряда специалистов криптография по возрасту – ровесник египетских пирамид. В документах древних цивилизаций – Индии, Египта, Месопотамии – есть сведения о системах и способах составления шифровальных систем.

1. Обзор древних шифров и шифров средневековья

Наиболее полные и достоверные сведения о шифрах относятся к Древней Греции. Как правило, в древние времена использовались так называемые шифры замены и шифры перестановки.

Шифром замены называется криптографическое преобразование, при котором символы исходного текста заменяются символами шифротекста по какому-либо закону криптографического преобразования, но позиции символов в шифротексте не изменяются. В шифрах перестановки использовалось некоторое преобразование над исходным текстом, которое состояло в «перемешивании» букв или блоков по определенному закону.

Шифр Цезаря и его модификации

Историческим примером шифра замены является шифр Цезаря (1 век до н. э.), описанный историком Древнего Рима Светонием. Гай Юлий Цезарь использовал в своей переписке шифр собственного изобретения. Идея этого шифра состояла в следующем. Выписывался алфавит в обычном виде, а затем

под ним выписывался тот же алфавит, но со сдвигом на 3 буквы влево. Применительно к современному русскому языку это выглядело бы так:

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

При зашифровке буква А заменялась буквой Г, буква Б заменялась на Д и так далее. Например, слово «РИМ» превращалось в слово «УЛП». Получатель сообщения «УЛП» искал эти буквы в нижней строке и по буквам над ними восстанавливал исходное слово «РИМ». Ключом в шифре Цезаря является величина сдвига нижней строки алфавита. В нашем случае – это число 3. Для произвольного ключа k шифр Цезаря можно представить так: $C_i = (i + k) (\mathrm{mod} n)$, где n – количество букв в алфавите *(мощность алфавита)*, C_i – шифруемый символ. Тогда обратной подстановкой является $C_i^{-1} = (i + n - k) (\mathrm{mod} n)$.

Условием для успешной реализации этого метода является совпадение размера множеств открытого текста и шифротекста. Это условие в современных криптосистемах называется *гомоморфизмом*.

Позднее были предложены многочисленные модификации этого шифра, которые были направлены на повышение криптостойкости (устойчивости к дешифрованию).

Более эффективны обобщения подстановки Цезаря – шифр Хилла [6,7,8] и шифр Плэйфер [7–10]. Они основаны на подстановке не отдельных символов, а двуграмм (шифр Плэйфер) и n-грамм (шифр Хилла). При более высокой криптостойкости они значительно сложнее для реализации и требуют достаточно большого количества ключевой информации.

Шифр, называемый шифром Гронсфельда [9,11], состоит в модификации шифра Цезаря числовым ключом. Для этого под сообщением пишут числовой ключ. При шифровании сдвиг происходит не на постоянную величину, а на цифру, указанную под шифруемой буквой в ключе.

Еще одним обобщением системы Цезаря является аффинная криптосистема [9,10]. Она определяется двумя числами a и b, где $0 \le a,b \le n-1$, n – мощность алфавита. При этом числа a и b должны быть взаимно простыми. Соответствующими заменами являются: $C(a,b)_i = (a \cdot i + b) (\mathrm{mod} n)$ и $C^{-1}(a,b)_i = (i - b) \cdot a \, (\mathrm{mod} n)$.

Реализация алгоритма Цезаря на языке C++ для ASCII таблицы будет следующей:

```
// Функция шифрования
// char*  toCode    -    кодируемое сообщение
// int    key       -    ключ
// ret    char*     -    закодированное сообщение
char* CesarCrypt (char* toCode, int key)
{
    int i;
```

```
    for (i=0;toCode[i]!=0;i++)
    {
        int tmp;
        tmp = (toCode[i]+key);
        // Нормализация
        if (tmp>=256) tmp -= 256;
        toCode[i] = tmp;
    }
    return toCode;
}

// Функция дешифрования
// char* toDeCode   - сообщение, подлежащее раскодированию
// int key          - ключ
// ret  char*       - раскодированное сообщение

char* CesarEnCrypt (char* toDeCode, int key)
{
    int i;
    for (i=0;toDeCode[i]!=0;i++)
    {
        int tmp;
        tmp = (toDeCode[i]-key);
        // Нормализация
        if (tmp<0) tmp += 256;
        toDeCode[i] = tmp;
    }
    return toDeCode;
}
```

Шифр Атбаш

Пример еще одного шифра замены – это шифр Атбаш. Алгоритм этого шифра прост: первая буква алфавита заменялась на последнюю, вторая на предпоследнюю в алфавите и т. д. Для успешной дешифрации необходимо было знать только алфавит сообщения. По смыслу алгоритма функция, реализующая шифровку и зашифровку, одна и та же: $C_i = n - i$, где n – мощность алфавита.

Например, слово «ЗАМЕНА» выглядело бы после шифрования как «ХЮРЩПЮ».

Исходный текст алгоритма Атбаш для ASCII таблицы на языке C++:

```
// Функция шифрования/дешифрования
// char*  toCode         -    кодируемое сообщение
// ret    char*          -    закодированное сообщение
char* Atbash(char* toCode)
{
    int i;
    for (i=0;toCode[i]!=0;i++)
    {
```

```
            toCode[i] = (256-toCode[i]);
        }
        return toCode;
    }
```

Шифр моноалфавитной подстановки

Шифр моноалфавитной подстановки – это один из самых древних шифров на Земле. Шифр Цезаря является частным случаем этого шифра.

Прежде всего выбирается нормативный алфавит, то есть набор символов, которые будут использоваться при составлении сообщений, требующих зашифровки. Допустим, это будут прописные буквы русского алфавита (исключая буквы «Ё» и «Ъ») и пробел. Таким образом, нормативный алфавит будет состоять из 32 символов. Затем выбирается алфавит шифрования и устанавливается взаимно однозначное соответствие между символами нормативного алфавита и символами алфавита шифрования. Алфавит шифрования может состоять из произвольных символов, в том числе и из символов нормативного алфавита.

При шифровании исходного сообщения, каждый символ открытого текста заменяется соответствующим ему символом алфавита шифрования.

Нормативный алфавит	А Б В Г Д Е Ж З И Й К Л …
Алфавит шифрования	Н К А Л З Т П И О Р Г Б …

Например, слово «ЗВЕЗДА» в зашифрованном виде будет выглядеть как «ИАТИЗН».

Метод моноалфавитной замены можно представить как числовые преобразования символов исходного текста. Для этого каждой букве нормативного алфавита ставится в соответствие некоторое число, называемое числовым эквивалентом этой буквы. Тогда моноалфавитные подстановки можно описать выражением: $C_i = (M_i + S_i)(\mathrm{mod} n)$, где C_i, M_i – числовые эквиваленты символов алфавита шифрования и нормативного алфавита соответственно, S_i – коэффициент сдвига, n – мощность алфавита.

В художественной литературе классическим примером шифра замены является известный шифр «Пляшущие человечки» К. Дойля. В нем буквы текста заменялись символическими фигурками людей. Ключом такого шифра являлись позы человечков, заменяющих буквы.

В случае использования алфавита, содержащего только латинские прописные, буквы можно предложить такой алгоритм на языке С++:

```
// Нормативный алфавит
const char* norm  = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
// Алфавит шифрования
const char* shifr = "7d*a1j/f6oeg(,.)mb2&#p?{]@";
// В качестве сообщения могут использоваться
// только заглавные буквы латинского алфавита
// Функция шифрования
```

```
// char*  toCode    -    кодируемое сообщение
// ret    char*     -    закодированное сообщение
char* MonoCrypt(char* toCode)
{
    int i,j;
    for (i=0;toCode[i]!=0;i++)
    {
        for (j=0; norm[j]!=0;j++)
            if (toCode[i] == norm[j])
                toCode[i] = shifr[j];
    }
    return toCode;
}
// Функция дешифрования
// char* toDeCode   -  сообщение, подлежащее раскодированию
// ret  char*       - раскодированное сообщение
char* MonoEnCrypt(char* toDeCode)
{
    int i,j;
    for (i=0;toDeCode[i]!=0;i++)
    {
        for (j=0; norm[j]!=0;j++)
            if (toDeCode[i] == shifr[j])
                toDeCode[i] = norm[j];
    }
    return toDeCode;
}
```

«Квадрат Полибия»

Система Цезаря не является старейшей. Возможно, что наиболее древней из известных является система греческого историка Полибия, умершего за 30 лет до рождения Цезаря. Устройство для шифрования, называемое квадратом Полибия, или полибианским квадратом, представляло собой квадрат размером 5 на 5 (на самом деле, размер этого квадрата зависит от мощности используемого алфавита). В каждую клетку этого квадрата вписывалась буква в порядке её следования в алфавите (Рисунок 2.1).

В процессе шифрования каждой букве ставилась в соответствие пара чисел – это номер столбца и номер строки, на пересечении которых располагалась шифруемая буква. Так для латинского алфавита буква «O» представлялась как 34 (3 – это номер строки, в которой находится буква «O», 4 – это номер столбца), а для русского алфавита, буква «Т» – это 14.

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

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

Рисунок 2.1. «Полибианский квадрат» для латинского (а) и русского (б) алфавитов

Таким образом, зашифрованное сообщение представлялось в виде последовательности цифр, для расшифровки которого необходимо знать язык сообщения, порядок следования букв в алфавите и размер квадрата. Эта информация и является ключом. Очевидно, что такой способ не обладает достаточной криптостойкостью в силу отсутствия уникального ключа. Кроме того, при шифровании таким способом происходит увеличение размера кодируемого сообщения, так как вместо одной буквы получается две цифры.

Модификацией метода шифрования при помощи полибианского квадрата является так называемый «Тюремный шифр». Строго говоря, это не шифр, а способ перекодировки сообщения с целью его приведения к виду, удобному для передачи по каналу связи (через стенку). Дело в том, что в квадрате Полибия используется естественный порядок следования букв, так как при произвольном расположении возникает затруднение: либо нужно помнить отправителю и получателю сообщения заданный порядок следования букв,

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

Рисунок 2.2. «Полибианский квадрат» модифицированный по методу «Тюремного шифра»

либо иметь его при себе в виде записей, но в этом случае имеется возможность прочтения посторонними лицами ключа. Поэтому в ряде случаев ключ составляется следующим образом. Берется некоторое ключевое слово, например «КРИПТОГРАФИЯ». Затем из этого слова удаляются повторяющиеся буквы (получается слово «КРИПТОГАФЯ») и оно записывается в начальные клетки квадрата. В оставшиеся клетки записываются остальные буквы алфавита в обычном порядке следования, но из них удаляются те буквы, которые присутствуют в ключевом слове. Для данного примера получается такой квадрат (Рисунок 2.2):

Далее шифрование проводилось по такой же схеме, как и для обычного полибианского квадрата.

Шифр скитала

Известно, что в V веке до нашей эры правители Спарты, наиболее воинственного из греческих государств, имели хорошо отработанную систему секретной военной связи и шифровали свои послания с помощью скитала – одного из первых простейших криптографических устройств, реализующего метод простой перестановки.

Шифрование выполнялось следующим образом. На стержень цилиндрической формы, который назывался скитала, наматывали спиралью несколько строк текста сообщения, написанного на ленте из кожи (Рисунок 2-3). Таким образом, сообщение «НАСТУПАЙТЕ» при размещении его по окружности стержня по две буквы дает шифротекст «НПААСЙТТУЕ». Для расшифровки такого сообщения нужно не только знать правило шифрования, но и обладать ключом в виде стержня определенного диаметра. Зная только вид шифра, но, не имея ключа, расшифровать сообщение было не просто. Шифр скитала многократно совершенствовался в последующие времена.

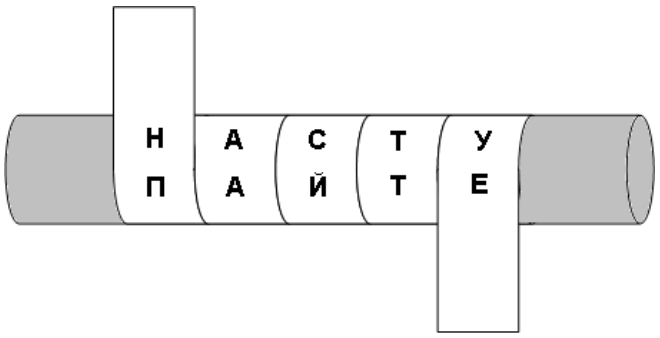


Рисунок 2-3. Шифр скитала

Интересно, что изобретение первого дешифровального устройства, называемого «Антискитала», приписывается великому Аристотелю. Он предложил для этого использовать конусообразное «копье», на которое наматывался перехваченный ремень и передвигался по оси до того положения, пока не появлялся осмысленный текст.

На основе этого метода было предложено достаточно большое количество модификаций [6, 7,8,11]. Наиболее практический метод шифрования, называемый *одиночной перестановкой,* использовался в эпоху

Возрождения. Текст сообщения записывался в таблицу по столбцам, а затем для определения перестановки столбцов использовалось некоторое ключевое слово. Например, фраза «НЕЯСНОЕ СТАНОВИТСЯ ЕЩЕ БОЛЕЕ НЕПОНЯТНЫМ» сначала записывалось в таблицу, размер которой должен был быть известен как отправителю, так и получателю этого сообщения. Далее подбиралось ключевое слово, длина которого соответствовала количеству столбцов в таблице (если таблица была слишком большой, то ключевое слово циклически повторялось). Сверху таблица дополнялась двумя пустыми строками, первую из которых записывалось ключевое слово (например «ЛУНАТИК»), а во вторую – порядок букв в алфавите из ключевого слова (Рисунок 2.4,а). Если в ключе встречаются одинаковые буквы, то они нумеруются слева направо. После этого столбцы в таблице перестанавливались так, чтобы номера во второй строке были отсортированы по возрастанию (или убыванию) (Рисунок 2.4,б).

Для дополнительной скрытности сообщение шифровалось повторно. Этот способ известен под названием *двойная перестановка*. Чаще всего для этого

Л	У	Н	А	Т	И	К
4	7	5	1	6	2	3
Н	О	Н	С	Б	Н	Я
Е	Е	О	Я	О	Е	Т
Я	С	В	Е	Л	П	Н
С	Т	И	Щ	Е	О	Ы
Н	А	Т	Е	Е	Н	М

а

А	И	К	Л	Н	Т	У
1	2	3	4	5	6	7
С	Н	Я	Н	Н	Б	О
Я	Е	Т	Е	О	О	Е
Е	П	Н	Я	В	Л	С
Щ	О	Ы	С	И	Е	Т
Е	Н	М	Н	Т	Е	А

б

Рисунок 2.4. Таблица до перестановки (а) и таблица после перестановки (б)

размер второй таблицы подбирался таким образом, чтобы длины её строк и столбцов были другие, чем в первой таблице. Кроме того, в первой таблице можно переставлять столбцы, а во второй строки. Наконец можно заполнять таблицу зигзагом, змейкой, по спирали или каким-то другим способом.

В результате получается зашифрованное сообщение «СНЯНН БОЯЕТ ЕООЕЕ ПНЯВЛ СЩОЫС ИЕТЕН МНТЕА». Объединение букв в группы (в данном случае по 5 букв) не входит в ключ шифра и используется только для удобства записи несмыслового текста.

Магический квадрат

Магическими квадратами пользовались еще в Средневековье. Магическим квадратом называются квадратные таблицы со вписанными в их клетки последовательными натуральными числами от 1, которые дают в сумме по каждому столбцу, каждой строке и каждой диагонали одно и то же число.

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

а

О	И	Р	Т
З	Ш	Е	Ю
	Ж	А	С
Е	Г	О	П

б

Рисунок 2.5.Пример магического квадрата (а) и сообщение, записанное в него (б)

Шифруемый текст вписывался в квадрат по приведенной в нем нумерации. Если потом выписывать содержимое таблицы по строкам, то получится шифровка перестановкой букв. Считалось, что созданные таким способом шифровки охраняет не только ключ, но и магическая сила.

Пример магического квадрата 4 на 4 (Рисунок 2.5,а) и его шифровки (Рисунок 2.5,б) для сообщения «ПРИЕЗЖАЮ ШЕСТОГО» будет такой: «ОИРТЗШЕЮ ЖАСЕГОП».

На первый взгляд кажется, что магических квадратов очень мало. Тем не менее, их число возрастает с размерностью таблицы. Так существует лишь один магический квадрат для таблицы 3 на 3, если не принимать во внимание его повороты. Магических квадратов 4 на 4 насчитывается уже 880, а число магических квадратов 5 на 5 около 250 000. Поэтому магические квадраты больших размеров могли быть хорошей основой для надежной системы шифрования того времени, потому что ручной перебор всех вариантов ключа для этого шифра был немыслимым.

Книжный шифр

В XIX вв. был очень распространен книжный шифр. Это было связано с появлением первых периодических печатных изданий. Идея шифра была достаточно проста. Для шифрования обеим сторонам нужно было обладать одинаковым печатным изданием (книгой, газетой, журналом и т. п.). Процесс шифрования сообщения состоял в указании номера страницы, номера строки и столбца, в котором располагалась шифруемая буква сообщения. Если такое сообщение было перехвачено, то расшифровать его без помощи ключа, которым являлось название книги или газеты, было практически невозможно.

2. Основные понятия и термины современной криптографии

В настоящее время большинство средств защиты информации базируется на использовании криптографических шифров и процедур шифрования-расшифрования. Эти процессы происходят в рамках некоторой криптосистемы, которая диктует правила и определяет параметры шифрования и дешифрования.

Криптография – наука о защите информации от прочтения ее посторонними лицами. Защита достигается путем шифрования, которое делает защищенные данные трудноraскрываемыми без знания специальной информации.

В соответствии со стандартом ГОСТ 28147–89 понятию «шифр» дается следующее определение:

Шифр – это совокупность обратимых преобразований множества открытых данных на множество зашифрованных данных, задаваемых ключом и алгоритмом преобразования.

Как видно из определения, в шифре можно выделить два основных элемента – это *ключ* и *алгоритм*.

Ключ – конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования данных, обеспечивающий выбор одного варианта из совокупности возможных для данного алгоритма.

Алгоритм (функция, уравнение) шифрования – соотношение, описывающее процесс образования зашифрованных данных из открытых.

Для определения качества шифра используется понятие *криптостойкость*.

Криптостойкость – это характеристика шифра, определяющая ее стойкость к дешифрованию.

3. Теория секретных систем

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

Согласно К. Шеннону [12] существуют три общих типа секретных систем:

1. *Системы маскировки*, при помощи которых скрывается сам факт наличия сообщения (стеганография). Например, невидимые чернила или маскировка сообщения за безобидным текстом.
2. *Тайные системы*, в которых для раскрытия сообщения требуется специальное оборудование. Например, инвертирование речи.

3. *Криптографические системы,* где смысл сообщения скрывается при помощи шифра, кода и т. п., но само существование сообщения не скрывается.

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

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

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

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

Понятие криптографической системы

Общий вид криптографической системы можно представить следующим образом (Рисунок 2.6).

Для использования такой системы для определенного сообщения M_i выбирается некоторый ключ K_i из множества возможных ключей K. После чего при помощи ключа K_i формируется криптограмма E_i. Эта криптограмма, полученная при помощи преобразования T_{Ki}, по каналу передачи передается в точку приема. На приемном конце с помощью отображения T_{Ki}^{-1}, обратного выбранному, из криптограммы E_i восстанавливается исходное сообщение M_i.

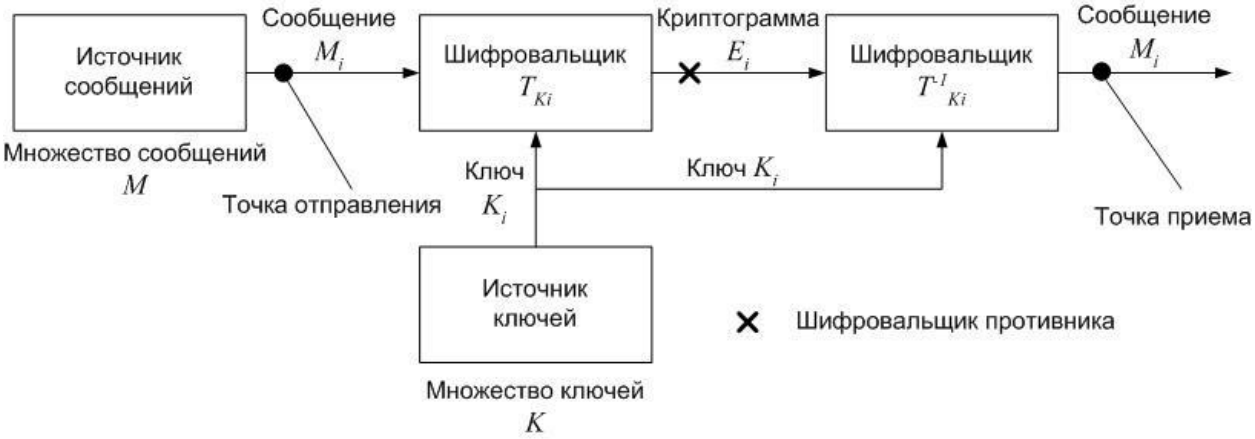


Рисунок 2.6. Общая схема криптографической системы

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

Вычисление апостериорных вероятностей – есть общая задача дешифрования.

Например, в шифре простой подстановки со случайным ключом для английского языка имеется *26!* отображений, соответствующих *26!* способам, которыми мы можем заменить *26* различных букв. Все эти способы равновозможные, и поэтому каждый имеет априорную вероятность $P = 1/26! \approx 2.48 \cdot 10^{-27}$.

Если противник ничего не знает об источнике сообщений, кроме того, что он создает английский текст, то апостериорными вероятностями различных сообщений из *N* букв являются просто их относительные частоты в нормативном английском языке.

Если *N* достаточно велико (≈ *50* букв), имеется обычно единственное сообщение с апостериорной вероятностью, близкой к единице $P(K_u)$, в то время как все другие сообщения имеют вероятность, близкую к нулю $P(K_{oi})$, то есть, по существу, имеется единственное «решение» такой криптосистемы.

$P(K_u) >> P(K_{oi})$, где K_u – истинный ключ, K_{oi} – ошибочные ключи.

Для меньших *N* (≈ *15* букв) обычно найдется много сообщений и ключей, вероятности которых сравнимы $P(K_i)$, и не найдется ни одного сообщения и ключа с вероятностью, близкой к единице. В этом случае решение криптосистемы невозможно.

$P(K_1) \approx P(K_2) \approx P(K_3) \approx ... \approx P(K_m)$, где K_i – возможные ключи.

Пример вычисления апостериорных вероятностей

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

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

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

№	Множество возможных криптограмм	*P(A)*, %	*P(B)*, %	*P(C)*, %
1	A B A C A C A B C A	50	20	30
2	B A B C B C B A C B	20	50	30
3	B C B A B A B C A B	30	50	20
4	C B C A C A C B A C	30	20	50
5	C A C B C B C A B C	20	30	50
6	A C A B A B A C B A	50	30	20

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

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

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

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

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

1. Информация об источнике сообщений неполная или ее вообще нет.
2. Мощность множества возможных ключей настолько велика, что перебор всех возможных значений займет слишком много времени (для алфавита в *256* символов мощность множества ключей в алгоритме простой перестановки составит $256! \approx 8.578 \cdot 10^{506}$).
3. Вероятность использования символов может быть либо неизвестной (неизвестный язык источника сообщений), либо выражаться нечетко (метод использования имитовставок, когда в шифруемый текст преднамеренно вводится лишняя информация с целью «потопления статистики»).
4. Схема, по которой осуществлялось шифрование, неизвестна, либо достаточно сложна.

Основные характеристики секретных систем

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

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

5. *Увеличение объема сообщения* – в некоторых системах объем сообщения увеличивается в результате операции шифрования с целью «потопления статистики», этот нежелательный эффект нужно пытаться минимизировать.

Комбинирование секретных систем

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

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

1. Взвешенная сумма

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

$$S = pT + qR, \qquad p + q = 1$$

где p – вероятность использования системы T, q – вероятность использования системы R.

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

$$T = p_1T_1 + p_2T_2 + p_3T_3 + \ldots + p_mT_m,$$

где T_i – определенная операция шифрования в системе T, соответствующая выбору ключа i, причем вероятность такого выбора равна p_i.

Обобщая далее, можно образовать сумму нескольких систем:

$$S = p_1T + p_2R + p_3Q + \ldots + p_mU, \qquad \sum p_i = 1.$$

2. Произведение

Образование произведения двух секретных систем (Рисунок 2.7) осуществляется следующим образом:

$S = RT$, причем $RS = SR$, а $RS \neq RS$.

Рисунок 2.7. Произведение секретных систем

То есть сначала применяется система T, а затем система R к результатам первой операции.

Ключ системы S состоит как из ключа системы T, так и из ключа системы R.

4. Классификация современных криптосистем

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

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

Все многообразие симметричных криптографических систем (Рисунок 2.8) основывается на следующих базовых классах.

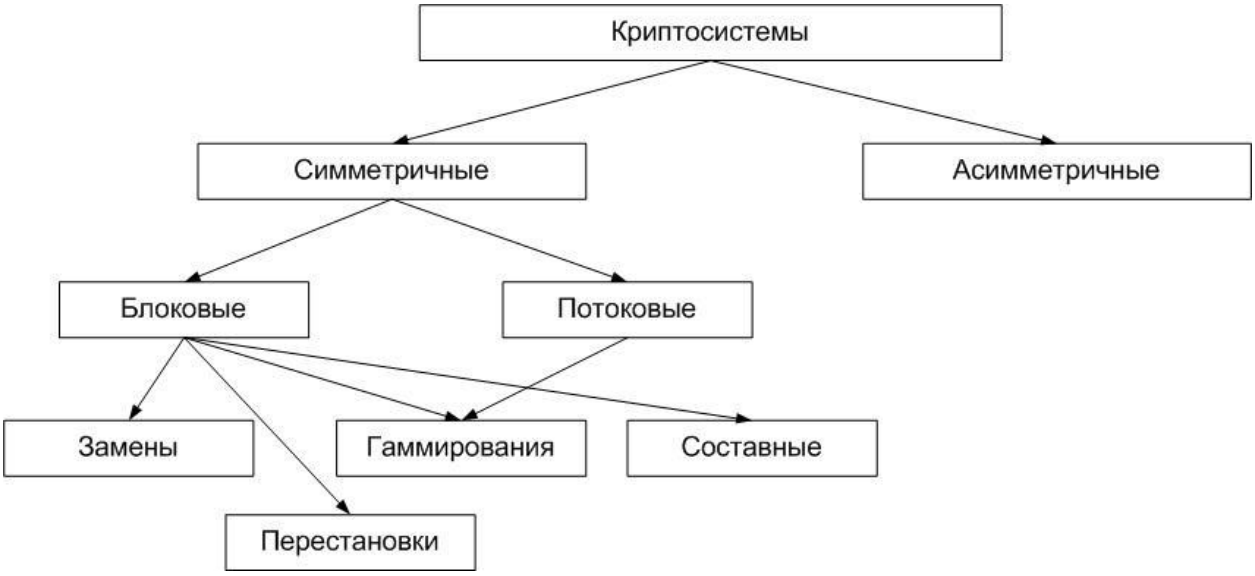


Рисунок 2.8. Классификация криптосистем

Блочные шифры.

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

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

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

Шифры замены (подстановки).

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

Шифры перестановки.

Перестановки – метод криптографического преобразования, заключающийся в перестановке местами символов исходного текста по

некоторому правилу. Шифры перестановки в настоящее время не используются в чистом виде, так как их криптостойкость недостаточна.

Гаммирование.

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

Потоковые шифры.

Потоковые шифры представляют собой разновидность гаммирования и преобразуют открытый текст в шифрованный последовательно, по одному биту. *Генератор ключевой последовательности,* иногда называемый *генератором бегущего ключа*, выдает последовательность бит $k_1, k_2, ..., k_i, ...$. Эта ключевая последовательность складывается по модулю *2* с последовательностью бит исходного текста $p_1, p_2, ..., p_i, ...$ для получения шифрованного текста $c_i = p_i \oplus k_i$. На приемной стороне текст складывается по модулю *2* с идентичной ключевой последовательностью для получения исходного текста. Такое преобразование называется гаммированием с помощью операции XOR. Однако при потоковом шифровании для повышения криптостойкости генератор ключевой последовательности «завязывается» на текущее состояние кодируемого символа. То есть значения, выдаваемые генератором, зависят не только от ключа, но и от номера шифруемого бита и входной последовательности.

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

5. Методы программной генерации случайных чисел

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

Однако если каждый раз при включении генератор будет выдавать одну и ту же последовательность, то взлом криптосистемы будет тривиальной задачей. Например, в случае использования потокового шифрования, перехватив два шифрованных текста, злоумышленник может сложить их по модулю *2* и получить два исходных текста, сложенных также по модулю *2*. Такую систему

раскрыть очень просто. Если же в руках противника окажется пара «исходный текст – шифрованный текст», то задача вообще становится тривиальной.

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

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

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

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

Использование стандартных функций языков высокого уровня

Функция Rand() выдает псевдослучайное число в указанном диапазоне (способ задания диапазона отличается в различных языках). Начальная инициализация генератора случайных чисел происходит при помощи системного вызова специальной функции. Для языка C – это функция srand, в Object Pascal задание начального значения реализовано через свойство RandSeed. Часто значение, используемое в качестве начального, называют *затравкой* генератора.

В реальных криптосистемах эта возможность не используется, так как обладает низкой криптостойкостью в силу своей доступности.

Конгруэнтные генераторы

Линейными конгруэнтными генераторами являются генераторы следующей формы:

$$X_n = (aX_{n-1} + b) \bmod m,$$

в которых X_n –n-й член последовательности, а X_{n-1} – предыдущий член последовательности. Переменные a, b и m – постоянные: a – множитель, b – инкремент и m – модуль. Ключом или затравкой служит значение X_0.

Период такого генератора не больше, чем m. Если a, b и m подобраны правильно, то генератор будет *генератором с максимальным периодом*, и его период будет равен m. (Например, для линейного конгруэнтного генератора b должно быть взаимно простым с m).

Если инкремент *b* равен нулю, то есть генератор имеет вид

$$X_n = (aX_{n-1}) \bmod m,$$

и мы получим самую простую последовательность, которую можно предложить для генератора с равномерным распределением. При соответствующем выборе констант $a = 7^5 = 16\,807$ и $m = 2^{31}-1 = 2\,147\,483\,647$ мы получим генератор с максимальным периодом повторения. Эти константы были предложены учеными Парком и Миллером, поэтому генератор вида

$$X_n = (7^5 \cdot X_{n-1}) \bmod 2^{31} - 1$$

называется *генератором Парка-Миллера*.

Преимуществом линейных конгруэнтных генераторов является их быстрота за счет малого количества операций на байт и простота реализации. К сожалению, такие генераторы в криптографии используются достаточно редко, поскольку являются предсказуемыми.

Иногда используют квадратичные и кубические конгруэнтные генераторы, которые обладают большей стойкостью к взлому.

Квадратичный конгруэнтный генератор имеет вид

$$X_n = (aX_{n-1}^2 + bX_{n-1} + c) \bmod m.$$

Кубический конгруэнтный генератор задается как

$$X_n = (aX_{n-1}^3 + bX_{n-1}^2 + cX_{n-1} + d) \bmod m.$$

Для увеличения размера периода повторения конгруэнтных генераторов часто используют их объединение [13]. При этом криптографическая безопасность не уменьшается, но такие генераторы обладают лучшими характеристиками в некоторых статистических тестах.

Пример такого объединения для 32-битовой архитектуры может быть реализован так:

```
// Long должно быть 32-х битовым целым
static long s1 = 1;
static long s2 = 1;
//MODMULT рассчитывает s*b mod m при условии, что m=a*b+c и 0<=c<m
#define MODMULT(a,b,c,m,s) q = s/a; s = b*(s-a*q)-c*q;
if (s<0) s+=m;
double combinedLCG (void)
  {
    long q;
    long z;
    MODMULT (53668, 40014, 12211, 2147483563L, s1)
    MODMULT (52774, 40692, 3791, 2147483399L, s2)
     z = s1 - s2;
     if (z<1)
        z += 2147483562;
     return z*4.656613e-10;
```

```
 }
void InitLCG (long InitS1, long InitS2)
  {
     s1 = InitS1;
     s2 = InitS2;
  }
```

Этот генератор работает при условии, что архитектура компьютера позволяет представлять все целые числа между $-2^{31}+85$ и $2^{31}-249$. Переменные *s1* и *s2* глобальные и содержат текущее состояние генератора. Перед первым вызовом их необходимо проинициализировать при помощи функции *InitLCG*. Для переменной *s1* начальное значение должно лежать в диапазоне между *1* и *2 147 483 562*, для переменной *s2* – между *1* и *2147483398*. Период такого генератора близок к 10^{18}. Функция *combinedLCG* возвращает действительное псевдослучайное число в диапазоне *(0,1)*. Она объединяет линейные конгруэнтные генераторы с периодами $2^{15}-405$, $2^{15}-1041$ и $2^{15}-1111$, и ее период равен произведению этих трех простых чисел.

Сдвиговые регистры с обратной связью

Сдвиговый регистр с обратной связью (FSR) состоит из двух частей: *сдвигового регистра* и *функции обратной связи*.

Рисунок 2.9. Сдвиговый регистр с обратной связью (FSR)

Сдвиговый регистр (Рисунок 2.9) представляет собой последовательность битов. Когда нужно извлечь бит, все биты сдвигового регистра сдвигаются вправо на 1 позицию. Новый крайний левый бит является значением функции обратной связи от остальных битов регистра. *Периодом* сдвигового регистра называется длина получаемой последовательности до начала ее повторения.

Простейшим видом сдвигового регистра с обратной связью является *линейный сдвиговый регистр с обратной связью (LFSR – Left Feedback Shift Register)*(Рисунок 2.10). Обратная связь представляет собой просто XOR

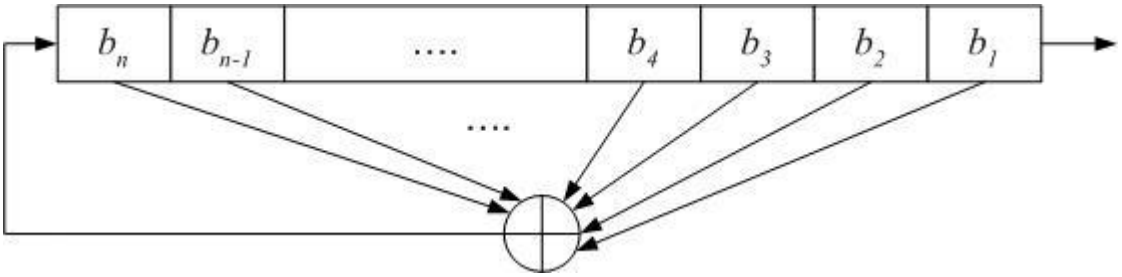


Рисунок 2.10. Сдвиговый регистр с линейной обратной связью (LFSR)

некоторых битов регистра, перечень этих битов называется *отводной последовательностью*.

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

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

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

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

```
int LFSR (void)
{
  static unsigned long ShiftRegister = 1;
  // Любое значение, кроме нуля
  ShiftRegister = ((((ShiftRegister >> 31)
      ^ (ShiftRegister >> 6)
      ^ (ShiftRegister >> 4)
      ^ (ShiftRegister >> 2)
      ^ (ShiftRegister >> 1)
      ^ ShiftRegister)& 0x00000001) <<31)
      | (ShiftRegister >> 1);
  return ShiftRegister & 0x00000001;  }
```

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

Схему обратной связи также можно модифицировать. При этом генератор не будет обладать большей криптостойкостью, но его будет легче реализовать программно. Вместо использования для генерации нового крайнего левого бита битов отводной последовательности выполняется XOR каждого бита отводной последовательности с выходом генератора и замена его результатом этого действия, затем результат генератора становится новым левым крайним битом (Рисунок 2.11).

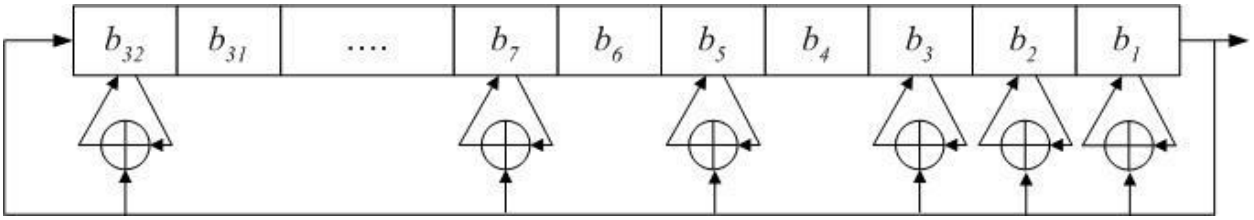


Рисунок 2.11. LFSR Галуа

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

```
static unsigned long ShiftRegister = 1;
void seed_LFSR (unsigned long seed)
{
  if (seed ==0)
    seed = 1;
  ShiftRegister = seed;
}

int Galua_LFSR (void)
{
  if (ShiftRegister & 0x00000001) {
    ShiftRegister = (ShiftRegister ^ mask >> 1) | 0x8000000;
    return 1;
  } else {
    ShiftRegister >>= 1;
    return 0;
  }
}
```

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

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

Генератор Геффа

Одним из комбинирующих генераторов является генератор Геффа (Рисунок 2.12). В нем используются три LFSR, объединенные нелинейным способом. LFSR-2 и LFSR-3 являются входами мультиплексора *(рабочие регистры)*, а третий управляет входом мультиплексора. Если длины LFSR равны n_1, n_2, n_3 соответственно, то линейная сложность генератора равна $(n_1+1)\cdot n_2 + n_1 \cdot n_3$.

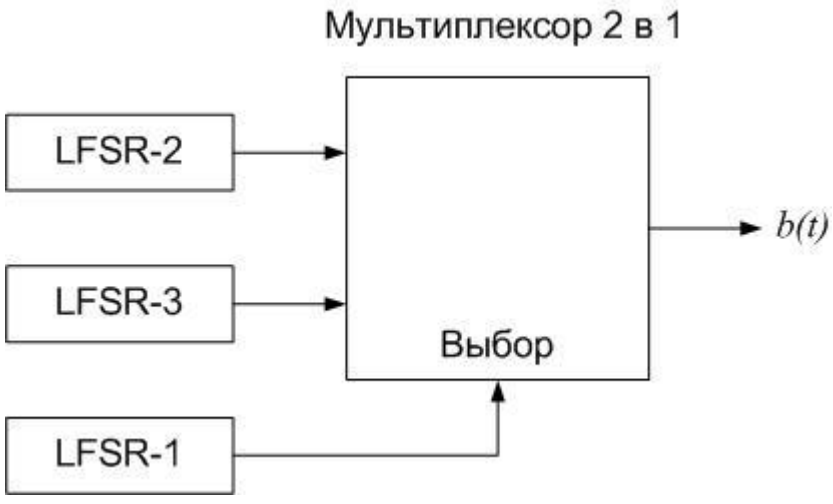


Рисунок 2.12. Генератор Геффа

Период такого генератора будет равен наименьшему общему делителю периодов трех генераторов. При условии, что размеры регистров взаимно просты, то период этого генератора будет равен произведению

периодов трех LFSR. В обобщенной схеме генератора Геффа используются несколько рабочих LFSR.

Генератор «Стоп-пошел»

Этот генератор использует выход одного LFSR для управления тактовой частотой другого LFSR. Тактовый выход LFSR-2 управляется выходом LFSR-1, так что LFSR-2 может изменять свое состояние в момент времени *t* только, если выход LFSR-1 в момент времени *t–1* был равен *1*.

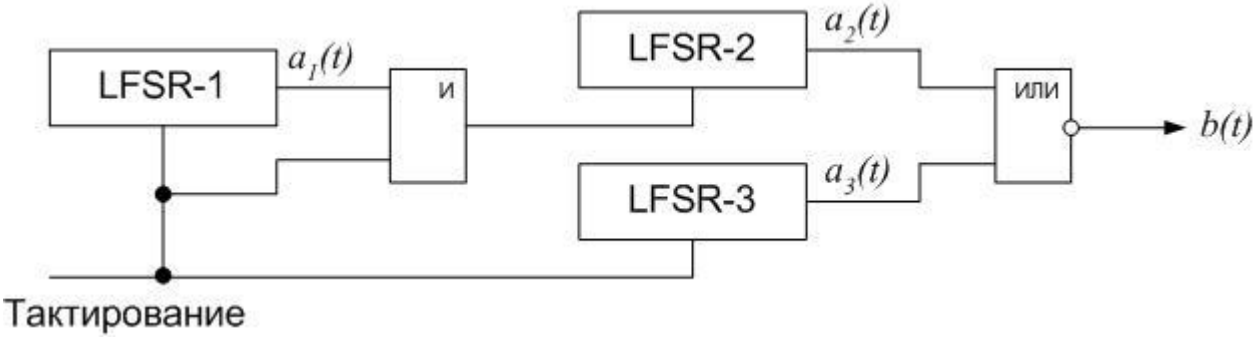


Рисунок 2.13. Генератор «Стоп-пошел»

Аддитивные генераторы

Аддитивные генераторы (называемые иногда запаздывающими генераторами Фиббоначи) очень эффективны, так как их результатом являются случайные слова, а не биты.

Начальное состояние генератора представляет собой массив *n*-битовых слов $X_1,X_2,X_3, \ldots,X_m$. Это первоначальное состояние и является ключом. *i*-е слово генератора получается как

$$X_i = \left(X_{i-a} + X_{i-b} + X_{i-c} + \ldots + X_{i-m}\right) \bmod 2^n.$$

При правильном выборе коэффициентов *a,b,c, ...,m* период этого генератора не меньше 2^n-1. Для этого должно выполнятся условие взаимной простоты коэффициентов *a,b,c, ...,m*. Например, если $a = 55$, а $b = 24$, то мы получим аддитивные генератор с максимальным периодом повторения вида:

$$X_i = \left(X_{i-55} + X_{i-24}\right) \bmod 2^n.$$

Существует несколько модификаций аддитивных генераторов. Самые известные из них – ish, Pike, Mush [12,13].

6. Генераторы реальных случайных последовательностей

Иногда криптографические псевдослучайные последовательности недостаточно хороши, так как генератор – это слабое звено большинства криптосистем. Для любого генератора важным вопросом является его проверка. Тесты на случайность можно найти в Internet. Доказано, что все из описанных выше генераторов можно воспроизвести. Это только вопрос времени.

Поэтому для получения действительно случайных чисел чаще всего используются естественные случайности реального мира. Часто такой метод требует специальной аппаратуры, но его можно применить в компьютерах.

К основным способам получения реальных случайных последовательностей относятся следующие:

1. Использование специальных таблиц RAND.
2. Использование случайного шума.
3. Использование таймера компьютера.
4. Измерение скрытого состояния клавиатуры.
5. Аппаратно-временные характеристики компьютера:
   - положение мыши на экране монитора;
   - текущий номер сектора и дорожки дисковода или винчестера;
   - номер текущей строки развертки монитора;
   - времена поступления сетевых пакетов;
   - выход микрофона.

7. Потоковые шифры

В основу потокового шифрования положена идея использования операции сложения по модулю два (исключающее «или», xor) исходного текста с некоторой гаммой. Гамма создается с помощью генераторов псевдослучайных чисел.

Процесс *дешифрования* данных сводится к повторной генерации гаммы шифра при известном ключе и наложении такой гаммы на зашифрованные данные.

Полученный зашифрованный текст является достаточно трудным для раскрытия в том случае, если гамма шифра не содержит повторяющихся битовых последовательностей. По сути дела гамма шифра должна изменяться случайным образом для каждого шифруемого слова. Фактически же, если период гаммы превышает длину всего зашифрованного текста и неизвестна никакая часть исходного текста, то шифр можно раскрыть только прямым перебором (пробой на ключ). Криптостойкость в этом случае определяется размером ключа.

Метод гаммирования становится бессильным, если злоумышленнику становится известен фрагмент исходного текста и соответствующая ему шифрограмма. Простым вычитанием по модулю получается отрезок псевдослучайной последовательности и по нему восстанавливается вся последовательность. Злоумышленники может сделать это на основе догадок о содержании исходного текста. Так, если большинство посылаемых сообщений начинается со слов “СОВ.СЕКРЕТНО”, то криптоанализ всего текста значительно облегчается. Это следует учитывать при создании реальных систем информационной безопасности [13].

Пример реализации и использования генераторов псевдослучайных для шифрования показан в лабораторной работе №1 «Потоковые шифры».

8. Лабораторная работа №1 «Потоковые шифры»

Задание:

1. Изучить алгоритмы хеширования паролей.
2. Изучить известные алгоритмы работы генераторов случайных чисел.
3. Реализовать свой упрощенный вариант алгоритма хеширования пароля согласно варианту (по таблице 1)
4. Реализовать свой алгоритм генератора случайных чисел согласно варианту.
5. Проанализировать выходную последовательность, выдаваемую генератором при различных параметрах.

Дополнительные требования к лабораторной работе:

1. Паролем может быть ЛЮБАЯ последовательность символов (русских и английских, цифр, знаков препинания и т. д.). Схема хеширования пароля также берется из лабораторной работы №1
2. Текст программы оформляется согласно общепринятым правилам (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
3. В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса генерации.
4. Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек VCL)
5. Среда разработки и язык программирования могут быть произвольными.

Требования для сдачи лабораторной работы:

1. Демонстрация работы реализованной вами системы.
2. Авторство.
3. Теория (ориентирование по алгоритму и теоретическим аспектам).
4. Оформление и представление письменного отчета по лабораторной работе, который содержит:
   - Титульный лист
   - Задание на лабораторную работу
   - Описание используемых алгоритмов шифрования
   - Листинг программы
   - Выводы

Варианты заданий

Таблица 1.

№ варианта	Тип генератора	Диапазон значений
1.	Конгруэнтный генератор	0-255
2.	Генератор Парка-Миллера	0-65535
3.	Генератор Геффа	0-4294967296
4.	Аддитивный генератор	0-255
5.	Конгруэнтный генератор	0-4294967296
6.	Генератор Парка-Миллера	0-65535
7.	Генератор Геффа	0-255
8.	Аддитивный генератор	0-65535
9.	Конгруэнтный генератор	0-4294967296
10.	Генератор Парка-Миллера	0-4294967296
11.	Генератор Геффа	0-255
12.	Аддитивный генератор	0-255
13.	Конгруэнтный генератор	0-255
14.	Генератор Парка-Миллера	0-65535
15.	Генератор Геффа	0-255
16.	Аддитивный генератор	0-4294967296
17.	Конгруэнтный генератор	0-4294967296
18.	Генератор Парка-Миллера	0-255
19.	Генератор Геффа	0-255
20.	Аддитивный генератор	0-65535
21.	Конгруэнтный генератор	0-4294967296
22.	Генератор Парка-Миллера	0-4294967296
23.	Генератор Геффа	0-255
24.	Аддитивный генератор	0-255
25.	Конгруэнтный генератор	0-65535
26.	Генератор Парка-Миллера	0-255
27.	Генератор Геффа	0-65535
28.	Аддитивный генератор	0-255
29.	Конгруэнтный генератор	0-65535
30.	Генератор Парка-Миллера	0-255

Примечание к выполнению лабораторной работы №1

Методы хеширования паролей

Хеширование – это преобразование, получающее из данных произвольной длины некое значение (свертку) фиксированной длины.

Рассмотрим самый простой пример хеширования пароля для конгруэнтного генератора, реализуемого формулой

$$I(n+1)=(a*I(n)+c)(mod\ m).$$

Инициализация генератора осуществляется тройкой чисел, которая устанавливает начальные значения и параметры генерации. Известно, что

генерируемая последовательность будет иметь больший период, если выполняются два условия:

1. A mod 4 = 1.
2. C – нечетное.

Для того чтобы установить привязку текстового пароля к начальным параметрам инициализации, необходимо разработать алгоритм хеширования этого пароля. При этом должны выполняться несколько условий:

- полученное значение хеш-функции должно однозначно описывать текстовый пароль, причем изменение хотя бы одного символа в пароле должно вызывать существенное изменение значения хеш-функции;
- результат хеширования должен удовлетворять условиям, позволяющим получить максимальный период генерации случайных чисел.

Предложим следующий вариант:

I_0 – длина пароля

A – сумма ASCII кодов пароля, вычисляемая по формуле: $S = \sum_{i=1}^{n} abs(pass[i] - pass[i+1])$, где n = I_0−1

C – сумма ASCII кодов гласных букв пароля.

Например, если пароль pass = «qwerty», то

I_0 = 6

A = 44

C = 222

Эта тройка чисел описывает текстовый пароль, но не удовлетворяет условиям успешной генерации чисел, поэтому увеличим C на единицу, то есть C =223 , а A также увеличим на 1, чтобы выполнялось условие A mod 4 = 1.

9. Блочные шифры

Примеры блочных шифров

В алгоритме *перестановки* в каждом блоке меняется последовательность некоторых подблоков внутри блока, например байт или бит в слове, причем порядок перестановок определяется ключом.

Пусть имеется некоторое исходное сообщение «M E S S A G E » , которое необходимо закодировать (Рисунок 2.14). Это сообщение имеет длину в 7 байт (если используется ASCII код). Разобьем этот блок текста на три подблока: «M E S », «S A » и «G E ». Числа *M* и *N*, которые определяют границы подблоков, получены при помощи генератора случайных чисел и зависят от конкретного ключа.

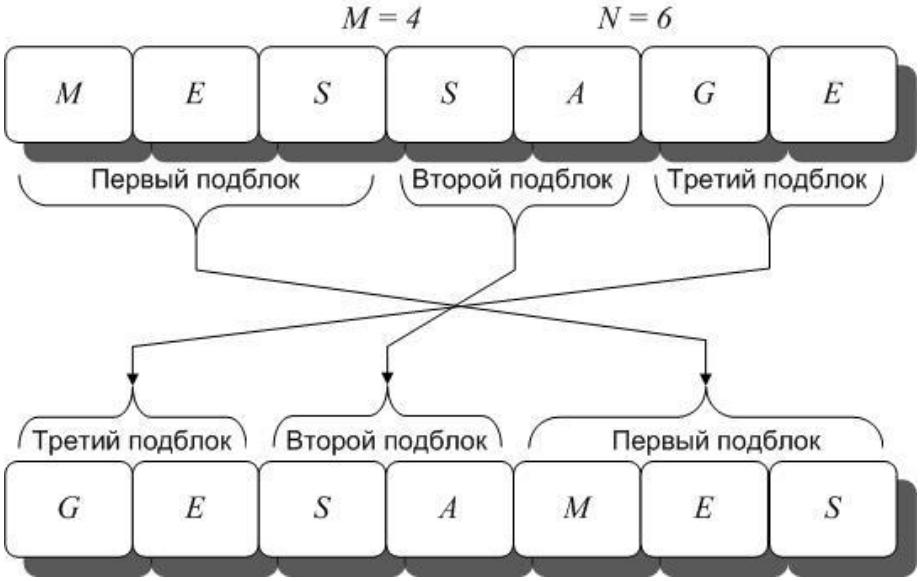


Рисунок 2.14. Пример перестановки

Для данного примера $M = 4$, $N = 6$.

После перестановки мы получим зашифрованное сообщение: «G E S A M E S » .

Дешифрование происходит по той же схеме, но в обратном порядке.

Механизм шифрования/дешифрования этого блочного шифра реализуется при помощи следующего алгоритма:

```
// Функция производящая перестановку в блоке
// str - исходный текст до перестановки
// n и m - границы подблоков в блоке
// return - текст после перестановки
// Правило формирования:
```

```
// 1. блок разбивается на три подблока, границами которых служат
числа m и n
// 2. Третий подблок записывается на место первого
// 3. Второй  - на место второго
// 4. Первый - на место третьего
function TForm1.cryptblock (str:block;n,m:integer):block;
var
  // Счетчики внутренних и внешнего циклов
  i,k:integer;
  // Текст после выполнения перестановки
  retstr :block;
begin
  k := 1;
  // Третий подблок записывается на место первого
  for i:=n to sblock do begin
    retstr[k] := str[i];
    k:=k+1;
  end;
  // Второй  - на место второго
  for i:=m to n-1 do begin
    retstr[k] := str[i];
    k:=k+1;
  end;
  // Первый - на место третьего
  for i:=1 to m-1 do begin
    retstr[k] := str[i];
    k:=k+1;
  end;
  cryptblock := retstr;
end;
```

Аналогично производится операция перестановки над группами бит.

Примечание: для блоков большой длины метод перестановок становится неэффективным, так как и длина подблоков в этом случае также велика. Поэтому шифрование должно осуществляться в несколько раундов, то есть перемешивание производится несколько раз над одним и тем же блоком, но с различными значениями коэффициентов M и N. При дешифровании необходимо воспроизвести эту последовательность псевдослучайных чисел в обратном порядке.

В алгоритме шифрования *методом сдвига (скремблера)* исходный текст разбивается на подблоки и внутри каждого такого подблока реализуется операция циклического сдвига на несколько бит в указанном направлении. Например, для блока длиной в 7 байт, где шифрование осуществляется циклическим сдвигом влево на 3 бита, схема шифрования будет следующей (Рисунок 2.15):

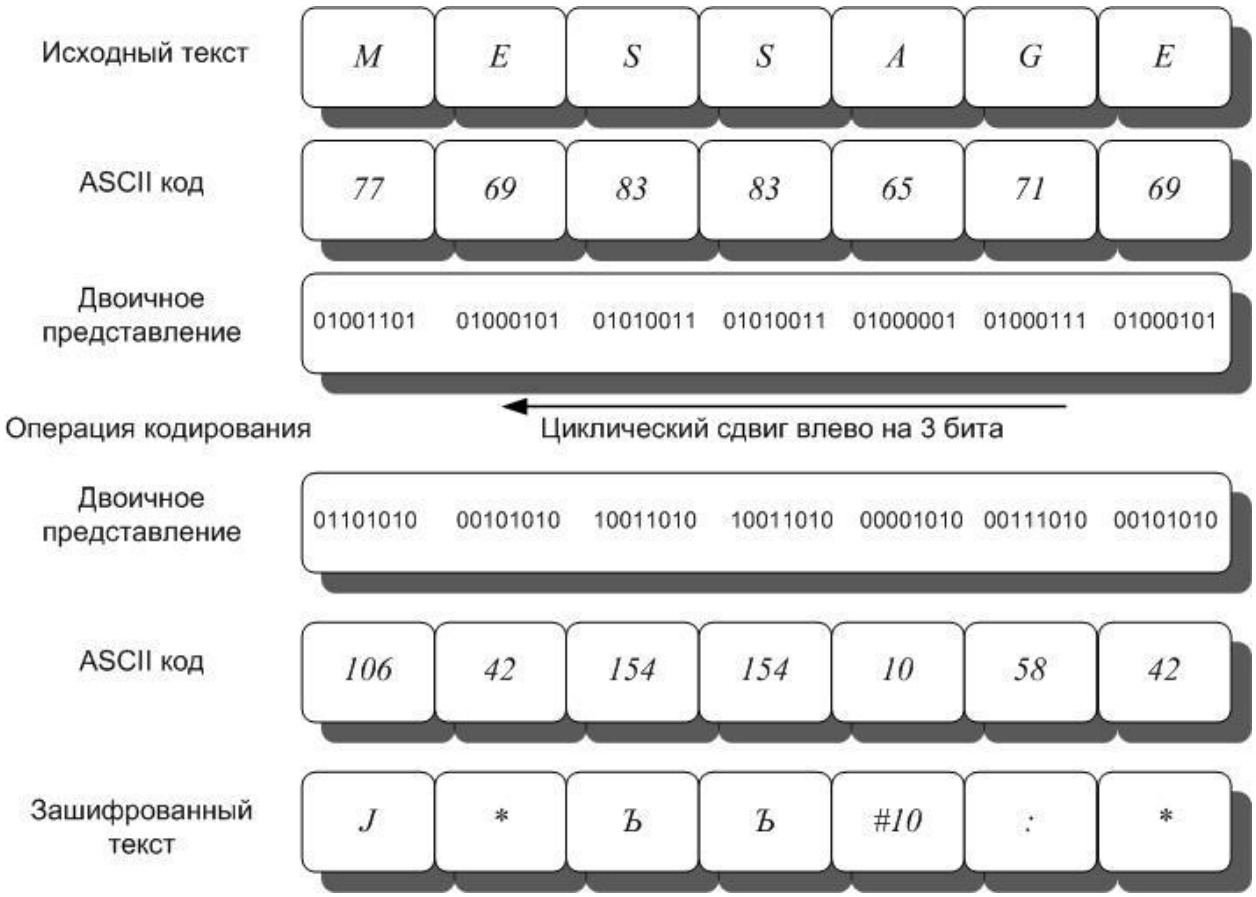


Рисунок 2.15. Пример шифрования методом циклического сдвига

Соответственно для раскодирования необходимо произвести циклический сдвиг подблока в противоположную сторону на такое же количество бит. Обычно величина сдвига и его направление определяется паролем, что обеспечивает привязку к ключу. В том случае, если блок шифруется по байтам, то следует исключать ситуации, при которых размер сдвига кратен 8.

Программная реализация такого шифра включает три функциональных блока, где каждый может быть оформлен как отдельная процедура. Это процедура «распаковки» (`procedure TForm1.unpack`) в битовое представление, процедура, организующая циклический сдвиг (`function TForm1.shiftblock`), и процедура «упаковки» битового представления обратно в текстовый блок (`procedure TForm1.pack`).

```
// Функция преобразования байтового блока в массив бит
// cblock - текущий кодируемый блок
// cbitblock - битовое представление cblock
// Правило преобразования: все байты исходного блока переводятся в
// битовое представление (8 -> 2) и поочередно записываются в
массив бит
// Например, если исходный блок - "AB" (#65#66), то массив бит
будет таким:
```

```
// 01000001 1000010
procedure TForm1.unpack(cblock:block; var cbitblock:bitblock);
var
  // Счетчики циклов
  i,j:integer;
begin
  // Перебор байтов в блоке
  for i:=1 to sblock do begin
    // Перевод в 2-ю систему счисления
    for j:=8 downto 1 do begin
      // берем остаток от деления на 2 от текущего байта
      cbitblock[(i-1)*8+j]:=ord(cblock[i]) mod 2;
      // производим целочисленное деление текущего байта на 2
      cblock[i]:=chr(ord(cblock[i]) div 2);
    end;
  end;
end;

// Функция преобразования массива бит в байтовый блок
// cblock - текущий кодируемый блок
// cbitblock - битовое представление cblock
//   Правило преобразования: битовое представление массива
свертывается
// в байтовый блок, то есть производится действие обратное функции
pack
procedure TForm1.pack(var cblock:block; cbitblock:bitblock);
var
  // Счетчики циклов
  i,j:integer;
begin
  // Поочередно формируем байты блока
  for i:=1 to sblock do begin
    cblock[i] := #0;
    for j:=1 to 8 do
      // "Свертываем" битовое представление в байт
      cblock[i] := chr((ord(cblock[i]) shl 1) or cbitblock[(i-
1)*8+j]);
  end;
end;

// Функция, производящая сдвиг массива бит на n разрядов в
указанном направлении
// cbitblock - исходный массив бит
// n - размер сдвига
// shift - направление сдвига: sright - сдвиг вправо, sleft -
сдвиг влево
// return - массив бит, сдвинутый на n разрядов в указанном
направлении
// Правило преобразования: массив бит делится на две части, где n
- граница
// и происходит перестановка полученных подблоков местами
```

```
function      TForm1.shiftblock(cbitblock:bitblock;      n:integer;
shift:tshift):bitblock;
var
  // Счетчики внутреннего и внешнего циклов
  i,k:integer;
  // Массив для хранения текущего получаемого значения
  cshiftblock:bitblock;
begin
  k:=1;
  // сдвиг вправо
  if shift = sright then begin
    for i:=sblock*8-n+1 to sblock*8 do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
    for i:=1 to sblock*8-n do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
  end else
  // сдвиг влево
  begin
    for i:=n+1 to sblock*8 do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
    for i:=1 to n do begin
      cshiftblock[k]:= cbitblock[i];
      k:=k+1;
    end;
  end;
  shiftblock := cshiftblock;
end;
```

К недостаткам вышеперечисленных методов блочного шифрования можно отнести их слабую криптостойкость, так как одинаковые блоки исходного текста выглядят одинаково в зашифрованном виде. Это позволяет, проведя несколько экспериментов, выяснить принцип формирования шифротекста и расшифровать его. Кроме того, такие шифры легко поддаются анализу на относительные частоты.

Следующий метод – *метод замены по таблице*, использует специальную функцию «потопления статистики», которая значительно затрудняет взлом. В табличном методе при шифровании одному символу может быть поставлено в соответствие одно из нескольких значений, которое выбирается абсолютно случайно и не зависит от ключа. Это означает, что даже используя один и тот же ключ для шифрования одинакового текста, каждый раз мы будем получать разные шифры.

Для этого необходимо сформировать таблицу значений для подстановки. Размерность этой таблицы зависит от двух параметров алгоритма: длины шифруемого блока (n) и некоторой постоянной величины, которая определяет число возможных значений подстановки для шифрования одного и того же символа. Обозначим эти параметры как *rows* – число строк в таблице и *columns* – число столбцов. Число строк определяется как $rows = 2^n$, где n – это длина шифруемого блока в битах.

Полученная таблица размерностью $rows \times columns$ заполняется псевдослучайными значениями от 0 до $rows \times columns$. При этом обязательным требованием должно быть требование уникальности этих значений, то есть они не должны повторяться.

Этого можно добиться следующим образом:

```
// Процедура генерации таблицы значений для подстановки
// tab - таблица для подстановочных значений
// Правило формирования:
// Шаг 1: Таблица заполняется последовательными значениями от 0 до
columns*rows
// Шаг 2: Значения в таблице перемешиваются counthash раз в
зависимости от значений, выдаваемых собственным конгруэнтным
генератором, который инициализируется hash значением
procedure TForm1.genTable(var tab:ttab);
var
   // Внутренний и внешний счетчики циклов
  i,j:integer;
  // Массив для хранения 2-х координат таблицы.
  // Значения с такими координатами меняются местами
  rr:trr;
  // Предыдущее значение выданное конгруэнтным генератором
  prev:cardinal;
   // Буфер для организации обмена двумя числами в таблице
  temp:integer;
begin
  // Начальное заполнение таблицы последовательными значениями
  for i:=0 to rows-1 do
    for j:=0 to columns-1 do
      tab[i,j] := i*columns+j;
  //  Извлекаем первое случайное значение из конгруэнтного
генератора
  prev := genCongr(hash(PasswordDlg.Password.Text));
  // Организуем цикл перемешивания таблицы в counthash шагов
  for j := 0 to counthash do begin
    // Заполняем массив rr четырьмя случайными значениями
    for i:=0 to 3 do
      if i mod 2 =0 then begin
        prev:=genCongr(prev);
        rr[i] := prev mod rows;
      end
```

```
      else begin
        prev := genCongr(prev);
        rr[i]:=prev mod columns;
      end;
    // Меняем элементы местами в зависимости от полученных
координат
    temp := tab[rr[0]][rr[1]];
    tab[rr[0]][rr[1]] := tab[rr[2]][rr[3]];
    tab[rr[2]][rr[3]] := temp;
  end;
end;
```

В приведенном алгоритме используется прием перемешивания значений таблицы в зависимости от пароля. То есть изначально таблица заполняется последовательными значениями, а затем происходит перетасовка элементов. В результате получается таблица, заполненная псевдослучайными значениями, которая и называется таблицей подстановки.

Допустим, что размер группы *4* бита, а число столбцов равно *3*. Значит, таблица будет иметь размерность $2^4 \times 3 = 16 \times 3$ и должна заполняться значениями в диапазоне от *0* до *48*.

Например, в нашем случае может получиться такая таблица:

	1	*2*	*3*
0	*15*	*10*	*2*
1	*3*	*17*	*9*
2	*8*	*14*	*1*
3	*0*	*16*	*45*
4	*...*	*...*	*...*
7	*20*	*5*	*38*
8	*...*	*...*	*...*
15	*12*	*30*	*4*

При шифровании исходный блок представляется в виде числового эквивалента и в таблице отыскивается строка, номер которой равен этому эквиваленту. Далее в этой строке абсолютно случайно выбирается одно из нескольких значений.

Например, был прочитан блок, числовой эквивалент которого равен *7*. По таблице в строке с номером *7* находится тройка значений (*20, 5, 38*). При помощи функции *random(3)* выбирается любое значений из этой тройки. Это значение и есть зашифрованный блок.

На языке Object Pascal такой алгоритм можно представить следующим образом:

```
// Функция шифрования
// cblock - текущий блок, подлежащий шифрованию
// return - зашифрованный блок двойного размера
// Правило формирования:
```

```
// Шаг 1: Текущий кодируемый блок переводится в число
// Шаг 2: Производится замена по таблице
// Шаг 3: Значение из таблицы переводится в строковое
представление
function TForm1.encryptblock(cblock:block):dblock;
var
  i,                  // Счетчик цикла
  value:integer;       // Текущий числовой эквивалент кодируемого
значения
  retblock: dblock; // Возвращаемый зашифрованный блок
begin
  value := 0;
  // Переводим кодируемый блок в числовой эквивалент
  for i:=1 to sblock do
    value := (value shl 8) or ord(cblock[i]);
  // Берем значение из таблицы
  value := tab[value][random(columns)];
  // Переводим полученное числовое значение в текстовый эквивалент
  for i:=1 to sblock*2 do begin
    retblock[i]:= chr(value and $00ff);
    value := value shr 8;
  end;
  encryptblock := retblock;
end;
```

При дешифровании сначала воспроизводится таблица подстановки, так как она зависит от ключа. Затем читается закодированный блок и по таблице отыскивается строка, в которой присутствует это значение. Например, если был прочитан зашифрованный блок, числовой эквивалент которого равен *16*, то в таблице такой значение находится в строке с номером *3*. Это и есть требуемое значение. Номер строки преобразуется в строковое представление и записывается в выходной поток.

Ниже приведен соответствующий алгоритм:

```
// Функция дешифрования
// cblock - текущий блок двойного размера, подлежащий дешифрованию
// return - расшифрованный блок одинарного размера
// Шаг 1: Текущий декодируемый блок переводится в число
// Шаг 2: Производится поиск значения в таблице
// Шаг 3: Номер строки таблицы переводится в строковое
представление
function TForm1.decryptblock(cblock:dblock):block;
var
  i,j,        // Счетчики внешнего и внутреннего циклов
  value,      // Текущий числовой эквивалент декодируемого блока
  temp:integer;   // Временная переменная
  retblock:block; // Возвращаемый декодирвоанный блок
begin
  value := 0;
```

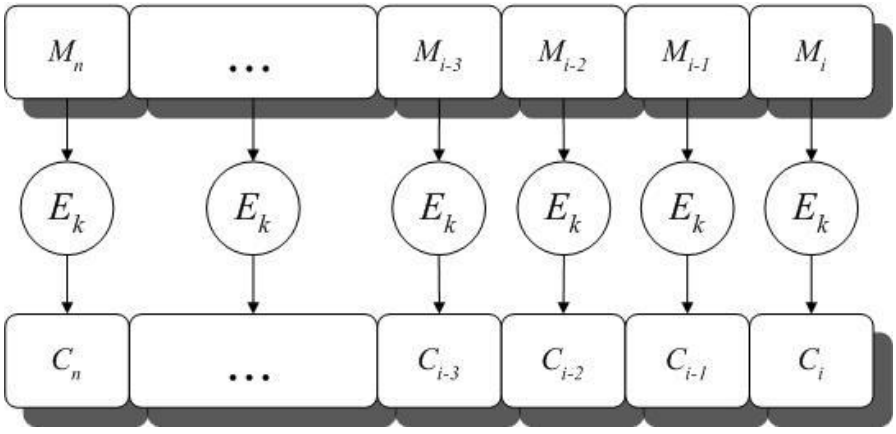
```
  // Переводим блок в число
  for i:=sblock*2 downto 1 do
    value := (value shl 8) or ord(cblock[i]);
  j:=0;
  i:=0;
  // Ищем значение в таблице
  while temp <> value do begin
    if j div columns-1 = 0 then begin
      i:=i+1;
      j:=0;
    end;
    temp := tab[i][j];
    j := j+1;
  end;
  // Переводим полученное значение в стоковое представление
  for j:=1 to sblock do begin
    retblock[sblock-j+1]:=chr(i and $ff);
    i := i shr 8;
  end;
  decryptblock := retblock;
end;
```

Как было сказано раньше, такой алгоритм обладает высокой криптостойкостью за счет абсолютной случайности выбора при шифровании. Однако он имеет существенный недостаток – это увеличение объема шифруемого текста, что не является хорошим показателем.

Режимы использования блочных шифров

Для шифрования исходного текста произвольной длины блочные шифры могут быть использованы в нескольких режимах. Чаще всего используются четыре основных режима. Это режим *электронной кодировочной книги* (ECB – Electronic Code Book), *сцепления блоков шифрованного текста* (CBC – Cipher Block Chaining), *обратной связи по шифрованному тексту* (CFB – Cipher Feedback) и *обратной связи по выходу* (OFB – Output Feedback).

В режиме *электронной кодировочной книги (ECB)* каждый блок



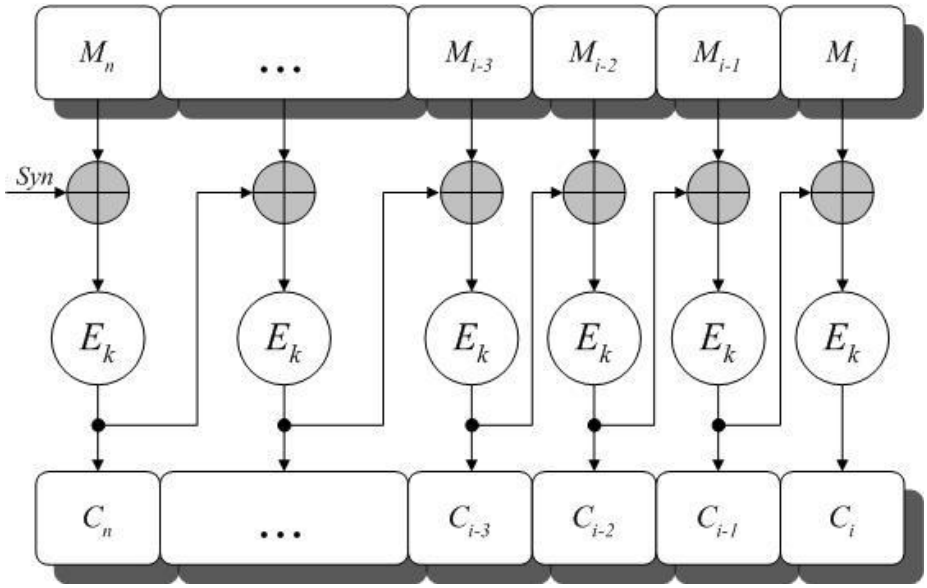
$$C_i = E_k(M_i); \quad M_i = D_k(C_i)$$

Рисунок 2.16. Режим электронной кодировочной книги (ECB)

исходного текста шифруется блочным шифром независимо от других (Рисунок 2.16).

Стойкость режима ECB равна стойкости самого шифра, однако структура шифрованного текста при этом не скрывается, так как каждый одинаковый блок исходного текста приводит к появлению одинакового блока шифрованного текста. Режим ECB допускает простое распараллеливание для увеличения скорости шифрования. Режим ECB соответствует режиму простой замены алгоритма ГОСТ 28147-89.

В режиме *сцепления блоков шифрованного текста (CBC)* каждый блок исходного текста складывается поразрядно по модулю два с предыдущим блоком шифрованного текста, а затем шифруется (Рисунок 2.17). Для начала процесса шифрования используется *синхропосылка* (или начальный вектор), которая передается в канал связи в открытом виде.



$$C_i = E_k(M_i \oplus C_{i-1});\ M_i = D_k(C_i) \oplus C_{i-1}$$

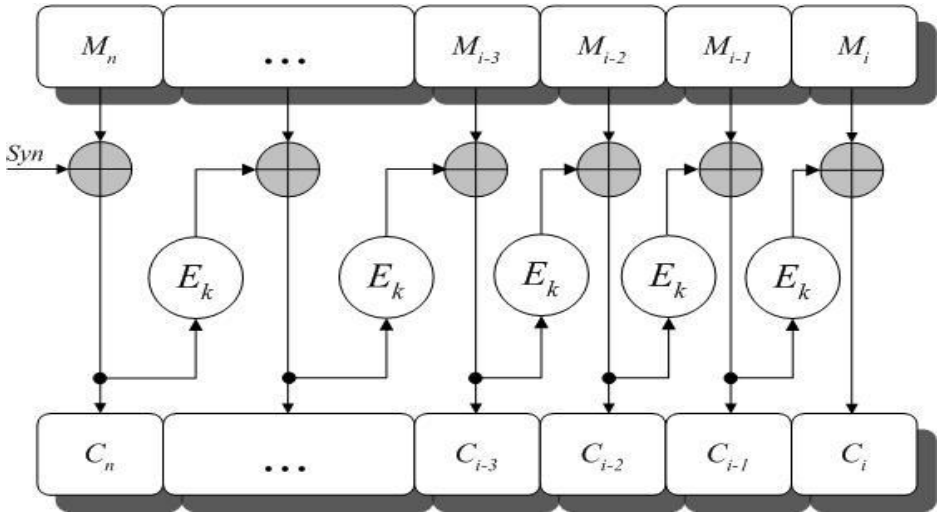
Рисунок 2.17. Режим сцепления блоков шифрованного текста (CBC)

Стойкость режима CBC равна стойкости блочного шифра, лежащего в его основе. Кроме того, структура исходного текста скрывается за счет сложения предыдущего блока шифрованного текста с очередным блоком открытого текста. Стойкость шифрованного текста увеличивается, поскольку становится невозможной простая манипуляция исходным текстом, кроме как путем удаления блоков из начала или конца шифрованного текста. Скорость шифрования равна скорости работы блочного шифра, но простого способа распараллеливания процесса шифрования не существует, хотя дешифрование может проводиться параллельно.

В режиме *обратной связи по шифрованному (CFB)* тексту предыдущий блок шифрованного текста шифруется еще раз, и для получения очередного блока шифрованного текста результат складывается поразрядно по модулю 2 с

блоком исходного текста (Рисунок 2.18). Для начала процесса шифрования также используется начальный вектор.

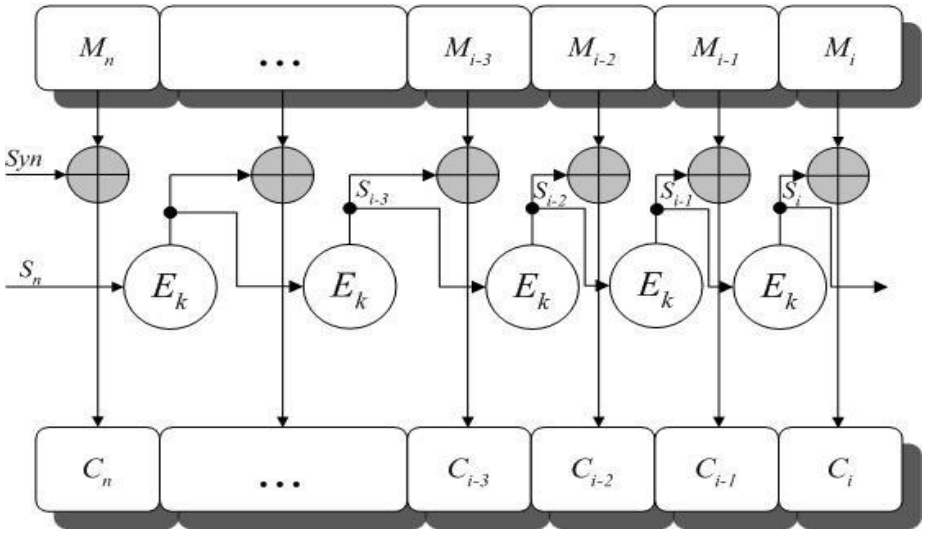
Стойкость режима CFB равна стойкости блочного шифра, лежащего в основе и структура исходного текста скрывается за счет использования операции сложения по модулю 2. Манипулирование исходным текстом путем удаления блоков из начала или конца шифрованного текста становится невозможным. Однако в режиме CFB при шифровании двух идентичных блоков на следующем шаге шифрования также получатся идентичные зашифрованные блоки, что создает возможность утечки информации об исходном тексте.



$$C_i = M_i \oplus E_k(C_{i-1}); \quad M_i = C_i \oplus D_k(C_{i-1})$$

Рисунок 2.18. Режим обратной связи по шифрованному тексту (CFB)

Скорость шифрования равна скорости работы блочного шифра и простого способа распараллеливания процесса шифрования также не



$$C_i = M_i \oplus S_i; \quad M_i = C_i \oplus S_i; \quad S_i = E_k(S_{i-1})$$

Рисунок 2.19. Режим обратной связи по выходу (OFB)

существует. Этот режим в точности соответствует режиму гаммирования с обратной связью алгоритма ГОСТ 28147–89.

Режим *обратной связи по выходу (OFB)* подобен режиму CFB за исключением того, что величины, складываемые по модулю 2 с блоками исходного текста, генерируются независимо от исходного или шифрованного текста (Рисунок 2.19). Для начала процесса шифрования также используется начальный вектор. Режим OFB обладает преимуществом перед режимом CFB в том смысле, что любые битовые ошибки, возникшие в процессе передачи, не влияют на дешифрование последующих блоков. Однако возможна простая манипуляция исходным текстом путем изменения шифрованного текста.

В некоторых рассмотренных режимах шифрование блоков зависит от шифрования предыдущего блока. Например, представим аппаратное устройство шифрования, работающее в режиме CBC. Даже если там будут три микросхемы, осуществляющие шифрование, только одна сможет работать в данный момент времени, так как следующей микросхеме понадобится результат работы предыдущей. Решение заключается в *перемежении* нескольких потоков шифрования. Вместо одного потока в режиме CBC можно использовать четыре. Первый, пятый и далее каждый четвертый блоки будут шифроваться на одной микросхеме с одной синхропосылкой. Второй, шестой и далее каждый четвертый блоки будут обрабатываться второй микросхемой со второй синхропосылкой и т. д. Таким образом, например, имея три микросхемы, осуществляющие скорость шифрования 33 Мбит/с, можно шифровать трафик в канале со скоростью 100 Мбит/с.

Объединение блочных шифров

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

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

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

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

$$C = E_{k2}(E_{k1}(M)); \qquad M = D_{k1}(D_{k2}(C)).$$

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

$$C = E_{k1}(D_{k2}(E_{k1}(M))); \qquad M = D_{k1}(E_{k2}(D_{k1}(C))).$$

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

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

$$C = E_{k1}(D_{k2}(E_{k3}(M))); \qquad M = D_{k1}(E_{k2}(D_{k3}(C))).$$

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

1. Генерируется строка случайных битов R того же размера, что и сообщение M.
2. R шифруется первым алгоритмом.
3. $M \oplus R$ шифруется вторым алгоритмом.
4. Шифротекст сообщения является объединением результатов этапов 2 и 3.

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

Сеть Фейстела

Одним из наиболее распространенных способов задания блочных шифров является использование так называемых сетей Фейстела [10–14]. Сеть Фейстела представляет собой общий метод преобразования произвольной функции (обычно называемой *F*-функцией) в перестановку на множестве блоков.

Пусть X – блок текста (длина блока текста обязательно должна быть четной). Представим его в виде двух подблоков одинаковой длины $X = \{A,B\}$. Тогда одна итерация (или *раунд*) сети Фейстела определяется как

$$X_{i+1} = B_i \,||\, (F(B_i, K_i) \oplus A_i),$$

где $X_{i+1} = \{A_{i-1}, B_{i-1}\}$, $\|$ – операция конкатенации, а $\oplus$ – побитовое

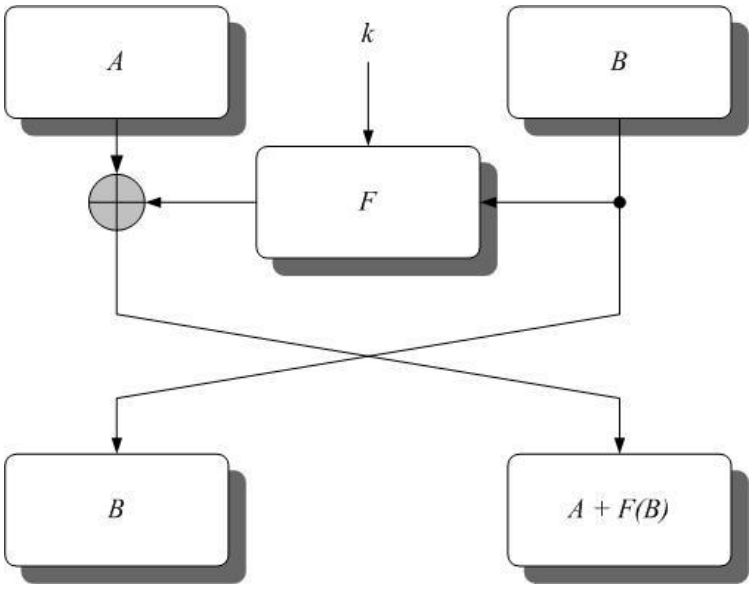


Рисунок 2.20. Структура итерации сети Фейстела.

исключающее ИЛИ (Рисунок 2.20). Сеть Фейстела состоит из некоторого фиксированного числа итераций, определяемого соображениями стойкости разрабатываемого шифра, при этом на последней итерации перестановка местами половин блоков текста не производится, так как это не влияет на стойкость шифра.

Данная структура шифров обладает рядом достоинств, а именно:

- Процедуры шифрования и дешифрования совпадают, с тем исключением, что при дешифровании ключевая информация используется в обратном порядке.
- Для построения устройств шифрования можно использовать те же блоки в цепях шифрования и дешифрования.

Недостатком является то, что на каждой итерации изменяется только половина обрабатываемого текста, что приводит к необходимости увеличивать число итераций для достижения требуемой стойкости.
В отношении выбора *F*-функции определенных стандартов не существует, однако, как правило, эта функция представляет собой последовательность зависящих от ключа нелинейных замен, перемешивающих перестановок и сдвигов.

Пример реализации блокового шифра приведен в лабораторной работе №2 «Блочные шифры».

10. Лабораторная работа №2 «Блочные шифры»

Задание:

1. Изучить блочные алгоритмы шифрования: алгоритм перестановки, алгоритм скремблеров, алгоритм замены по таблице, матричный метод преобразования и алгоритм Винжера.
2. Изучить режимы использования блочных шифров (ECB, CBC, CFB и OFB).
3. Изучить способы объединения блочных шифров (многократное шифрование, сеть Фейстела).
4. Реализовать систему в соответствии с вариантами, указанными в таблице 2 и заданием:

   Изучить принцип работы алгоритма, который указан в варианте (Таблица 2, Поле А) на приведенном примере. Разработать собственный алгоритм (или модифицировать пример), который реализует указанный в варианте:

   - *режим использования блочного шифра (Таблица 2, Поле В);*
   - *работает с указанной длиной блока (Таблица 2, Поле С);*
   - *позволяет оценивать скорость шифрования/дешифрования.*

Дополнительные требования:

1. Функции шифрования/дешифрования + вспомогательные функции, необходимые для осуществления процесса кодирования/декодирования, помещаются в **отдельную** библиотеку dll (это необходимо для выполнения третьей лабораторной работы).
2. Пароль в зашифрованном виде записывается в закодированный файл. При попытке расшифровать его с другим паролем выводится сообщение об ошибке.
3. В процессе кодирования осуществляется **подсчет контрольной суммы.** При декодировании осуществляется проверка контрольной суммы. При несовпадении выдается сообщение об ошибке. Для четных вариантов контрольная сумма считается по открытому тексту, для нечетных вариантов по шифрованному тексту.
4. Должна иметься возможность отключения режима использования блочного шифра и работа программы в режиме простого ECB.
5. Паролем может быть ЛЮБАЯ последовательность символов (русских и английских, цифр, знаков препинания и т. д.).
6. Программа должна быть оформлена в виде удобной утилиты, позволяющей работать с любыми файлами.
7. Программа должна обеспечивать шифрование файлов произвольной длины.

8. Текст программы оформляется согласно общепринятым правилам (удобочитаемо, с описанием ВСЕХ функций, переменных и критических мест).
9. В процессе работы программа ОБЯЗАТЕЛЬНО выдает информацию о состоянии процесса кодирования/декодирования.
10. После завершения работы программы выдает информацию о скорости шифрования / дешифрования (символ /сек)
11. Интерфейс программы может быть произвольным, но удобным и понятным (разрешается использование библиотек VCL)
12. Среда разработки и язык программирования могут быть произвольными.

Требования для сдачи лабораторной работы:

5. Демонстрация работы реализованной вами системы.
6. АВТОРСТВО.
7. Теория (ориентирование по алгоритму и теоретическим аспектам методов гаммирования и перестановок).
8. Оформление и представление письменного отчета по лабораторной работе, который содержит:
   - Титульный лист
   - Задание на лабораторную работу
   - Описание используемых алгоритмов шифрования
   - Листинг программы

Таблица 2. Варианты заданий

№ варианта	A	B	C	D
1.	Перестановка	Фейстела	10 байт	Число подблоков 3
2.	Скремблер	OFB	3 байта	Сдвиг влево
3.	Замена по таблице	CBC	1 байт	Число столбцов 5
4.	Матричный метод	CFB	7 байт	-
5.	Система Винжера	Фейстела	1 байт	-
6.	Перестановка	OFB	11 байт	Число подблоков 4
7.	Скремблер	CFB	7 байт	Сдвиг вправо
8.	Замена по таблице	CBC	1 байт	Число столбцов 4
9.	Матричный метод	CBC	9 байт	-
10.	Система Винжера	OFB	1 байт	-
11.	Перестановка	CFB	12 байт	Число подблоков 3
12.	Скремблер	Фейстела	6 байт	Сдвиг влево
13.	Замена по таблице	Фейстела	2 байта	Число столбцов 6
14.	Матричный метод	OFB	11 байт	-
15.	Система Винжера	CBC	1 байт	-
16.	Перестановка	CBC	15 байт	Число подблоков 4
17.	Скремблер	CBC	7 байт	Сдвиг вправо
18.	Замена по таблице	CFB	2 байта	Число столбцов 7
19.	Матричный метод	Фейстела	8 байт	-
20.	Система Винжера	CFB	1 байт	-

11. Системы шифрования с открытым ключом

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

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

Механизм распространения открытых ключей

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

1. Разложение произведения больших простых чисел на сомножители.
2. Вычисление логарифма в конечном поле.
3. Вычисление корней алгебраических уравнений.

Такие системы могут быть использованы по следующим назначениям:

1. Как самостоятельные средства защиты передаваемых и хранимых данных.
2. Как средства для распределения ключей.
3. Как средства аутентификации пользователей.

Распространение ключей происходит так (Рисунок 2.2221):

1. Получатель сообщения производит вычисление пары ключей $K_ш$ и $K_д$. Ключ $K_д$ считается закрытым, $K_ш$ – открытым.
2. Закрытый ключ $K_д$ получатель оставляет у себя, а открытый ключ $K_ш$ пересылает отправителю.
3. Пользуясь открытым ключом $K_ш$ любой абонент может зашифровать текст и отослать его получателю. Расшифровать зашифрованное сообщение может только тот, у кого есть закрытый ключ, а так как он

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

4. Пользуясь закрытым ключом $К_д$, получатель расшифровывает полученное сообщение.

Возможна и другая схема распространения ключей (Рисунок 2.22):

1. Отправитель сообщения производит вычисление пары ключей $К_ш$ и $К_д$. Ключ $К_ш$ считается закрытым, $К_д$ – открытым.
2. Закрытый ключ $К_ш$ отправитель оставляет у себя, а открытый ключ $К_д$ пересылает получателю.
3. Пользуясь закрытым ключом $К_ш$, отправитель производит шифрование сообщения и отправляет зашифрованный текст получателю.
4. Пользуясь открытым ключом $К_д$, любой абонент может расшифровать зашифрованный текст. Причем ключ дешифрования $К_д$ может расшифровать только то сообщение, которое было зашифровано закрытым ключом $К_ш$.

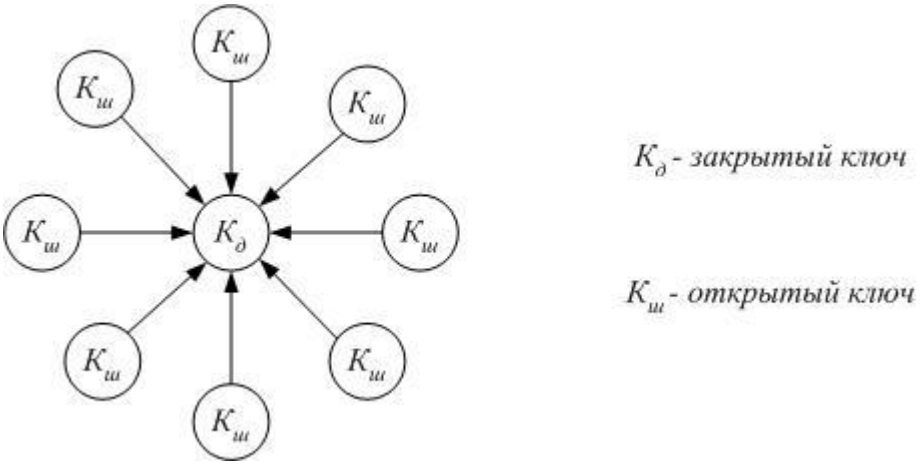


Рисунок 2.21. Первая схема распространения открытых ключей

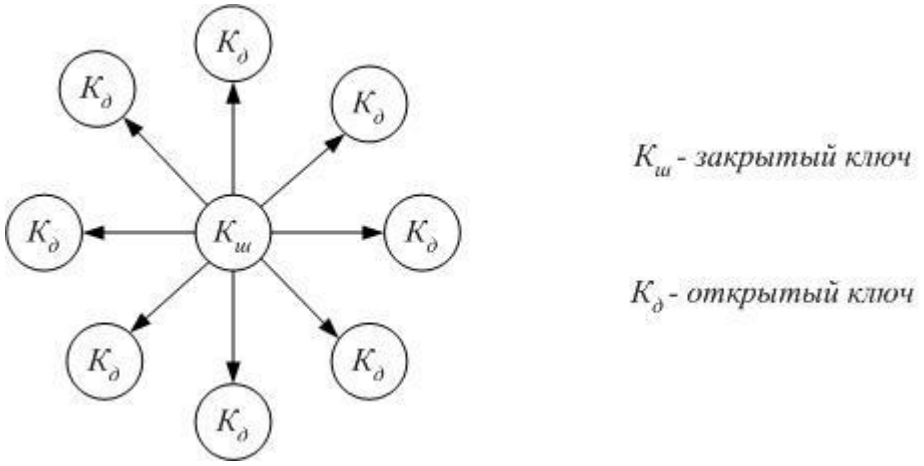


Рисунок 2.22. Вторая схема распространения открытых ключей

Механизм кодирования и декодирования по алгоритму RSA

RSA – аббревиатура от сокращения фамилий 3-х ученых, разработавших эту систему – Рональда Райвеста, Ади Шамира и Леонарда Адельмана. Алгоритм был предложен в 1977 году. Он основывается на том, что нахождение больших простых чисел осуществляется сравнительно легко, но разложение на множители произведения двух таких чисел требует значительных вычислительных затрат.

Простым числом называется число из натурального ряда, большее 1, которое делится без остатка только на 1 и само на себя.

Аксиома. Число простых чисел бесконечно.

Число *x* называется *простым относительно y*, если его нельзя разложить на сомножители, на которые число *y* не делится без остатка. Например число 4 является простым относительно 15, а число 6 не является простым относительно числа 15.

Числа *x* и *y* называются *взаимно простыми*, если наименьший общий делитель *НОД (x,y) = 1*.

Малая теорема Ферма. Если P – простое число, то $x^{p-1} = 1 \pmod p$ для любого *x*, простого относительно P.

Следовательно:

$$x^{p-1} = 1 \pmod p,$$

$$x^p \cdot x^{-1} = 1 \pmod p,$$

$$\frac{x^p}{x} = 1 \pmod p \quad \Big| \times x.$$

$$x^p = x \pmod p$$

Функцией Эйлера (n) называется число положительных целых меньших *n* и простых относительно *n*, на которые *n* не делится без остатка.

N	2	3	4	5	6	7	8	9	10	11	12
(n)	1	2	2	4	2	6	4	6	4	10	4
СПИСОК	1	1 2	1 3	1 2 3 4	1 5	1 2 3 4 5 6	1 3 5 7	1 2 4 5 7 8	1 3 7 9	1 2 3 4 5 6 7 8 9 10	1 5 7 11

Теорема 2. Если $n = p \cdot q$, где p и q простые и не равны друг другу, то $(n) = (p-1)\cdot(q-1)$.

Например, 3*5 = 15, (15) = 8, (3)=2, (5) = 4.

Теорема 3. Если $n = p \cdot q$, где p и q простые и не равны друг другу, а x – простое относительно p и q, то $x \cdot (n) = 1 \pmod{p}$.

Из этого следует, что если *e* простое относительно *n*, то легко можно подобрать целое число *d*, такое, что $x \cdot d = 1 (\mathrm{mod}(n))$.

Алгоритм генерации ключей.

1. Отправитель выбирает два очень больших простых числа *P* и *Q* и вычисляет два произведения *N=P*Q* и *M=(P-1)*(Q-1).*
2. Затем он выбирает случайное число *D*, взаимно простое с *M*, и вычисляет *E*, удовлетворяющее условию *D*E=1 mod M.*
3. После этого он публикует *D* и *N* как свой открытый ключ шифрования, сохраняя *E* (в паре с *N*) как закрытый ключ.
4. Теперь, чтобы зашифровать данные по известному ключу *{D,N}*, необходимо сделать следующее:
   a. разбить шифруемый текст на блоки, каждый из которых может быть представлен в виде числа *S(i)=0,1, ... ,N-1*;
   b. зашифровать текст, рассматриваемый как последовательность чисел *S(i)* по формуле $S'(i)=(S(i)^D) \bmod N$.
5. Чтобы расшифровать эти данные, используют секретный ключ *{E,N}*, необходимо выполнить следующие вычисления:
   $S''(i)=S(i)=(S'(i)^E) \bmod N$.
6. В результате будет получено множество чисел *S''(i)*, которые представляют собой исходный текст.

12. Криптосистема Эль-Гамаля

Данная система является альтернативой алгоритму RSA и при равном значении ключа обеспечивает ту же криптостойкость. Метод Эль-Гамаля основан на проблеме дискретного логарифма. Если возводить число в степень в конечном поле достаточно легко, то восстановить аргумент по значению (то есть найти логарифм) довольно трудно.

Для генерации пары ключей сначала выбирается простое число *p* и два случайных числа, *g* и *x*, оба меньше *p*. Затем вычисляется

$y = g^k \bmod p$.

Открытым ключом являются *y*, *g* и *p*. И *g*, и *p* можно сделать общими для группы пользователей. Закрытым ключом является *x*.

Для шифрования сообщения *M* сначала выбирается случайное число *k*, взаимно простое с *p* - 1. Затем вычисляются

$a = g^k \bmod p$,

$b = y^k M \bmod p$.

Пара (a,b) является шифротекстом. Получаемый шифротекст в два раза длиннее открытого текста. Для дешифрирования (a,b) вычисляется

$M = b/a^x \bmod p$.

Так как $a^x \equiv g^{kx} \pmod p$ и $b/a^x \equiv y^k M/a^x \equiv g^{xk} M/ g^{kx} = M \pmod p$, то все работает. Или иначе:

$a^x \bmod p = y^k \bmod p \rightarrow (g^k \bmod p)^x \bmod p = (g^x \bmod p)^k \bmod p$.

Каждая подпись или шифрование EIGamal требует нового значения k, и это значение должно быть выбрано случайным образом. Если когда-нибудь Злоумышленник раскроет k, он сможет раскрыть закрытый ключ x. Если Злоумышленник когда-нибудь сможет получить два сообщения, подписанные или зашифрованные с помощью одного и того же k, то он сможет раскрыть x, даже не зная значение k.

Открытый ключ:

p простое число (может быть общим для группы пользователей)

$g < p$ (может быть общим для группы пользователей)

$y = g^x \bmod p$.

Закрытый ключ:

$x < p$.

Шифрование:

k выбирается случайным образом, взаимно простое с p–1

a (шифротекст) $= g^k \bmod p$,

b (шифротекст)$= y^k M \bmod p$.

Дешифрирование:

M (открытый текст) $= b/a^x \bmod p$.

Другой – более практичный вариант криптосистемы Эль-Гамаля:

$y = g^x \bmod p$,

$a = g^k \bmod p$,

$b = M \oplus (y^k \bmod p)$,

$M = (a^x \bmod p) \oplus b$,

где $\oplus$ операция сложения по модулю 2.

13. Криптографические протоколы

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

Условные обозначения участников протоколов:
Алиса – Первый участник всех протоколов
Боб – Второй участник всех протоколов
Мэллори – Взломщик протоколов
Трент – Заслуживающий доверия посредник.

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

Протокол обмена сеансовыми ключами, использующий криптографию с открытыми ключами

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

Вскрытие "человек-в-середине"

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

Протокол «держась за руки»

Протокол "держась за руки", позволяет избежать вскрытия «человек-в-середине». Вот как он работает:
1. Алиса посылает Бобу свой открытый ключ.
2. Боб посылает Алисе свой открытый ключ.

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

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

Примеры разбиения на части:

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

Сертификация ключей с помощью цифровых подписей

Другим решением, позволяющим снизить угрозу атаки «человек-в-середине» является использование цифровой сертификации открытых ключей. В качестве лица, выдающего сертификаты, обычно выступает один из сертификационных центров, которому доверяют оба участника обмена. Центр сертификации подписывает открытые ключи Алисы и Боба, тем самым удостоверяя их подлинность. Обменявшись ключами, и Алиса, и Боб проверяют подпись сертификационного центра, затем выполняется протокол обмена ключами.

Цифровой сертификат состоит из:

1. Открытого ключа.
2. Информации о владельце ключа: имя, адрес, ИНН, псевдоним.
3. Цифровых подписей.

Цифровой сертификат может иметь несколько подписей, каждая из которых гарантирует подлинность определенной части информации о владельце и сертифицируется различными службами

Разделение секрета.

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

Например Трент желает разделить сообщение между Алисой и Бобом:

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

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

$M \oplus R \oplus S \oplus T = U.$

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

$R \oplus S \oplus T \oplus U = M.$

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

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

Метки времени

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

1. Алиса вычисляет значение однонаправленной хэш-функции для документа.
2. Алиса передает это значение Тренту.
3. Трент добавляет время и дату получения этого значения и затем подписывает результат цифровой подписью.

4. Трент отправляет подписанное значение хэш-функции вместе с меткой времени Алисе.

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

Пример протокола защиты базы данных

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

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

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

14. Лабораторная работа №3 «Использование систем шифрования с открытым ключом»

Задание:

1. Изучить теоретические основы построения систем с открытым ключом (СОК) и схемы распределения открытых ключей.
2. Изучить алгоритмы оптимизации наиболее сложных вычислительных аспектов СОК (тесты Рабина-Миллера и Лемана, алгоритм Евклида, расширенный алгоритм Евклида, алгоритмы ускоренного умножения в конечном поле).
3. Реализовать следующие функциональные блоки:
    a. **Генератор ключей** – модуль, предназначенный для генерации пары ключей (открытого и закрытого). *Входные данные* – левая граница диапазона, с которой начинается процесс поиска простых чисел. *Выходные данные* – файлы `close_key.txt` и `open_key.txt`, содержащие значения полученных ключей. *Требования по функциональности*: в процессе генерации ключей программа выдает информацию о состоянии процесса (Progress Bar или что-то подобное); пользователь должен иметь возможность прервать процесс генерации в любое время.
    b. **Шифратор/Дешифратор** – модуль, осуществляющий кодирование/декодирование текстовых файлов по схеме, указанной в Таблице 1 согласно варианту. *Входные-выходные данные*: файл

close_key.txt или open_key.txt в зависимости от режима использования; файл pass.txt или pass.cod в зависимости от режима использования. Файл pass.txt содержит текстовый пароль, который необходимо зашифровать для последующего использования и записать в файл pass.cod. Файл pass. cod содержит зашифрованный текстовый пароль, который необходимо расшифровать для последующего использования и записать в файл pass.txt. *Требования по функциональности*: в процессе кодирования/декодирования программа выдает информацию о состоянии процесса (Progress Bar или что-то подобное); пользователь должен иметь возможность прервать процесс кодирования/декодирования в любое время; время обработки для текстового файла размером 10-30 символов не должно превышать 20 секунд.

c. **Блочный шифр** – берется из предыдущей лабораторной работы.
d. **Подсистему управления** – модуль, обеспечивающий частичную автоматизацию следующего сценария работы вышеперечисленных модулей (эти пункты отмечены звездочкой):
    i. Пользователь создает в текущей директории две папки Sender (имитация отправителя) и Recipient (имитация получателя), копирует в эти папки разработанную программу, а также исходные данные: в папку Sender – файл pass.txt и файл message.txt.
    ii. *Запускает генератор ключей и распределяет ключи (файл close_key.txt кладет в папку Recipient, а файл open_key.txt в папку Sender).
    iii. *Запускает шифратор в папке Sender, на вход которого подает файл с текстовым паролем pass.txt и файл open_key.txt и результат кодирования (файл pass.cod) сохраняет в папке и Recipient.
    iv. *Запускает дешифратор в папке Recipient, на вход которого подает файл с закодированным паролем pass.cod и файл close_key.txt и результат декодирования (файл pass.txt) сохраняет в папке и Recipient.
    v. *Запускает программу блочного шифрования в режиме кодирования в папке Sender, на вход которой подается пароль (файл pass.txt) и исходное сообщение (message.txt). Результат кодирования (файл message.cod) сохраняет в папке Recipient.
    vi. *Запускает программу блочного шифрования в режиме декодирования в папке Recipient, на вход которой

подается пароль (файл `pass.txt`) и зашифрованное сообщение (`message.cod`). Результат декодирования (файл `message.txt`) сохраняет в папке `Recipient`.

vii. Пользователь сравнивает полученные результаты, и если файлы `message.txt` в папках `Recipient` и `Sender` совпадают, то процесс кодирования считается успешно завершенным.

Входные данные: перед запуском управляющей программы папки должны содержать следующие исходные файлы с данными: в папке `Sender` файлы `pass.txt` и `message.txt`, в папке `Recipient` должно быть пусто.

Входные данные: после успешной работы программы в папке `Recipient` должны находится файлы `close_key.txt` (закрытый ключ), `pass.cod` (закодированный пароль), `pass.txt` (расшифрованный пароль), `message.cod` (зашифрованное сообщение) и `message.txt` (расшифрованное сообщение).

15. Алгоритмы работы с большими числами

В процессе работы с большими простыми числами очень часто возникает ситуация в которой алгоритмы прямого перебора работают слишком медленно и неэффективно, поэтому проектировщику асимметричных криптографических систем приходится использовать вероятностные методы и математические алгоритмы для оптимизации.

Критичными с точки зрения производительности, здесь являются:

1. Алгоритмы поиска больших простых чисел.
2. Алгоритмы нахождения взаимно простых больших чисел.
3. Алгоритмы возведения в степень в конечном поле.

Алгоритмы поиска больших простых чисел

Алгоритмы поиска больших простых чисел можно разделить на две группы: алгоритмы перебора и вероятностные алгоритмы. К первой группе относятся алгоритм прямого перебора с проверкой делимости и «решето Эратосфена». Как было упомянуто выше, такие алгоритмы неэффективны для больших чисел, поскольку работают очень медленно. Вторая группа алгоритмов использует вероятностные оценки для определения простоты числа. Наиболее широкое распространение получили алгоритм Рабина-Миллера [11,12] и тест Лемана [10,11].

Алгоритм Рабина Миллера предлагает следующую схему генерации простого числа:

1. Выберите для проверки случайное число p. Вычислите b - число делений p - 1 на 2 (т.е., 2b - это наибольшая степень числа 2, на которое делится p - 1). Затем вычислите m, такое что p = 1 + 2b * m.
2. Выберите случайное число a, меньшее p.
3. Установите j = 0 и z = am mod p.
4. Если z = 1 или если z = p - 1, то p проходит проверку и может быть простым числом.
5. Если j > 0 и z = 1, то p не является простым числом.
6. Установите j = j + 1. Если j < b и z(p - 1, установите z = z2 mod p и вернитесь на этап (4). Если z = p - 1, то p проходит проверку и может быть простым числом.
7. Если j = b и z ≠ □p - 1, то p не является простым числом.

Гарантируется, что три четверти возможных значений a окажутся свидетелями. Это означает, что составное число проскользнет через t проверок с вероятностью не большей (1/4) t , где t - это число итераций. На самом деле и эти оценки слишком пессимистичны. Для большинства случайных чисел около 99,9% возможных значений a являются свидетелями.

Альтернативный алгоритм был разработан Леманом. Вот последовательность действий при проверке простоты числа p:

1. Выберите случайно число a, меньшее p.
2. Вычислите $a^{(p-1)/2}$ mod p.
3. Если $a^{(p-1)}/2 \neq 1$ или –1 (mod p), то p не является простым.
4. Если $a^{(p-1)/2} \equiv$ □1 или –1 (mod p), то вероятность того, что число p не является простым, не больше 50 процентов.

И снова, вероятность того, что случайное число a будет свидетелем составной природы числа p, не меньше 50 процентов. Повторите эту проверку t раз. Если результат вычислений равен 1 или -1, но не всегда равен 1, то p является простым числом с вероятностью ошибки $(1/ 2)^t$.

Алгоритмы нахождения взаимно простых больших чисел

Два числа называются взаимно простыми, если у них нет общих множителей кроме 1. Иными словами, если наибольший общий делитель a и n равен 1. Это записывается как:

НОД(a,n)=1.

Взаимно просты числа 15 и 28. 15 и 27 не являются взаимно простыми, а 13 и 500 - являются. Простое число взаимно просто со всеми другими числами, кроме чисел, кратных данному простому числу. Одним из способов вычислить наибольший общий делитель двух чисел является алгоритм Эвклида.

На языке Си, алгоритм выглядит так:

```
int gcd (int x, int y) {
int g;
if (x < 0)
x = -x;
if (y < 0)
y = -y;
if (x + y == 0 )
ERROR ;
g = y;
while (x > 0) {
g = x;
x = y % x;
y = g;
}
return g;
```

В общем случае у уравнения $a^{-1} \equiv \square x \pmod{n}$ существует единственное решение, если a и n взаимно просты. Если a и n не являются взаимно простыми, то $a^{-1} \equiv x \pmod{n}$ не имеет решений. Если n является простым числом, то любое число от 1 до n–1 взаимно просто с n и имеет в точности одно обратное значение по модулю n.

Обратное значение a по модулю n можно вычислить с помощью алгоритма Эвклида. Иногда это называется расширенным алгоритмом Эвклида.

Вот этот алгоритм на языке Си:

```
//---------------------------------------------------------
#include <stdio.h>
#pragma hdrstop

//---------------------------------------------------------

#pragma argsused
void swap(unsigned long &x, unsigned long &y)
{
 unsigned long t;
 t = x;
 x = y;
 y = t;
}

int even(unsigned long x)
{
```

```
 return ((x & 0x01) == 0);
}

int odd(unsigned long x)
{
 return (x & 0x01);
}

void ExtBinEuclid(unsigned long u, unsigned long v, unsigned
long &result)
{
  int didswap = 0;
  unsigned long u1, u2, u3;
  unsigned long k, t1, t2, t3;

  if (even(u) && even(v)) return;

  if (u < v)
    {
      didswap = 1;
      swap (u, v);
    }

  for (k = 0 ; even(u) && even(v) ; k++)
    {
     printf("%d && %d = %d\n", even(u), even(v), even(u) &&
     even(v));
      u >>= 1;
      v >>= 1;
    }

  u1 = 1;
  u2 = 0;
  u3 = u;
  t1 = v;
  t2 = u - 1;
  t3 = v;
  do
    {
      do
        {
          if (even(u3))
            {
              if (odd(u1) || odd(u2))
                {
                  u1 += v;
                  u2 +=u;
                }
              u1 >>= 1;
              u2 >>= 1;
              u3 >>= 1;
```

```
            }
          if (even(t3) || (u3 < t3))
            {
              swap(u1, t1);
              swap(u2, t2);
              swap(u3, t3);
            }
        } while (even(u3));
      while ((u1 < t1) || (u2 < t2))
        {
          u1 += v;
          u2 += u;
        }
        u1 -= t1;
        u2 -= t2;
        u3 -= t3;

    } while (t3 > 0);
  while ((u1 >= v) && (u2 >= u))
    {
      u1 -= v;
      u2 -= u;
    }
   u1 <<= k;
   u2 <<= k;
   u3 <<= k;
   result = u - u2;
  if (didswap) swap(v, u);
}



int main(int argc, char* argv[])
{
 unsigned long u,v, result;
 printf("Enter u:");
 scanf("%ld",&u);
 printf("Enter v: ");
 scanf("%ld",&v);
 result = -1;
 ExtBinEuclid(u, v, result);
 printf("Inverse is: %ld",result);
       return 0;
}
```

Алгоритмы возведения в степень в конечном поле

В системах с открытым ключом активно используется техника возведения в степень в конечном поле или, как ее еще называют, арифметика

вычетов. Такой подход позволяет избежать ситуации переполнения разрядной сетки при работе с большими числами.

Арифметика вычетов очень похожа на обычную арифметику: она коммутативна, ассоциативна и дистрибутивна. Кроме того, приведение каждого промежуточного результата по модулю n дает тот же результат, как и выполнение всего вычисления с последующим приведением конечного результата по модулю n.

```
(a + b) mod n == ((a mod n) + (b mod n)) mod n
(a - b) mod n == ((a mod n) - (b mod n)) mod n
(a * b) mod n == ((a mod n) * (b mod n)) mod n
(a * (b+c)) mod n == (((a*b) mod n) + ((a*c) mod n)) mod n
```

Вычисление mod n часто используется в криптографии, так как вычисление дискретных логарифмов и квадратных корней mod n может быть нелегкой проблемой. Арифметика вычетов, к тому же, легче реализуется на компьютерах, поскольку она ограничивает диапазон промежуточных значений и результата. Для k-битовых вычетов n, промежуточные результаты любого сложения, вычитание или умножения будут не длиннее, чем 2^k бит.

Поэтому в арифметике вычетов мы можем выполнить возведение в степень без огромных промежуточных результатов. Вычисление степени некоторого числа по модулю другого числа, a x mod n, представляет собой просто последовательность умножений и делений, но существуют приемы, ускоряющие это действие. Один из таких приемов стремится минимизировать количество умножений по модулю, другой - оптимизировать отдельные умножения по модулю. Так как операции дистрибутивны, быстрее выполнить возведение в степень как поток последовательных умножений, каждый раз получая вычеты. Сейчас вы не чувствуете разницы, но она будет заметна при умножении 200-битовых чисел.

Например, если вы хотите вычислить a 8 mod n, не выполняйте наивно семь умножений и одно приведение по модулю:

$$(a * a * a * a * a * a * a * a) \bmod n.$$

Вместо этого выполните три меньших умножения и три меньших приведения по модулю:

$$((a^2 \bmod n)^2 \bmod n)^2 \bmod n.$$

Точно также,

$$a^{16} \bmod n = (((a^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n.$$

Вычисление a^x , где x не является степенью 2, ненамного труднее. Двоичная запись представляет x в виде суммы степеней двойки: 25 – это бинарное 11001, поэтому $25 = 2^4 + 2^3 + 2^0$. Поэтому:

$$a^{25} \bmod n = (a*a^{24}) \bmod n = (a*a^8*a^{16}) \bmod n =$$
$$(a*((a^2)^2)^2*(((a^2)^2)^2)^2) \bmod n = (a*(((a*a^2)^2)^2)^2) \bmod n$$

С продуманным сохранением промежуточных результатов вам понадобится только шесть умножений:

$$((((((a^2 \bmod n)*a)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 \bmod n)^2 * a) \bmod n$$

Такой прием называется цепочкой сложений, или методом двоичных квадратов и умножения. Он использует простую и очевидную цепочку сложений, в основе которой лежит двоичное представление числа. На языке Си это выглядит следующим образом:

```
unsigned long qe2(unsigned long x, unsigned long y, unsigned
long n) {
unsigned long s, t, u;
int i;
s=1; t=x; u=y;
while (u) {
if(u&1) s=(s*t)%n;
u>>1;
t=(t*t)%n;
}
return(s)
```

Контрольные вопросы ко второй главе

1. Дайте определение понятию «Криптография».
2. Дайте определение понятию «Шифр».
3. Как называется конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования данных, обеспечивающее выбор одного варианта из совокупности возможных для данного алгоритма?
4. Как называется соотношение, описывающее процесс образования зашифрованных данных из открытых?
5. Какой термин используется для определения качества шифра?
6. Дайте определение понятию «Секретная система».
7. Что является решением общей задачи дешифрования?
8. Перечислите основные виды современных криптосистем.
9. Какие режимы использования блочных шифров используются в современной криптографии?
10. Какие режимы использования блочных шифров позволяют осуществлять простое распараллеливание?
11. С какой целью применяются различные режимы использования блочных шифров?
12. Перечислите, в каких известных стандартах шифрования используется сеть Фейстела?
13. Какие виды необратимых преобразований используются в современной криптографии с открытыми ключами?
14. На основе каких необратимых преобразований базируется алгоритм RSA?
15. На основе каких необратимых преобразований базируется алгоритм Эль-Гамаля?
16. Приведите примеры взаимно простых чисел.
17. Перечислите алгоритмы, используемые для нахождения больших простых чисел.

ЗАКЛЮЧЕНИЕ

Современные методы и задачи криптографической защиты данных очень разнообразны. Каждый из них имеет свою определенную область эффективности, в которой достигается максимальный эффект от их использования. В то же время, построение комплексной системы защиты включает большое количество разнообразных методов и средств, направленных на создание интегрированной системы защиты.

Требования, которые предъявляются сегодня к системам защиты, очень высоки. Рассмотренные в учебном пособии конкретные примеры различных реализаций позволяют в наибольшей степени осознать проблематику предметной и области, и дать будущим инженерам инструменты эффективной работы в сфере симметричных и ассиметричных криптографических систем.

Предложенные алгоритмы могут быть использованы не только в учебном процессе, но и при построении реальных систем защиты компьютерной информации, поскольку криптография сегодня используется и при построении систем шифрования, и для защиты сетевого трафика, и, даже при построении анализаторов кода антивирусных программ.

ПРИЛОЖЕНИЕ А

Таблица А. Константы для линейных конгруэнтных генераторов

Переполняется при	*a*	*b*	*m*
2^{20}	106	1283	6075
2^{21}	211	1663	7875
2^{22}	421	1663	7875
2^{23}	430	2531	11979
2^{23}	936	1399	6655
2^{23}	1366	1283	6075
2^{24}	171	11213	53125
2^{24}	859	2531	11979
2^{24}	419	6173	29282
2^{24}	967	3041	14406
2^{25}	141	28411	134456
2^{25}	625	6571	31104
2^{25}	1541	2957	14000
2^{25}	1741	2731	12960
2^{25}	1291	4621	21870
2^{25}	205	29573	139968
2^{26}	421	17117	81000
2^{26}	1255	6173	29282
2^{26}	281	28411	134456
2^{27}	1093	18257	86436
2^{27}	421	54773	259200
2^{27}	1021	24631	116640
2^{27}	1021	25673	121500
2^{28}	1277	24749	117128
2^{28}	741	66037	312500
2^{28}	2041	25673	121500
2^{29}	2311	25367	120050
2^{29}	1807	45289	214326
2^{29}	1597	51749	244944
2^{29}	1861	49297	233280
2^{29}	2661	36979	175000
2^{29}	4081	25673	121500
2^{29}	3661	30809	145800
2^{30}	3877	29573	139968
2^{30}	3613	45289	214326
2^{30}	1366	150889	714025
2^{31}	8121	28411	134456
2^{30}	4561	51349	243000
2^{30}	7141	54773	259200
2^{32}	9301	49297	233280
2^{32}	4096	150889	714025
2^{33}	2416	374441	1771875
2^{34}	17221	107839	510300

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Введение в защиту информации : учеб. пособие для вузов / В. Б. Байбурин [и др.]. - М. : ФОРУМ : ИНФРА-М, 2004. – 127 с. – (Профессиональное образование).
2. Малюк, А. А. Информационная безопасность: концептуальные и методологические основы защиты информации : учеб. пособие для вузов / Малюк, Анатолий Александрович. – М. : Горячая линия-Телеком, 2004. – 280 с. : ил.
3. Галатенко, В. А. Основы информационной безопасности : курс лекций : учеб. пособие : для студентов вузов, обучающихся по спец. 351400 "Прикладная информатика" / В. А. Галатенко ; под ред. В. Б. Бетелина. – М. : Интернет-университет информационных технологий, 2004. – 260 с. : ил. – (Серия "Основы информационных технологий").
4. Рябко, Б. Я. Криптографические методы защиты информации : учеб. пособие для вузов / Б. Я. Рябко, А. Н. Фионов. – М. : Горячая линия-Телеком, 2005. – 229 с. : ил.
5. Зубов, А. Ю. Криптографические методы защиты информации. Совершенные шифты : учеб. пособие для вузов / А. Ю. Зубов. – М. : Гелиос АРВ, 2005. – 191 с. : табл.
6. Мельников, В. П. Информационная безопасность и защита информации : учеб. пособие для вузов / В. П. Мельников, С. А. Клейменов, А. М. Петраков ; отв. ред. С. А. Клейменова. – М. : Академия, 2006. – 331 с. : ил. – (Высшее профессиональное образование. Информатика и вычислительная техника).
7. Куприянов, А. И. Основы защиты информации : учеб. пособие / А. И. Куприянов, А. В. Сахаров, В. А. Шевцов. – М. : Академия, 2006. – 254 с. : ил. – (Высшее профессиональное образование. Радиоэлектроника).
8. Девянин, П. Н. Модели безопасности компьютерных систем : учеб. пособие для вузов / П. Н. Девянин. – М. : Академия, 2005. – 143 с. : ил. – (Высшее профессиональное образование. Информационная безопасность).
9. Безопасность сетей NT4 : Пер. с англ. : В 2 т. Т. 1 / Штребе, Мэтью [и др.]. – М. : Мир, 1999. – 367с. : ил. – Прил.CD-ROM.
10. Безопасность сетей NT4 : Пер. с англ. : В 2 т. Т. 2 / Штребе, Мэтью [и др.]. – М. : Мир, 1999. – 735с. : ил. – Прил.CD-ROM.
11. Защита информации в компьютерных системах и сетях / Романец, Ю. В. [и др.]. – 2-е изд., перераб. и доп. – М. : Радио и связь, 2001. – 376с.

12. Анин, Б. Ю. Защита компьютерной информации / Анин, Борис Ю. – СПб. : BHV-Санкт-Петербург, 2000. – 373с. : ил. – (Мастер. Современные технологии).
13. Чирилло, Д. Защита от хакеров : Пер. с англ. / Чирилло, Джон. – СПб. : Питер, 2002. – 472с. : ил. – (Для профессионалов). – Прил. CD-ROM.
14. Форд, Д. Л. Персональная защита от хакеров : Руководство для начинающих: Пер. с англ. / Форд, Джерри Ли. – М. : КУДИЦ-ОБРАЗ, 2002. – 272с. : ил.
15. Фомичев, В. М. Дискретная математика и криптология : Курс лекций: / Фомичев, Владимир Михайлович ; Под общ. ред. Н. Д. Подуфалова. – М. : ДИАЛОГ-МИФИ, 2003. – 397 с. : табл.
16. Щеглов, А. Ю. Защита компьютерной информации от несанкционированного доступа / Щеглов, Андрей Юрьевич. – СПб. : Наука и техника, 2004. – 384 с. : ил. – (Профи).

Учебное пособие
МАРТЫНОВ Антон Иванович
Методы и задачи криптографической защиты информации
Учебное пособие
Редактор Н.А. Евдокимова
Подписано в печать 20.12.2007. Формат 60x84/16.
Бумага тип. №1. Усл. печ. л. 5,58. Тираж 100 экз.
Заказ
Ульяновский государственный технический университет
432027, г. Ульяновск, ул. Сев. Венец, д. 32.
Типография УлГТУ, 432027, ул. Сев. Венец, д. 32.