П. П. Урбанович, Д. М. Романенко, Е. В. Романцевич

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ И НАДЕЖНОСТЬ СИСТЕМ

Рекомендовано учебно-методическим объединением
Высших учебных заведений Республики Беларусь
по образованию в области информатики и радиоэлектроники
в качестве учебно-методического пособия
по одноименному курсу для студентов специальности 1-40 01 02-03
«Информационные системы и технологии»

Минск 2007

УДК 004.056(075.8)
БКК 32.97я7
У 69

Рецензенты:
кафедра электронно-вычислительных средств Белорусского государственного университета информатики и радиоэлектроники (заведующий кафедрой доктор технических наук, профессор *А. А. Петровский*);
директор учреждения «Главный информационно-аналитический центр Министерства образования Республики Беларусь» доктор технических наук, профессор *Н. И. Листопад*

Урбанович, П. П.

У 69 Информационная безопасность и надежность систем : учеб.-метод. пособие для студентов специальности 1-40 01 02-03 «Информационные системы и технологии» / П. П. Урбанович, Д. М. Романенко, Е. В. Романцевич. – Минск : БГТУ, 2007. – 90 с.

ISBN 978-985-434-750-9.

Учебно-методическое пособие предназначено для студентов, изучающих дисциплину «Информационная безопасность и надежность систем». Издание имеет своей целью помочь обучаемым в овладении теоретическими знаниями о методах и средствах повышения информационной безопасности и надежности систем хранения, преобразования и передачи информации, ее защиты в информационно-вычислительных системах, в приобретении практических навыков по созданию и использованию методов и средств повышения информационной безопасности и надежности систем.

Пособие также может быть полезно аспирантам, изучающим избранные аспекты вышеуказанной проблемы.

УДК 004.056(075.8)
ББК 32.97я7

ISBN 978-985-434-750-9

ПРЕДИСЛОВИЕ

Эффективность информационных систем (средств вычислительной и измерительной техники, автоматизированных систем управления, систем хранения, преобразования и передачи информации и др.) и программных комплексов в значительной степени определяется уровнем защищенности информации от постороннего (несанкционированного) доступа и воздействия на нее, а также достоверностью информации, которая перерабатывается в этих системах и комплексах. Указанные особенности имеют непосредственное отношение к проблемам безопасности и надежности информационных систем.

Достоверность информации зависит от целого ряда факторов, как технических, обусловленных конкретной реализацией и стратегией обслуживания таких систем и устройств, так и смысловых, связанных с реализованными методами представления и преобразования информации.

Защищенность информации (каналов ее передачи и хранения) определяется простотой (или сложностью) осуществления несанкционированного доступа к носителям информации (устройствам хранения), техническим и программным средствам, реализующим операции над информацией.

В последнее время все чаще защищенность и достоверность информации рассматриваются как равнозначные аспекты двуединой проблемы проектирования, разработки и эксплуатации безопасных и высоконадежных систем. Такой подход базируется, прежде всего, на том, что современные информационные системы различного назначения – компьютерные системы на основе сетей и сетевых технологий.

В результате изучения дисциплины и выполнения заданий на лабораторных занятиях (приводятся в конце разделов) студент должен знать:

− особенности информационно-вычислительных систем (ИВС) как объекта защиты, соответствующего требуемому уровню надежности функционирования;

− организационные и правовые методы защиты информации в ИВС;

− программно-технические средства преобразования и защиты информации в ИВС, повышения надежности ИВС;

− методы криптографической защиты информации и ИВС.

Студент должен научиться применять рассматриваемые методы на практике.

1. ВВЕДЕНИЕ В ПРОБЛЕМУ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ И НАДЕЖНОСТИ СИСТЕМ

1.1. Основные понятия и определения

Информация – сведения (данные) [1] о внутреннем и окружающем нас мире, событиях, процессах, явлениях и т. д., воспринимаемые и передаваемые людьми или техническими устройствами.

Информационная (информационно-вычислительная) система – организационно упорядоченная совокупность документов, технических средств и информационных технологий, реализующая информационные (информационно-вычислительные) процессы.

Информационные процессы – процессы сбора, накопления, хранения, обработки (переработки), передачи и использования информации.

Информационные ресурсы – отдельные документы или массивы документов в информационных системах.

Доступ – специальный тип взаимодействия между объектом и субъектом, в результате которого создается поток информации от одного к другому.

Несанкционированный доступ – доступ к информации, устройствам ее хранения и обработки, а также к каналам передачи, реализуемый без ведома (санкции) владельца и нарушающий тем самым установленные правила доступа.

Объект – пассивный компонент системы, хранящий, перерабатывающий, передающий или принимающий информацию; примеры объектов: страницы, файлы, папки, директории, компьютерные программы, устройства (мониторы, диски, принтеры и т. д.).

Субъект – активный компонент системы, который может инициировать поток информации; примеры субъектов: пользователь, процесс либо устройство.

Безопасность ИВС – свойство системы, выражающееся в способности системы противодействовать попыткам несанкционированного доступа или нанесения ущерба владельцам и

[1] Строго говоря, в теории информации понятия «данные» и «информация» не отождествляются.

пользователям системы при различных умышленных и неумышленных воздействиях на нее.

Защита информации – организационные, правовые, программно-технические и иные меры по предотвращению угроз информационной безопасности и устранению их последствий.

Информационная безопасность систем – свойство информационной системы или реализуемого в ней процесса, характеризующее способность обеспечить необходимый уровень своей защиты.

Надежность системы – характеристика способности программного, аппаратного, аппаратно-программного средства выполнить при определенных условиях требуемые функции в течение определенного периода времени.

Достоверность работы системы (устройства) – свойство, характеризующее истинность конечного (выходного) результата работы (выполнения программы), определяемое способностью средств контроля фиксировать правильность или ошибочность работы.

Ошибка устройства – неправильное значение сигнала (бита – в цифровом устройстве) на внешних выходах устройства или отдельного его узла, вызванное технической неисправностью или воздействующими на него помехами (преднамеренными либо непреднамеренными).

Ошибка программы проявляется в не соответствии реальному (требуемому) промежуточному или конечному значению (результату) вследствие неправильно запрограммированного алгоритма или неправильно составленной программы.

1.2. Оценка надежности цифровых систем и каналов передачи информации

Как следует из вышеприведенного определения, ***надежность*** есть внутреннее свойство объекта, заложенное в него при изготовлении и проявляющееся во время эксплуатации. Вторая особенность надежности состоит в том, что она обнаруживается во времени. И третья особенность заключается в том, что надежность выражается по-разному при различных условиях эксплуатации и различных режимах применения *объекта* (технического объекта).

Надежность является комплексным свойством, включающим в себя единичные свойства: безотказность, ремонтопригодность, сохраняемость, долговечность.

Безотказность – это свойство технического объекта непрерывно сохранять работоспособное состояние в течение некоторого времени (или *наработки*). Наработка, как правило, измеряется в единицах времени.

Ремонтопригодность – это свойство технического объекта, заключающееся в приспособленности к поддержанию и восстановлению работоспособного состояния путем технического обслуживания, ремонта (или с помощью дополнительных, избыточных технических средств, функционирующих параллельно с объектом). Большинство современных цифровых систем и устройств (в том числе компьютеры и компьютерные системы, отдельные блоки и модули компьютеров –полупроводниковая, магнитная или оптическая память) содержат специальные средства, призванные автоматически восстанавливать работоспособность этих объектов при нарушении нормального функционирования.

Такие специальные средства контроля называются *избыточными*. На рис. 1.1 приведена упрощенная структурная схема ИВС с избыточными средствами аппаратного контроля.

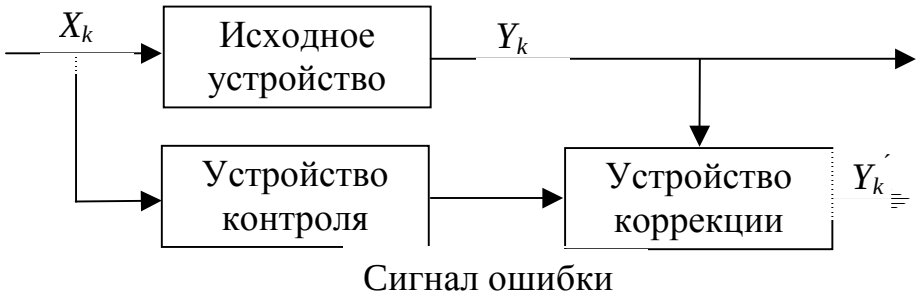

Рис. 1.1. Структурная схема ИВС со средствами аппаратного контроля достоверности функционирования и коррекции выходной информационной последовательности

Изначальной причиной нарушения нормальной работы цифрового устройства являются технические *дефекты* (неисправности), возникающие внутри узлов или блоков устройства либо в линиях связи между ними.

Дефекты или неисправности могут приводить либо к кратковременному нарушению достоверности работы устройства (*сбой*), либо к полной и окончательной потере достоверности (*отказ*).

В каждом из этих случаев следствием неисправности являются ошибки в информации (*информационные ошибки*). Количество таких ошибок (количество ошибочных двоичных символов) принято называть *кратностью ошибки*.

Пример 1.1. Исходная (правильная) информационная последовательность $X_k = 1001$. Длина этой последовательности равна 4 битам ($k = 4$). Одна из перечисленных причин привела к тому, что в этой последовательности появились две ошибки (кратность ошибки равна двум): $Y_k = 1\underline{11}1$.

Устройства контроля и коррекции являются избыточными и предназначены соответственно для контроля за появлением ошибок и коррекции (исправления) этих ошибок.

По возможности восстановления работоспособности после отказа различают невосстанавливаемые и восстанавливаемые объекты. В контексте дальнейшего анализа *невосстанавливаемым* будем считать цифровое устройство без избыточных схем контроля и, наоборот, объект с контролем функционирования относим к классу *восстанавливаемых*.

Третье свойство надежности – ***сохраняемость*** – характеризует способность технического объекта сохранять в заданных пределах значения параметров после его хранения или транспортировки.

Долговечность – это свойство технического объекта сохранять в заданных пределах работоспособное состояние до наступления предельного состояния при установленной системе технического обслуживания или ремонта.

Возможные состояния анализируемой ИВС (рис. 1.1) можно охарактеризовать следующим образом:

1) исходное устройство действительно работает правильно ($X_k = Y_k$) в течение периода времени t, вероятность такого события обозначим $P_{\text{пр}}(t)$;

2) исходное устройство работает с ошибкой ($X_k = Y_k$), о чем свидетельствует сигнал ошибки, вероятность этого события (правильное обнаружение) – $P_{\text{п.о}}(t)$;

3) исходное устройство работает неправильно, однако это состояние устройством контроля не обнаруживается (пропуск, необнаружение ошибки), соответствующая вероятность – $P_{\text{н.о}}(t)$;

4) исходное устройство работает правильно, однако устройство контроля выдает информацию об ошибке (состояние ложной тревоги), причиной чего может быть недостоверное

функционирование самого устройства контроля, вероятность такого события – $P_{\text{л.т}}(t)$.

Все перечисленные события образуют полную группу событий, описываемую следующим вероятностным соотношением:

$$P_{\text{пр}}(t) + P_{\text{п.о}}(t) + P_{\text{н.о}}(t) + P_{\text{л.т}}(t) = 1. \qquad (1.1)$$

Надежность недостаточно определить на качественном уровне (высокая, низкая) – необходимо уметь оценивать ее качественно. Приведенные количественные параметры (вероятности) являются косвенными показателями надежности цифрового устройства.

Основным количественным показателем надежности (имеет отношение, прежде всего, к невосстанавливаемым объектам) является *наработка до первого отказа* T_0. Вероятностные характеристики наработки являются показателями безотказности объекта. Рассмотрим некоторые из них.

Вероятностью безотказной работы $P(t)$ называют вероятность того, что изделие (система, устройство) будет работоспособно в течение заданной наработки при заданных условиях эксплуатации:

$$P(t) = P(T_0 > t).$$

Данная вероятность может быть рассчитана на основе статистических данных. Если принять, что $N_{\text{с}}$ соответствует суммарному числу изделий (объектов), из которых $N_{\text{о}}$ за время наблюдения t отказали (стали дефектными), то при достаточно большом числе $N_{\text{с}}$ вероятность может быть определена как

$$P(t) = \frac{N_{\text{с}} - N_{\text{о}}}{N_{\text{с}}}. \qquad (1.2)$$

Эта вероятность соответствует вероятности $P_{\text{пр}}(t)$.

Пример 1.2. Из партии постоянно функционирующих компьютеров в количестве 1000 штук за время эксплуатации t в 6 изделиях возникли отказы. Тогда вероятность безотказной работы произвольного компьютера того же производителя при неизменном технологическом процессе изготовления составляет в соответствии с формулой (1.2):

$$P(t) = \frac{N_{\text{с}} - N_{\text{о}}}{N_{\text{с}}} = \frac{1000 - 6}{1000} = 0.994.$$

Пример 1.3. В течение фиксированного времени (например, $t = 1$ ч) по каналам связи осуществлялась передача двоичной информации между двумя компьютерами со скоростью $S = 10$ Кбит/с. За время передачи 1000 символов были приняты с ошибками. Определим вероятность того, что произвольный двоичный символ при передаче по тому же каналу будет принят правильно.

Нетрудно обнаружить аналогию в двух последних задачах. Если принять, что $N_{\text{с}} = S \cdot t = 10\,000 \cdot 3600 = 36 \cdot 10^6$ бит, тогда искомая вероятность вычисляется как

$$P(t) = \frac{36 \cdot 10^6 - 1000}{36 \cdot 10^6} = \frac{35\,999\,000}{36\,000\,000} = 0.99997.$$

Вероятность отказа $Q(t)$ есть вероятность того, что при заданных условиях эксплуатации в течение заданной наработки произойдет хотя бы один отказ, т. е.

$Q(t) = P(T_0 < t).$

Отказ и безотказная работа – противоположные события. Поэтому

$$Q(t) = 1 - P(t). \qquad (1.3)$$

С другой стороны, следуя рассуждениям, на основе которых записано соотношение (1.2), можем также утверждать, что

$$Q(t) = \frac{N_{\text{о}}}{N_{\text{с}}}. \qquad (1.4)$$

Вероятность $Q(t)$ в некоторых случаях соответствует вероятности $P_{\text{п.о}}(t)$.

Пример 1.4. Из условий примера 1.2 определим вероятность отказа произвольного компьютера за время t. Как следует из формул (1.3) и (1.4), искомая величина равна 0.006.

Пример 1.5. Из условий примера 1.3 найдем вероятность приема бита с ошибкой. Легко установить, что эта величина составляет $2.78 \cdot 10^{-5}$.

Интенсивность отказов $\lambda(t)$ – плотность распределения наработки до первого отказа при условии, что отказавший объект до рассматриваемого момента времени работал безотказно. Согласно вероятностному определению,

$\lambda(t) = -\ln P(t),$

$$P(t) = -\exp\left(\int_0^t \lambda(x)dx\right).$$

По статистическому определению интенсивность отказов есть отношение числа объектов наблюдения, отказавших в единицу времени, к среднему числу объектов, работоспособных на рассматриваемом отрезке времени.

Если за такой отрезок времени принять 1 ч, то по условиям примера 1.3 получаем $\lambda(t) = 1000\ \text{ч}^{-1}$.

Как видим, между тремя рассмотренными количественными характеристиками надежности $(P(t), Q(t), \lambda(t))$ существует однозначная связь. Достаточно задать одну из них, чтобы определить остальные.

1.3. Методы повышения аппаратной надежности

Вопросам обеспечения надежности уделяется внимание на всех этапах жизненного цикла (проектирование, изготовление, эксплуатация) устройств и каналов передачи информации. Мировой опыт показывает, что значительный эффект при решении задач обеспечения качества и надежности дает системная организация работ на основе внедрения международных стандартов серии ISO 9000 или TQM (Total Quality Management), разработанных Международной организацией по стандартизации (International Standard Organization, ISO).

Даже краткое перечисление основных направлений работ по обеспечению надежности показывает, что современная техника обладает широким арсеналом методов повышения надежности. Эти методы можно условно разделить на следующие *группы*:

1) уменьшение наработки;
2) снижение интенсивности отказов;
3) улучшение восстанавливаемости;
4) резервирование.

Методы первой группы основываются на использовании более быстродействующих объектов. Снижение интенсивности отказов (а также сбоев) достигается в основном технологией изготовления (собственно, как и уменьшение наработки). Нас интересуют остальные группы методов, поскольку к их реализации может иметь непосредственное отношение проектировщик (и пользователь) всей системы, а не отдельных ее блоков. Улучшение восстанавливаемости (нормального функционирования) может быть достигнуто с применением параллельных средств контроля (рис. 1.1). Вместе с тем такие избыточные средства контроля можно отнести к методам,

основанным на использовании резервирования. С такой точки зрения можно третью и четвертую группы объединить под единым названием – *избыточные методы*.

Будем различать структурную, временную, информационную избыточности либо их комбинации.

Простая структурная избыточность нередко называется ***структурным резервированием***. Методы структурной избыточности подразумевают повышение надежности аппаратуры или каналов передачи за счет использования дополнительных (резервных) модулей или каналов. Характерной особенностью систем со структурным резервированием является то, что в идеально надежной системе все резервные элементы могут быть удалены без какого-либо ухудшения качества системы. Они необходимы только тогда, когда возникает принципиальная возможность появления отказа элементов системы. В соответствии с уровнем этой принципиальной возможности, назначением системы и критичностью (стоимостью) обрабатываемой информации структурный резерв может быть «холодным» или «горячим».

Структурная схема системы с «холодным» (ждущим) резервом может быть получена изменением функций системы, представленной выше на рис. 1.1. Такая видоизмененная структура показана на рис. 1.2.

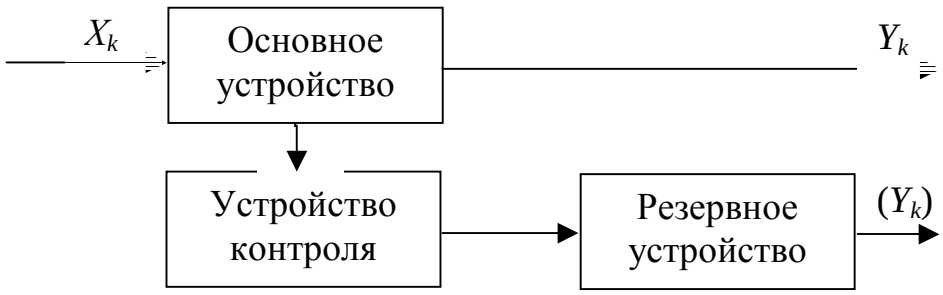

Рис. 1.2. Обобщенная структурная схема системы с «холодным» резервом

Резервное устройство (канал) начинает выполнять функции основного (исходного) после установления устройством контроля отказа основным устройством, который привел к ошибкам в обрабатываемой информации. Надежность всей системы определяется интенсивностями отказов не только основного (исходного) устройства ($\lambda_{\text{о.у}(t)}$), но и дополнительных (устройства контроля и резервного устройства $\lambda_{\text{у.к}}(t)$, $\lambda_{\text{р.у}}(t)$). Понятно, что вся система может находиться в работоспособном состоянии после отказа

основного устройства только при условии безотказной работы остальных блоков. При расчетах общей интенсивности отказов системы, равной сумме интенсивностей отказов составляющих ее модулей, принимается, что $\lambda_{\text{у.к}}(t)$, $\lambda_{\text{р.у}}(t) << \lambda_{\text{о.у}}(t)$.

Резервирование с постоянно включенным («горячим») резервом предполагает, что все модули системы начинают функционировать одновременно, начиная с момента включения системы. Такой способ наиболее часто реализуется в системах на основе мажоритарного определения выходного сигнала (Y_k) по принципу большинства. Пример реализации показан на рис. 1.3.

При таком способе резервирования допускается, что резервные устройства (модули) полностью идентичны основному устройству, и на их выходах при безотказном функционировании должны вырабатываться одинаковые сигналы (последовательности). Выходной сигнал резервированной системы (Y_k) формируется по принципу большинства: один из трех (при одинаковых сигналах Y_{k1}, Y_{k2}, Y_{k3}) или один из двух (если сигнал на выходе одного из трех устройств отличается от двух других). Предполагается, что интенсивности отказов в основном и в резервных устройствах одинаковы: $\lambda_{\text{о.у}}(t) = \lambda_{\text{р.у}A}(t) = \lambda_{\text{р.у}B}(t)$ и $\lambda_{\text{о.у}}(t) >> \lambda_{\text{м.с}}(t)$, где $\lambda_{\text{м.с}}(t)$ – интенсивность отказов мажоритарной схемы. Исходя из этого (интенсивностью отказов мажоритарного устройства пренебрегаем), можно рассчитать вероятность безотказной работы системы:

$$P(t) = P_3(t) + P_2(t),$$

где $P_3(t)$ – вероятность того, что все три устройства (основное и два резервных) работают безотказно; $P_2(t)$ – вероятность того, что два устройства из трех указанных работают безотказно.

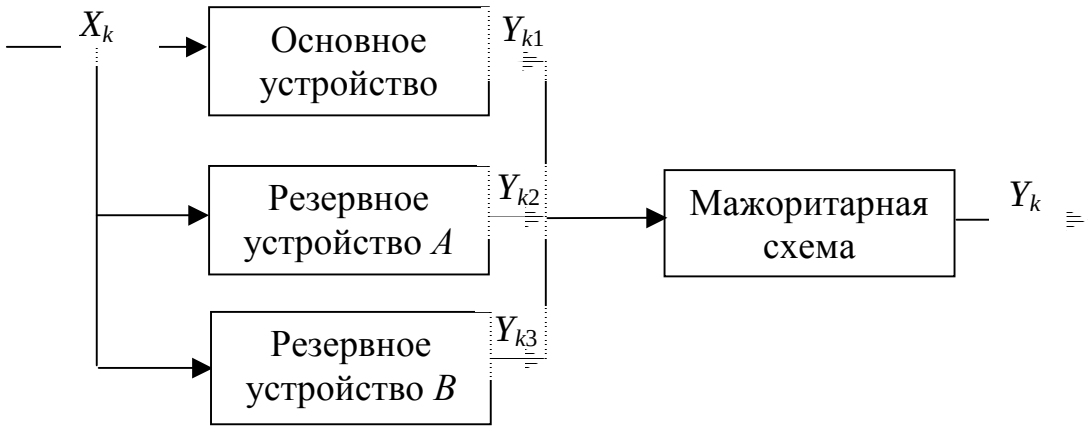

Рис. 1.3. Обобщенная структурная схема с резервированием

Нетрудно подсчитать, что

$$P_3(t) = P_{\text{о.у}}(t) \cdot P_{\text{р.у}A}(t) \cdot P_{\text{р.у}B}(t),$$

$$P_2(t) = P_{\text{о.у}}(t) \cdot P_{\text{р.у}A}(t) \cdot (1 - P_{\text{р.у}B}(t)) + P_{\text{о.у}}(t) \cdot P_{\text{р.у}B}(t) \cdot (1 - P_{\text{р.у}A}(t)) +$$
$$+ P_{\text{р.у}A}(t) \cdot P_{\text{р.у}B}(t) \cdot (1 - P_{\text{о.у}}(t)) = 3 \cdot (P_{\text{о.у}}(t))^2 \cdot (1 - P_{\text{о.у}}(t)).$$

Из сделанного анализа принципов функционирования резервных устройств видно, что они характеризуются также временной избыточностью: формирование выходного сигнала требует дополнительного времени. Однако классическими системами с временной избыточностью принято считать такие, которые характеризуются также информационной избыточностью на основе использования специальных кодов, обнаруживающих и корректирующих ошибки в информационных последовательностях (словах). Об этом более подробно речь пойдет в разделе 3.

1.4. Методы и технологии обеспечения надежности программных комплексов

В составе «мягкого оборудования» (software) аппаратно-программных комплексов (АПК) можно выделить три *части*:

1) математическое обеспечение (МО);

2) информационное обеспечение (ИО);

3) программное обеспечение (ПО).

Первую часть (МО) составляет совокупность математических моделей, методов и алгоритмов, на основе которых АПК выполняет задачи в соответствии с целевым назначением. Информационное обеспечение есть совокупность баз данных, файловых структур и других элементов, определяющих формирование, подготовку и условия обработки данных. И, наконец, программное обеспечение АПК реализует его МО на основе ИО. Следует также подчеркнуть, что методы преобразования информации могут быть реализованы аппаратно, программно либо аппаратно-программно.

Последствия ошибок в «мягком оборудовании» не отличаются от последствий отказов аппаратных средств. В этом смысле говорят об отказах и надежности программных средств (и о надежности МО, ИО).

Обеспечение корректности программ и данных означает недопущение ошибок в элементах МО, ИО, ПО до начала их

эксплуатации. Компьютерная программа как результат реализации алгоритмов на языке программирования считается корректной, если она удовлетворяет своим функциональным спецификациям.

Основными причинами ошибок в ИО являются необнаруженные при подготовке данных и их вводе неточности, ошибки адресации, ошибочные константы, операнды и т. д., а также искажения данных вследствие несанкционированного доступа к системе.

К основным *причинам ошибок* ПО относятся следующие:

– незамкнутость реализованного алгоритма, вызванная ошибками МО;

– синтаксические ошибки, допущенные при реализации алгоритма на конкретном языке программирования, неправильное применение конструкций языка, системные ошибки;

– неправильная отладка программы после выявления ошибок, которая приводит к новым ошибкам.

Методы обеспечения безошибочности данных могут реализовываться при проектировании программных комплексов. К таким методам относят использование автоматических процедур проверки корректности данных (например, проверку совместимости элементов структур данных); дублирование работ при подготовке данных и др.

Методы обеспечения безошибочности программ являются одной из важнейших характеристик технологий программирования. К основным из таких методов относят снижение сложности программ, широкое применение систем автоматизации проектирования ПО, использование языков программирования высокого уровня и др.

Основным средством уменьшения числа ошибок, допущенных при разработке ПО, является отладка. При создании условий для наиболее эффективного выявления ошибок следует иметь в виду, что ПО не отказывает, когда не находится в работе.

По определению программа считается отказоустойчивой, если она обеспечивает полезную работоспособность не ниже заданного минимального уровня при наличии программных ошибок. Разработка ПО, нечувствительного к определенным типам ошибок и искажений, считается более перспективным направлением по сравнению с разработкой безошибочных программ.

1.5. Общая характеристика безопасности информационных систем

Безопасность системы/процесса характеризует способность системы/процесса противодействовать несанкционированному доступу к ней вне зависимости от целей этого доступа. Параметрами безопасности являются конфиденциальность, целостность, доступность.

Конфиденциальность информации – свойство информации быть известной только допущенным и прошедшим авторизацию субъектам системы (пользователям, программам, процессам и др.).

Целостность – состояние данных или компьютерной системы, в которой данные и программы используются установленным способом, обеспечивающим устойчивую работу системы и единство данных.

Доступность компонента (ресурса) системы – свойство компонента (ресурса) быть доступным для использования авторизованными субъектами системы в любое время в соответствии с установленным регламентом.

Принято считать, что *информация стала самым дорогим продуктом* в сфере межличностных отношений. Именно поэтому информация и информационные системы все чаще становятся объектами атак (несанкционированного доступа).

Различают следующие основные *методы* несанкционированного доступа:

– физические (например, простое копирование);

– программные (вирусы, «троянские кони», клавиатурные анализаторы, анализаторы протоколов и другие деструктивные программы);

– электронные (на основе исследования электромагнитного излучения от различных блоков и устройств компьютерной техники, а также каналов передачи информации);

– технические (непосредственное подключение к каналам связи).

Все многообразие методов и средств противодействия несанкционированному доступу условно подразделяется на следующие группы:

1. Организационные методы и средства. Подразумевают разработку и исполнение в любой организации/лаборатории правил, регламентирующих и регулирующих доступ физических лиц к информации, хранящейся на носителях либо передаваемой внутри сети данного предприятия. В организациях, осуществляющих операции над критической информацией (правительственные, государственные, банковские, коммерческие и иные структуры),

назначается специальное ответственное лицо – администратор безопасности (АБ), ответственный за реализацию и соблюдение правил на основе реализуемой политики безопасности.

2. Правовые методы. Гражданский правовой кодекс предусматривает наказание за компьютерные преступления. В 1983 г. Организация экономического сотрудничества и развития определила под термином «компьютерная преступность» (или «связанная с компьютерами преступность») любые незаконные, неэтичные или неправомерные действия, связанные с автоматической обработкой или передачей информации. Практически во всех странах с развитой информационной инфраструктурой (в том числе и в Беларуси) предусматривается уголовно-правовая защита от компьютерных преступлений.

3. Физические методы. Объединяют методы ограничения физического доступа лиц к каналам передачи информации, устройствам ее хранения и обработки. Основаны на использовании простых замков, магнитных карт, чипов, таблеток, на анализе антропометрических и биологических параметров человека (сетчатка глаза, отпечатки пальцев и др.).

4. Программно-технические методы. Базируются на применении аппаратных и/или программных средств, позволяющих идентифицировать пользователя (либо техническое средство), а также оценить происхождение программного средства, поступающего в информационную сеть. Наиболее известными из указанных средств являются использование пароля, антивирусных программ, брандмауэров, или «огненных стен» (firewalls) на входе сети, криптографического преобразования информации на основе методов шифрования.

Далее в пособии в основном будем анализировать методы именно этой группы. Однако здесь подчеркнем простой и неоспоримый факт: абсолютно защищенный персональный компьютер (или компьютерная сеть), операционная система, прикладная программа – такая же иллюзия, как и абсолютно надежно охраняемый дом. Защита ваших данных и каналов связи с Интернетом от воздействия *интрузов* (нежелательных программ, или физических лиц) является, по определению, результатом известного компромисса. Этот компромисс базируется на важнейшем и универсальном подходе к разработке и реализации политики защиты: защита информации только тогда является оправданной и разумной, когда стоимость реализации политики безопасности, по крайней мере, не меньше

стоимости потерь, вызванных несанкционированным ее использованием.

Организация Microsoft Security Response Center сформулировала наиболее важные *положения* по компьютерной безопасности, которые выглядят следующим образом (приводятся здесь без комментариев):

1. Если злоумышленник убедит вас в том, что его программа должна выполняться на вашем компьютере, этот компьютер перестанет принадлежать вам.
2. Если злоумышленник сможет изменить операционную систему на вашем компьютере, этот компьютер перестанет вам принадлежать.
3. Если хакер (или кракер) имеет физический доступ к вашему компьютеру, этот компьютер перестанет вам принадлежать.
4. Если злоумышленник загрузит свои программы на ваш веб-сайт, этот сайт перестанет вам принадлежать.
5. Использование простых паролей ослабляет строгие меры безопасности.
6. Компьютер безопасен только в том случае, когда администратор (безопасности) добросовестно относится к своим обязанностям.
7. Шифрование данных является безопасным в той мере, в какой соблюдены правила безопасного хранения ключа.
8. Устаревший сканер вирусов лишь немногим лучше отсутствующего сканера вирусов.
9. Абсолютная анонимность лишена практического смысла как в реальной жизни, так и при работе в Интернете.
10. Любая технология не является панацеей от всех бед.

Вопросы для самоконтроля

1. Дайте определения основных понятий и терминов из области информационной безопасности и надежности систем.
2. Приведите соотношения для расчета вероятности безотказной работы ИВС, вероятности отказа ИВС и интенсивности отказов в ИВС.
3. Рассчитайте вероятность безотказной работы и вероятность отказа системы, структура которой приведена на рис. 1.2 (при заданных преподавателем интенсивностях отказов в каждом из модулей).
4. Вычислите вероятность безотказной работы и вероятность отказа системы, структура которой приведена на рис. 1.3 (при

заданных преподавателем интенсивностях отказов в каждом из модулей).

5. Охарактеризуйте надежность программного обеспечения.

6. Проанализируйте угрозы для ИВС.

7. Охарактеризуйте методы защиты ИВС от несанкционированного доступа.

Задание для самостоятельного выполнения

Создайте программное средство, с помощью которого произведите сравнительную оценку надежности ИВС на рис. 1.2 и 1.3 при заданных преподавателем условиях.

2. ИНФОРМАЦИЯ, ЕЕ КОЛИЧЕСТВО И КАНАЛЫ ПЕРЕДАЧИ

2.1. Характеристики системы передачи информации

Передача информации (данных) осуществляется между двумя абонентами, называемыми ***источником сообщения*** (ИС) и ***получателем сообщения*** (ПС). Источником и получателем могут быть люди либо технические средства. ИС и ПС обмениваются информацией посредством канала передачи. Таким образом, простейшая информационная система состоит из трех перечисленных элементов. Ее обобщенная структурная схема приведена на рис. 2.1.

Рис. 2.1. Обобщенная структурная схема информационной системы

В современных информационных системах качество передачи достигается применением трех базовых *методов* преобразования информации:

1) помехоустойчивого кодирования;

2) сжатия (компрессии) данных;

3) криптографического преобразования.

Любой из этих методов (или в комбинации с другими) используется для решения конкретных задач, более подробный анализ которых будет дан в следующих разделах. Здесь отметим, что при использовании указанных методов преобразования на входе и на выходе канала передачи появятся дополнительные блоки.

ИС и ПС обмениваются информацией в технических системах в виде сигналов, сформированных на основе определенного *алфавита*. Характеристикой алфавита является его мощность N_{a} – количество символов, на основе которых формируется сообщение. Например, мощность английского алфавита – 26 символов, русского – 33 символа, мощность алфавита, на основе которого функционируют и взаимодействуют между собой компьютеры, составляет 2 символа (0 и 1).

Другой важной характеристикой источника (и получателя) сообщений является тип используемого сигнала. Существует два основных *типа* сигналов: 1) аналоговый, 2) цифровой. Аналоговый сигнал, в отличие от цифрового, характеризуется бесконечным числом состояний (значений). ИС на основе аналогового сигнала называется *источником непрерывных сообщений*. ИС на основе цифрового сигнала называется *источником дискретных сообщений*. Примером последнего может служить источник двоичных (бинарных) сообщений.

Далее будем рассматривать только ИС и ПС дискретных сообщений. Наибольший интерес представляет собой цифровой сигнал на основе алфавита $A\{0,1\}$.

В произвольном сообщении символы алфавита могут появляться с различной вероятностью. Если длина сообщения достаточно велика, то статистический анализ этого сообщения позволит получить вероятностные характеристики этого алфавита. Очевидно, что различные символы в произвольном сообщении (особенно при $N_{\text{a}} > 2$) появляются с различной вероятностью, т. е. существуют символы с минимальной и максимальной вероятностью появления.

Например, подсчитано, что наиболее часто (в 13% случаев) в документах на английском языке ($N_{\text{a}} = 26$) появляется буква «e», а наиболее редко (в 0,7% случаев) – буквы «x», «y» и «z». Можем записать, что вероятность того, что произвольный символ ξ произвольного документа (текст, база данных, текст программы) будет буквой «e» (или другой из указанных букв), как

$$P(\xi = \text{e}) = 0.13,$$

$$P(\xi = \text{x}) = P(\xi = \text{y}) = P(\xi = \text{z}) = 0.007.$$

Информационной характеристикой алфавита (источника сообщений на основе этого алфавита) является *энтропия*. Этот термин применительно к техническим системам был введен Шенноном и Хартли.

Энтропию алфавита $A\{a_i\}$ по Шеннону рассчитывают по следующей формуле:

$$H_S(A) = -\sum_{i=1}^{N} P(a_i) \cdot \log_2 P(a_i), \qquad (2.1)$$

где $i = \overline{1, N}$; a_i – элемент алфавита; $P(a_i)$ – вероятность $P(\xi = a_i)$.

Заметим, что $\sum_{i=1}^{N} P(a_i) = 1$.

С физической точки зрения *энтропия показывает, какое количество информации приходится в среднем на один символ алфавита*.

Частным случаем энтропии Шеннона является энтропия Хартли. Дополнительным условием при этом является то, что все вероятности одинаковы и постоянны для всех символов алфавита. С учетом этого формулу (2.1) можно преобразовать к следующему виду:

$$H_{Ch}(A) = \log_2 N.$$

Например, энтропия Хартли для латинского (английского) алфавита составляет 4,7 бит.

Если подсчитать энтропию Шеннона и энтропию Хартли для одного и того же алфавита, то они окажутся не равными. Это несовпадение указывает на избыточность любого алфавита (при $N > 2$).

Сообщение M, которое состоит из n символов, должно характеризоваться определенным *количеством информации* $I(M)$:

$$I(M) = H(A) \cdot n. \qquad (2.2)$$

Нетрудно предположить и просто убедиться, что количество информации в сообщении, подсчитанное по Шеннону, не равно количеству информации, подсчитанному по Хартли. На основе этого парадокса строятся и функционируют все современные системы сжатия (компрессии) информации.

Пример фрагмента кода программы, позволяющей вычислить энтропию английского алфавита, приведен ниже (листинг).

```
//подключение необходимых для работы программы библиотек
#include <iostream.h>
#include <fstream.h>
#include <stdio.h>
#include <math.h>
#include <string.h>
#include <ctype.h>
int main()            //тело главной функции
{char ch;             //переменная, которая будет
                      //хранить считываемый из файла
                      //символ
int count[26];        //массив count[], элементы
                      //которого будут содержать
                      //количество появлений в
                      //информационном сообщении
                      //некоторого символа
double p[26];         //массив вероятностей появления
                      //символов алфавита
int t;                //переменная, которая будет
                      //хранить общее количество
                      //символов файла (текстового
                      //сообщения), включая символ
                      //конца строки
t=0;
char alpha[]={'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'};
for (int i=0;i<26;i++)
{count[i]=0;}
                      //инициализация элементов
                      //массива count[] нулями
ifstream in ("b");    //открытие файла для чтения
if (!in) {
cout<<"Cannot \n";return 1;
                      //проверка возможности открытия
                      //файла
}
while(in){            //проверка достижения
                      //конца файла
in.get(ch);           //считывание очередного символа
```

```
                        //из файла
t++;                    //увеличение счетчика общего
                        //количества символов файла
                        //(включая пробел) на 1
for (int i=0; i<26; i++)
{                       //приведение всех символов
                        //файла к нижнему регистру и
                        //проверка вновь считанного
                        //символа на соответствие
                        //символам исходного алфавита
if (putchar(tolower(ch))==alpha[i])
                        //увеличение счетчика появления
                        //символа на 1
count[i]=count[i] + 1;
for (int i=0;i<25;i++)
{
double p[i]=count[i]/(t - 1);
                        //массив p[] содержит значения
                        //появления символов алфавита
                        //установки на вывод числа с точкой
cout.setf(ios::showpoint);
cout.setf(ios::fixed);
cout.precision(2);
cout<<alpha[i]<<"* - *";
cout<<count[i]<<"* - *"<<p[i]<<"\n";
}
double h;
h=0;
for (int i=0;i<26;i++)
{double result;
if (p[i]!=0)            //условие вычисления логарифма
{result=log(p[i]);
                        //вычисление     энтропии     Шеннона
                        //(переменная h)
h=h + (p[i]*result);
}
}
h=-h;                   //взятие энтропии с
                        //противоположным знаком
                        //и ее вывод
cout<<"h="<<h<<"\n";
int V=0;
V=h*(t - 1);            //вычисление количества информации
cout<<"I="<<V<<"\n";
                        //вывод      значения      количества
                        //информации
in.close();             //закрытие файла
return 0;
```

Листинг. Пример фрагмента кода программы в C++,
позволяющей вычислить энтропию английского алфавита

2.2. Двоичный канал передачи информации

Как было подчеркнуто выше, двоичный канал передачи информации является дискретным. Он основан на алфавите, состоящем из двух символов (0 и 1) – $A\{0,1\}$. Используя (2.1), вычислим энтропию этого алфавита:

$$H(A_2) = -P(0) \cdot \log_2(P(0) - P(1) \cdot \log_2(P(1)). \qquad (2.3)$$

К примеру, полагая, что сообщение M состоит только из единиц ($M = 11...1$) и имеет длину n: $M = \underbrace{11\text{K}1}_{n}$, т. е. вероятность того, что произвольный символ равен единице, составляет единицу ($P(1) = 1$), тогда $P(0) = 0$ для $i = \overline{1, N}$. Фактически здесь имеет место использование моноалфавита – алфавита, состоящего из одного символа.

Если в этом случае подставить в (2.3) соответствующие значения, то получим, что энтропия моноалфавита равна 0 бит, количество информации в сообщении из единиц (либо из нулей) также составляет 0 бит. Этот практический результат поясняет физический смысл понятия информации в теории Шеннона: *информацией является лишь такое сообщение, которое снимает некоторую неопределенность, т. е. содержит новые для получателя данные*. Если априори известно, что сообщение будет состоять из набора одинаковых символов, то для получателя сообщения оно никакой неопределенности не содержит.

Если для бинарного алфавита вероятность появления в произвольном сообщении одного из этих символов стремится к нулю (или равна ему), то энтропия такого алфавита также будет стремиться к нулю (или равняться ему).

Между этими точками значение функции (энтропии) должно пройти через максимум (так как энтропия не может быть отрицательной). Для нахождения этого максимального значения надо найти производную:

$$\frac{\partial H(A)}{\partial P(1) \cdot \partial P(0)} \Rightarrow \max H(A_2) = \begin{cases} P(0) = \dfrac{1}{2} \\ P(1) = \dfrac{1}{2} \end{cases}$$

$$\max H(A_2) = -\frac{1}{2} \cdot \log_2 \frac{1}{2} - \frac{1}{2} \cdot \log_2 \frac{1}{2} = 1 \text{ бит}.$$

Таким образом, энтропия бинарного алфавита принимает максимальное значение, равное 1 бит, при условии равновероятного появления каждого символа алфавита в сообщении (рис. 2.2).

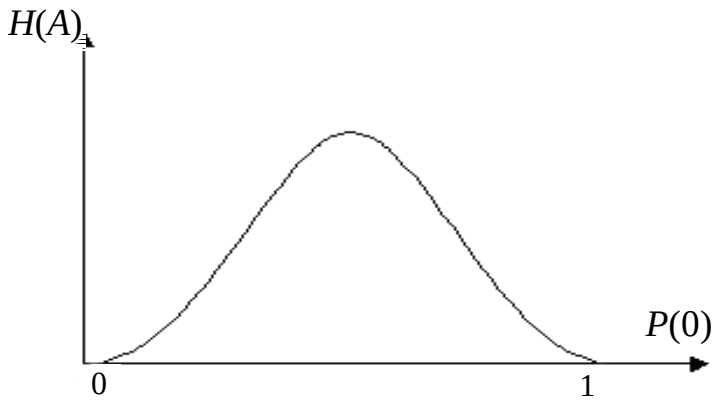

Рис. 2.2. Качественная характеристика энтропии бинарного алфавита при различных значениях вероятностей появления символов в сообщении

Интересным представляется оценка количества информации в сообщении, выполненная на основе различных подходов.

Пример. Анализируем сообщение $M =$«We are happy» (пробелы не учитываем). Если принять, что энтропия английского алфавита, вычисленная по Шеннону, составляет 4,2 бит (примерно), то в анализируемом сообщении содержится 42 бита информации: $I(M) = 4.2 \cdot 10 = 42$ бита.

С другой стороны, предполагая, что сообщение переведено в ASCII коды (один символ алфавита заменяется соответствующим байтом двоичных символов), и допустив, что $P(0) = P(1) = 0.5$, получаем $I(M_{\text{ASCII}}) = 1 \cdot 80 = 80$ бит.

Данный пример является свидетельством и подтверждением избыточности не только алфавита, но и сообщений, сформированных и обрабатываемых в компьютерных системах, т. е. любые сообщения характеризуются информационной избыточностью, что позволяет сжимать их без потери информации.

Вопросы для самоконтроля

1. Дайте определения мощности и энтропии алфавита.

2. Как рассчитываются энтропия Шеннона и энтропия Хартли? В чем принципиальное различие между этими характеристиками? Дайте толкование физического смысла энтропии.

3. Изобразите обобщенную структурную схему системы передачи информации. Поясните назначение и особенности элементов системы.

4. Что такое избыточность алфавита и избыточность сообщений, сформированных в компьютерных системах? Принцип действия каких систем основан на существовании данной избыточности?

5. Расположите в порядке возрастания энтропии известные вам алфавиты.

Задание для самостоятельного выполнения

Создайте программное средство, с помощью которого произведите вычисление энтропии заданного алфавита, и определите количество информации, содержащейся в вашей фамилии, имени и отчестве.

Примечание.

Файл выходных данных (*result.txt*) должен содержать таблицу вероятностей появления символов алфавита, значения энтропии заданного и двоичного алфавитов, количество информации для исходного и преобразованного файлов.

Символами русского алфавита считать символы а(А)…я(Я), английского – a(A)…z(Z).

Все символы алфавита должны быть приведены к одному регистру (прописные/строчные). Следует отметить, что символами алфавита не являются служебные символы и цифры, но при вычислении количества информации в сообщении эти символы следует включить в общее количество символов сообщения. Программа может быть построена на основе листинга, приведенного в подразделе 2.1.

Для перевода информационного сообщения в двоичный код можно использовать код ASCII каждого символа, к которому следует применить функцию *itoa()*.

Функция *itoa()* конвертирует целое число *num* в строчный эквивалент и помещает результат в строку, на которую указывает параметр *str*. Основание системы счисления для записи выходной строки определено параметром *radix*, который может принимать значения в интервале от 2 до 36. Функция *itoa()* возвращает указатель на *str*. Прототип функции *itoa()* содержится в файле *stdlib.h*.

Синтаксис функции: *char *itoa(int num, char *str, int radix).* Например, результат работы функции *itoa(num,buf,2)* будет сохранен в переменной *buf =”11”*.

3. МЕТОДЫ ПОМЕХОУСТОЙЧИВОГО КОДИРОВАНИЯ ДАННЫХ

3.1. Цели и задачи кодирования. Основные понятия

Методы помехоустойчивого кодирования относятся к классу избыточных и призваны обеспечить повышение надежности передачи (или хранения) информации. Сущность этих методов проанализирована нами ранее в подразделе 1.2. Вспомним, что

повышение надежности достигается *аппаратно-временной* и *информационной избыточностью*, вводимой в исходную систему или устройство для обнаружения и исправления (либо только для обнаружения) ошибок, возникающих при передаче или хранении данных. Определение рассматриваемых методов как «помехоустойчивые» означает, прежде всего, что они являются противодействием помехам, влияющим на систему и приводящим к ошибкам в данных.

Суть метода состоит в преобразовании исходного информационного сообщения X_k (k – длина сообщения), называемого также *информационным словом*. К слову X_k дополнительно присоединяют (наиболее часто – по принципу конкатенации) избыточные символы длиной r бит, составляющие *избыточное слово* X_r. Таким образом, формируют *кодовое слово* X_n длиной $n = k + r$ двоичных символов: $X_n = X_k X_r$. Информацию содержит только информационное слово. Назначение избыточности X_r – обнаружение и исправление ошибок.

В зависимости от принципа вычисления дополнительных символов и их числа реализуются различные алгоритмы помехоустойчивого кодирования. Общим является то, что избыточное слово X_r генерируется на передающей стороне и используется принимающей для обнаружения и исправления ошибок. С учетом избыточных блоков обобщенная структурная схема системы передачи информации, представленная на рис. 2.1, примет вид, показанный на рис. 3.1.

Для дальнейшего рассмотрения нам необходимо упомянуть о некоторых базовых понятиях, относящихся к предметной области.

Вес по Хеммингу произвольного двоичного слова X ($w(X)$) равен количеству ненулевых символов в слове.

Пример 3.1. $X = 1101$. Тогда $w(X = 1101) = 3$.

Расстояние по Хеммингу или кодовое расстояние (d) между двумя произвольными двоичными словами (X, Y) одинаковой длины равно количеству позиций, в которых X и Y отличаются между собой.

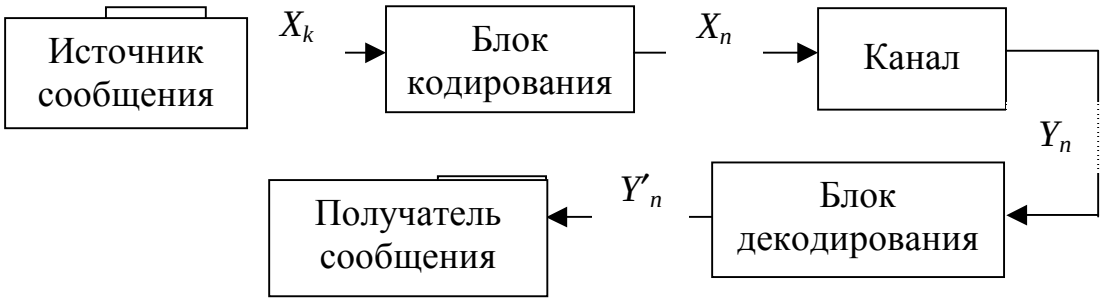

Рис. 3.1. Обобщенная структурная схема системы передачи информации с помехоустойчивым кодированием

Пример 3.2. $X = 101$, $Y = 111$. Очевидно, что $d(X, Y) = 1$.

Кодовое расстояние можно вычислить как вес от суммы по модулю 2 этих двух слов: $d(X, Y) = w(X \oplus Y)$.

Пример 3.3. $X = 1011$, $Y = 0000$; $d(X, Y) = 3$:

$$\begin{array}{r} 1011 \\ 0000 \\ \hline w(1011) = 3 \end{array}.$$

Пример 3.4. $X = 11111$, $Y = 11111$; $d(X, Y) = 0$.

Длина слова и расстояние Хемминга – основополагающие понятия в теории помехоустойчивого кодирования информации.

Все многообразие существующих кодов для обнаружения и исправления ошибок можно разделить на два больших класса: линейные и нелинейные коды. Коды первого класса базируются на использовании линейных (как правило, умножение и сложение по модулю 2 соответствующих символов) операций над данными, коды второго класса – соответственно нелинейных операций.

В информационных системах нашли практическое применение коды, принадлежащие двум семействам. К таким системам (из числа известных читателю) следует отнести следующие: системы полупроводниковой, магнитной и оптической памяти; системы связи, в том числе сотовой; системы передачи данных по сетям (например, частью протокола TCP является CRC-32 (CRC-16); блок, отвечающий за обнаружение и исправление ошибок – check redundancy code, CRC) и т. д. В дальнейшей части настоящего раздела рассмотрим некоторые из числа простых и эффективных линейных блочных кодов.

3.2. Теоретические основы линейных блочных кодов

Линейные блочные коды – это класс кодов с контролем четности, которые можно описать парой чисел (n, k).

Первое из чисел определяет длину кодового слова X_n, второе – длину информационного слова X_k. Отношение числа бит данных к общему числу бит данных k/n именуется ***степенью кодирования*** (code rate) – доля кода, которая приходится на полезную информацию.

Для формирования проверочных символов (кодирования) используется порождающая матрица. Совокупность базисных векторов будем далее записывать в виде матрицы G размерностью $k \times n$ с единичной подматрицей I в первых k строках и столбцах:

$$G = [P \mid I]. \tag{3.1}$$

Матрица G называется ***порождающей*** матрицей линейного корректирующего кода в приведенно-ступенчатой форме. Кодовые слова являются линейными комбинациями строк матрицы G (кроме слова, состоящего из нулевых символов). Кодирование, результатом которого является кодовое слово X_n, заключается в умножении вектора сообщения длиной k (X_k) на порождающую матрицу по правилам матричного умножения (все операции выполняются по модулю 2). Очевидно, что при этом первые k символы кодового слова равны соответствующим символам сообщения, а последние r символов (X_r) образуются как *линейные* комбинации первых.

Для всякой порождающей матрицы G существует матрица H размерности $r \times n$, задающая базис нулевого пространства кода и удовлетворяющая равенству:

$$G \cdot H^T = 0. \tag{3.2}$$

Матрица H, называемая ***проверочной***, может быть представлена так:

$$H = \left[-P^T \mid I \right]. \tag{3.3}$$

В последнем выражении I – единичная матрица порядка r.

Кодовое слово X_n может быть получено на основе следующего тождества:

$$H \cdot (X_n)^T = 0. \tag{3.4}$$

Результат умножения сообщения Y_n на *транспонированную* проверочную матрицу H называется *синдромом* (вектором ошибки) S:

$$S = (Y_n)^T \cdot H, \tag{3.5}$$

где $Y_n = y_1, y_2, \ldots, y_n$. Слово Y_n обычно представляют в следующем виде:

$$Y_n = X_n \oplus E, \tag{3.6}$$

где $E = e_1, e_2, \ldots, e_n$ – вектор ошибки.

Если все r символов синдрома нулевые ($S = 0$), то принимается решение об отсутствии ошибок в принятом сообщении, в противном случае – об их наличии.

В общем случае код, характеризующийся *минимальным кодовым расстоянием* $d_{\min}$ между двумя произвольными кодовыми словами, позволяет обнаруживать t_0 ошибок, где $t_0 = \frac{d}{2}$, если d – четно, и $t_0 = \frac{d-1}{2}$, если d – нечетно. Количество исправляемых кодом ошибок $t_и$ определяется следующим образом:

$$t_и = \begin{cases} \frac{d-1}{2}, & d - \text{нечетное}, \\ \frac{d-2}{2}, & d - \text{четное}. \end{cases} \tag{3.7}$$

Избыточный код простой четности. Простейший избыточный код основан на контроле четности (либо нечетности) единичных символов в сообщении. Количество избыточных символов r всегда равно 1 и не зависит от k. Значение этого символа будет нулевым, если сумма всех символов кодового слова по модулю 2 равна нулю.

Назначение X_r в данном алгоритме – обнаружение ошибки. Код простой четности позволяет *обнаруживать все нечетные ошибки* (нечетное число ошибок), но *не позволяет их исправить*. Нетрудно убедиться, что данный код характеризуется минимальным кодовым расстоянием, равным 2.

Пример 3.5. Пусть информационная последовательность будет X_k = 10101, тогда $X_r = \sum_i^n X_i = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1.$

Проверочная матрица для данного случая будет состоять из одной строки и шести столбцов и примет следующий вид:

H = 11111 1.

Кодовое слово X_n, вычисленное в соответствии с (3.4), будет равно 10101 1. Как видим, $w(X_n)$ имеет четное значение.

Слово X_n будет передаваться от ИС к ПС. Пусть на приемной стороне имеем Y_n = 1<u>1</u>101 1: Y_k = 1<u>1</u>101 и Y_r = 1 (ошибочный символ подчеркнут).

Для определения синдрома ошибки в соответствии с (3.5) достаточно выполнить следующие простые действия:

а) вычислить дополнительное слово (в данном случае – символ) Y_r', которое является сверткой по модулю 2 слова Y_k: $Y_r' = 1 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 0$;

б) найти синдром $S = Y_r \oplus Y_r' = 1 \oplus 0 = 1$. Неравенство синдрома нулю означает, что получено сообщение с ошибкой (или с ошибками).

3.3. Код Хемминга

Данный код характеризуется минимальным кодовым расстоянием $d_{\min} = 3$. При его использовании кодирование сообщения также должно удовлетворять соотношению (3.4). Причем *вес столбцов подматрицы A должен быть больше либо равен 2*. Второй особенностью данного кода является то, что используется *расширенный контроль четности групп символов информационного слова*, т. е. $r > 1$. Для упрощенного вычисления r можно воспользоваться следующим простым соотношением:

$$r = \log_2 k + 1. \tag{3.8}$$

В сравнении с предыдущим кодом данный позволяет не только обнаруживать, но и исправлять одиночную ошибку в кодовом слове (см. (3.7)).

В нашем случае подматрицу A можно определить как

$$A = \begin{bmatrix} h_{11} & h_{12} & \text{K} & h_{1k} \\ h_{21} & \text{M} & \text{M} & \text{M} \\ \text{M} & \text{M} & \text{M} & \text{M} \\ h_{r1} & \text{K} & \text{K} & h_{rk} \end{bmatrix}. \tag{3.9}$$

Элемент этой подматрицы (0 или 1) h_{ij} относится к i-й строке и j-му столбцу ($i = \overline{1,r}, \quad j = \overline{1,k}$).

Вычислим проверочные символы в соответствии с (3.4):

$$x_{ri} = \sum_{ij}^{rk} h_{ij} \cdot x_k \bmod 2. \tag{3.10}$$

Определим синдром:

$$y_{ri} = \sum_{ij}^{rk} h_{ij} \cdot y_k \bmod 2, \quad S = y_{ri} \oplus y_{ri}'. \tag{3.11}$$

Пример 3.6. Имеется информационное слово $X_k = 1001$. Проанализируем использование рассматриваемого кода.

Для начала отмечаем, что $k = 4$. В соответствии с (3.8) подсчитываем длину избыточного слова: $r \geq \log_2(4 + 1) = 3$, тогда $n = k + r = 7$.

Создаем проверочную матрицу $H_{7,4}$:

$$H_{7,4} = \left[\begin{array}{ccc|cccc} 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{array}\right].$$

$n \cdot k$

A I

Вычисляем проверочные символы, используя (3.10).

В соответствии с этим первый проверочный символ x_{r1} будет равен 1, остальные – нулю:

$$x_{r1} = h_{11} \cdot x_1 \oplus h_{12} \cdot x_2 \oplus \ldots \oplus h_{14} \cdot x_4 = 0 \cdot 1 \oplus 1 \cdot 0 \oplus 1 \cdot 0 \oplus 1 \cdot 1 = 1,$$

$$x_{r2} = h_{21} \cdot x_1 \oplus h_{22} \cdot x_2 \oplus \ldots \oplus h_{24} \cdot x_4 = 1 \cdot 1 \oplus 0 \cdot 0 \oplus 1 \cdot 0 \oplus 1 \cdot 1 = 0,$$

$$x_{r3} = h_{31} \cdot x_1 \oplus h_{32} \cdot x_2 \oplus \ldots \oplus h_{34} \cdot x_4 = 1 \cdot 1 \oplus 1 \cdot 0 \oplus 0 \cdot 0 \oplus 1 \cdot 1 = 0.$$

Таким образом, избыточное слово будет таким: $X_r = 100$, а кодовое слово – $X_n = 1001\ 100$.

Рассмотрим ситуацию, когда ошибок в переданной информации нет ($t = 0$) т. е. $X_n = Y_n = 1001\ 100$.

Вычислим новый набор проверочных символов в соответствии с (3.11) и синдром:

$$Y_r' = 100,$$

$$S = Y_r \oplus Y_r' = 100 \oplus 100 = 000 \equiv 0.$$

Нулевой синдром означает безошибочную передачу (или прием) информации.

Рассмотрим ситуацию, когда возникает одиночная ошибка ($t = 1$).

Пусть ошибка произошла в служебных символах $Y_n = 1001\underline{0}00$ (ошибочный символ подчеркнут).

Синдром вычисляем по методике, приведенной для случая отсутствия ошибок. Получаем $S = 100$. Вес синдрома равен 1 и это означает, что произошла ошибка. Местоположение ошибки выявляется анализом (декодированием) синдрома. Декодирование опирается на вышеприведенное соотношение (3.5), в соответствии с которым, принимая во внимание (3.6), можем записать:

$$H \cdot (Y_n)^T = H \cdot (X_n \oplus E)^T = H \cdot (X_n)^T \oplus H \cdot (E)^T = 0 \oplus h_5, \qquad (3.12)$$

где h_5 – пятый столбец матрицы, номер которого соответствует номеру ошибочного символа в принятом кодовом слове. Действительно, $h_5 = S = 100$.

В результате декодирования синдрома получается вектор ошибки (*унарный вектор*, имеющий единичный вес): $E = 0000100$. Исправление ошибочного бита достигается простым сложением по модулю 2 вектора E и кодового слова Y_n:

$$Y_n = E \oplus Y_n = 0000100 \oplus 1001000 = 1001100.$$

Пусть ошибка произошла в бите информационного слова $Y_n = \underline{0}001100$.

Вычислим дополнительные проверочные символы и синдром:

$$Y_r' = 111,$$

$$S = Y_r \oplus Y_r' = 011.$$

Убеждаемся, что синдром соответствует первому столбцу используемой проверочной матрицы. Это означает, что декодирование синдрома однозначно укажет на местоположение ошибочного бита:

$E = 1000000$ и $Y_n' = E \oplus Y_n = 0001100 \oplus 1000000 = 1001100 \equiv X_n$.

При возникновении ошибок кратности два (например, на позициях l и m) данный код не позволяет *однозначно* идентифицировать ошибки, поскольку с учетом (3.12) имеем:

$$S = h_1 \oplus h_m. \qquad (3.13)$$

Таким образом, код Хемминга с $d_{\min} = 3$ *гарантированно обнаруживает и исправляет одиночную ошибку* в любом разряде кодового слова.

Порядок следования вектор-столбцов в матрице A не имеет значения, однако важно, чтобы на передающей и на принимающей сторонах используемые матрицы были бы абсолютно идентичны.

3.4. Модифицированный код Хемминга

Этот код характеризуется минимальным кодовым расстоянием, равным 4, и позволяет *обнаруживать две ошибки и исправлять одну*.

Алгоритм использования кода такой же, как и для $d_{\min} = 3$. Принципиальное отличие состоит в виде проверочной матрицы, которая при неизменной длине информационного слова имеет одну дополнительную строку и один дополнительный столбец (в подматрице I):

$$H' = \left[\begin{array}{ccccc|c} & & & & & 0 \\ & & & & & 0 \\ & & H & & & \text{M} \\ & & & & & 0 \\ \hline 1 & 1 & \text{L} & 1 & 1 & 1 \end{array}\right]. \qquad (3.14)$$

Запись матрицы в таком виде не считается канонической, так как подматрица I не является единичной диагональной матрицей. Для преобразования такой записи к каноническому виду воспользуемся свойствами линейного кода: в дополнительную строку необходимо записать сумму по модулю 2 соответствующих символов матрицы кода с $d_{\min} = 3$.

Пример 3.7. Пусть $k = 6$ для случая $d_{\min} = 3$ и $r = \log_2 k + 1 = 4$, $n = 10$.

При этих параметрах матрица $H_{10,6}$ может иметь следующий вид:

$$H_{10,6} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Стоит задача преобразования такой матрицы для случая $d_{\min} = 4$. Дополним ее строкой, состоящей из единиц, и одним дополнительным столбцом:

$$H_{11,6} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

Для преобразования такой матрицы к каноническому виду, используя свойство линейности кода, надо сложить по модулю 2

символы соответствующих столбцов последней матрицы и записать полученный результат вместо последней строки. Тогда имеем:

$$H_{11,6} = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

В общем случае любая проверочная матрица кода Хемминга с $d_{\min} = 4$ имеет нечетный вес столбцов, т. е. вес любого из столбцов подматрицы $A`$ может быть равен 3, 5, 7, … .

Как и в предыдущем случае (при $d_{\min} = 3$), *равенство нулю синдрома означает отсутствие ошибок*. Если же *синдром не равен нулю* и *имеет нечетный вес*, то это говорит о том, что произошла *одиночная ошибка*. Если же *синдром не равен нулю* и его *вес четный*, то произошла *двойная ошибка*, *так как вес суммы любых двух столбцов всегда четный* (см. (3.12)).

3.4.1. Программная реализация метода кодирования кодом Хемминга. Наиболее трудоемкой является процедура генерации проверочной матрицы. Существует несколько способов, позволяющих заполнить матрицу Хемминга, один из которых заключается в следующем. Поскольку известно, что минимальный вес столбцов в матрице A равен 2, вычисляем значение бинома Ньютона по формуле

$$C_{wt}^{r} = \frac{r!}{wt \cdot (r - wt)!}, \qquad (3.15)$$

где $wt = 2$; $r = \log_2 k + 1$.

Бином позволяет подсчитать количество различных комбинаций из 0 и 1, которые мы можем получить при заданных значениях r и wt. Ниже (листинг 3.1) приведен синтаксис функций, которые дают возможность вычислить бином Ньютона.

```
 int fact (int n)          //функция вычисления
                           //факториала натурального
                           //числа n
{
int answer;                //переменная, которая будет
                           //хранить значение факториала
                           //натурального числа n
if (n==1)
```

```
return 1;                    //1 (1!=1)
                             //рекурсивное обращение функции
                             //fact() к самой себе
answer=fact(n - 1)*n;
return (answer);             //возвращение результата
                             //работы функции в место
                             //вызова ее в теле основной
                             //программы
}
                             //функция вычисления бинома
                             //Ньютона
int binom(int wt,int r) {
int C;                       //переменная, которая будет
                             //хранить значение бинома
                             //Ньютона
C=fact(r)/(fact(wt)*fact(r - wt));
     return (C);             //возвращение результата работы
                             //функции в место вызова ее в
                             //теле основной программы
}
```

Листинг 3.1. Синтаксис функций,
позволяющих вычислить бином Ньютона

Далее, переводя натуральные числа, начиная с 3 (первое натуральное число, которое при переводе в двоичную систему счисления имеет вес, равный 2), в двоичную систему счисления и дополняя полученную комбинацию нулями до r позиций (если это необходимо), по очереди заполняем столбцы матрицы A.

Пример 3.8. Положим, что $r = 5$, $wt = 2$.

Тогда

$$C_2^5 = \frac{5!}{2 \cdot (5-2)!} = 10,$$

$3_2 = 11$.

Первым столбцом матрицы A будет вектор [0 0 0 1 1].

дополнено в соответствии со значением r

$4_2 = 100 – wt(4_2) = 1$ – не подходит,
$5_2 = 101 – wt(5_2) = 2$ – подходит.

Следовательно, вторым столбцом матрицы A будет: 00101 и т. д. Программно это выглядит следующим образом (листинг 3.2).

```
itoa(num,buf,2);             //представление натурального
```

```
                              //числа в двоичной системе
                              //счисления
int len=strlen(buf);
                              //переменная len хранит длину
                              //полученной двоичной
                              //последовательности
sum=0;                        //переменная, используемая для
                              //вычисления веса полученной
                              //двоичной последовательности
                              //(двоичного представления
                              //натурального числа n)
for (int i=0;i<len;i++)
  {if (buf[i]=='1')
  sum++; }                    //при встрече в двоичном
                              //представлении числа символа
                              //'1' счетчик sum увеличивается
                              //на 1
 if (sum==wt)                 //если вес числа в двоичном
                              //представлении равен текущему
                              //весу (первоначально wt=2),
  {count++;                   //тогда число комбинаций,
                              //подходящих для заполнения
                              //столбцов матрицы A(k,r),
                              //увеличивается на 1
    if (len<r)                //если длина полученной
                              //двоичной последовательности
                              //меньше r, происходит
                              //увеличение ее на 0 (r -
                               длина последовательности)
      {for (int j=0;j<(r - len);j++)
      {
       char buf1[80]={'0'};
       strcat(buf1,buf);
       strcpy(buf,buf1);}
   }
  for (int i=0;i<r;i++){
                              //заполнение столбца матрицы
                              //A(k,r)
   A[i][count - 1]= buf[i];}
  num++;                      //переход к следующему
                              //натуральному числу
```

Листинг 3.2. Программная реализация
метода заполнения столбцов проверочной матрицы

В случае если все комбинации при заданных r и wt уже исчерпаны, увеличиваем вес wt столбцов матрицы A на единицу и опять вычисляем бином Ньютона.

В заполнении единичной матрицы I нет никаких сложностей. Для начала элементы матрицы инициализируются нулями. Если общее

количество столбцов матрицы H составляет $n = k + r$, а первый столбец матрицы I имеет индекс k (нумерация элементов матрицы H начинается с 0), тогда:

```
for (int i=k; i<n; i++)
  {for (int j=0;j<r;j++)
   { if (j=i - k) I[i,j]=1;
    else I[i,j]=0; }
    }
```

Далее при вычислении i-го символа слова X_r достаточно подсчитать количество пар (1; 1) в соответствующих позициях слова X_k и i-й строки матрицы A. Если количество таких пар четное, т. е. по модулю 2 равно 0, то i-й элемент слова $X_r[i]$ равен 0, а в обратном случае – 1.

Вычисление синдрома – это сложение по модулю 2 векторов (слов) Y_r и Y_r'. Пусть размерность этих векторов равна r (нумерацию элементов векторов примем с 0), тогда:

```
for (int i=0;i<r;i++)
  {if (Yr[i]<>Yr'[i])
   s[i]=1;
  else s[i]=0; }
```

Другими словами, i-й элемент синдрома S будет равен 0, если i-е элементы векторов Y_r и Y_r' равны между собой, и этот же элемент равен 1 – в противном случае.

Если анализ синдрома показал, что принятое слово содержит ошибку, ищем номер позиции в слове Y_n, в которой эта ошибка возникла. Зная позицию ошибки, можем сформировать унарный вектор ошибки E.

Пусть позицию ошибки хранит некоторая переменная *mist*. Первоначально необходимо проинициализировать все элементы вектора E нулями, далее организовать цикл:

```
for (int i=0;i<n;i++)
 {if (i<>mist) E[i]=0;
  else E[i]=1; }
```

3.4.2. Аппаратная реализация кодера и декодера. Внутренняя структура оборудования на передающей и на принимающей стороне определяется в основном используемыми проверочными матрицами A и I. Эти устройства могут быть построены на основе дискретных логических элементов, либо на

основе программируемых логических матриц. На передающей стороне *кодер* (блок кодирования) для $k = 6$ выглядит примерно так, как показано на рис. 3.2.

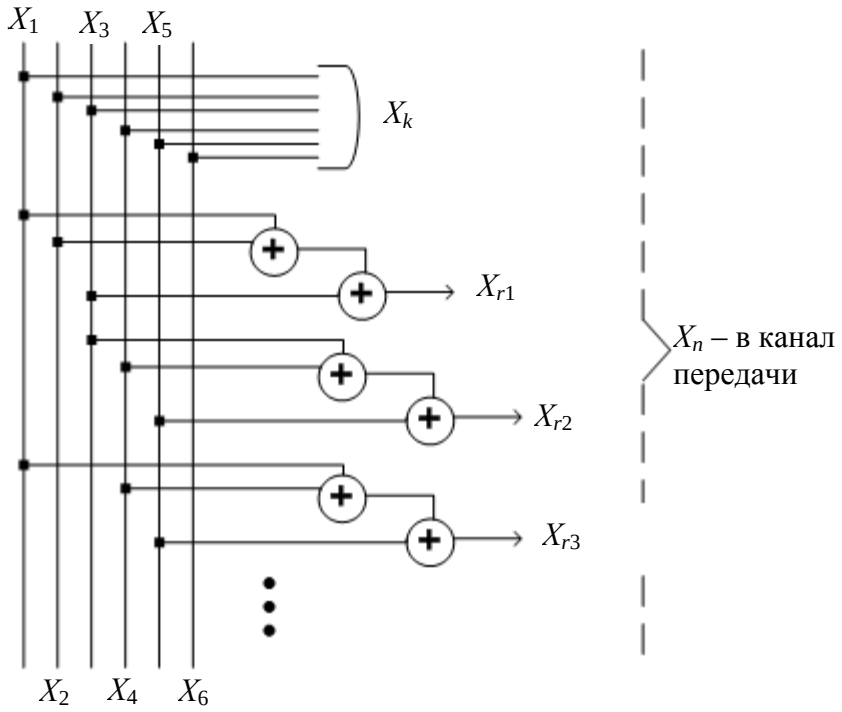

Рис. 3.2. Примерная внутренняя структура блока кодирования кодом Хемминга при $k = 6$ и $d_{\min} = 3$

На принимающей стороне схема декодера кода Хемминга выглядит аналогично рис. 3.2.

3.5. Итеративные коды

Итеративные коды, иногда называемые прямоугольными (rectangular code), либо композиционными (product code), являются одними из самых простых (с точки зрения аппаратной реализации) избыточных кодов, позволяющих исправлять ошибки в кодовых словах.

Простейшим из них является двумерный линейный итеративный код, при использовании которого кодовые слова записываются в виде таблицы. Основной является форма записи, при которой строки и столбцы содержат четное (нечетное) число единиц. Например, при кодировании информационного слова X_k = 011101111 с помощью таблицы с четностью по строкам и столбцам получим избыточные символы $X_r = X_h$, $X_v = 0010011$, как показано

на рис. 3.3 (информационные символы выделены жирным шрифтом, а проверочные – курсивом).

Кодовое слово имеет вид: X_n = 0111011110*010011*.

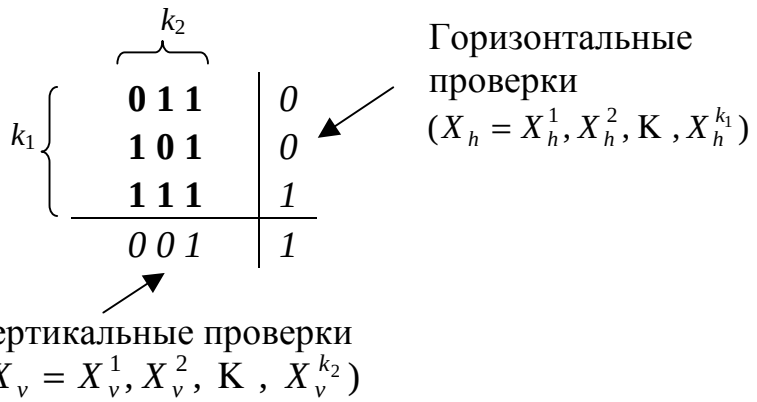

Рис. 3.3. Схематичное представление (в виде таблицы) двумерного линейного итеративного кода

Порождающая матрица для двумерного линейного итеративного кода (4, 3, 4, 3) с $d_{\min} = 4$, приведенного на рис. 3.3, будет иметь следующий вид:

$$G = [P \mid I] = \left[\begin{array}{c:c} 111000000 & 1000000 \\ 000111000 & 0100000 \\ 000000111 & 0010000 \\ 100100100 & 0001000 \\ 010010010 & 0000100 \\ 001001001 & 0000010 \\ 111111111 & 0000001 \end{array}\right]. \qquad (3.16)$$

Число проверочных символов определяется следующей зависимостью:

$$r = k_1 + k_2 + 1, \qquad (3.17)$$

где $k = k_1 \cdot k_2$.

Местоположение ошибки находится на пересечении строки и столбца с нарушенной четностью. Кодовое расстояние итеративного кода равно произведению минимальных расстояний составляющих его подкодов (свертки по модулю 2, $d = 2$). Следовательно, двумерный линейный итеративный код ($d_{\min} = 4$) позволяет корректировать все одиночные ошибки с одновременным обнаружением двукратных.

Увеличить корректирующие возможности двумерных итеративных кодов можно, дополнив проверочную матрицу избыточными символами (на четность) по диагонали, как показано на рис. 3.4 (информационные символы выделены жирным шрифтом, а проверочные – курсивом).

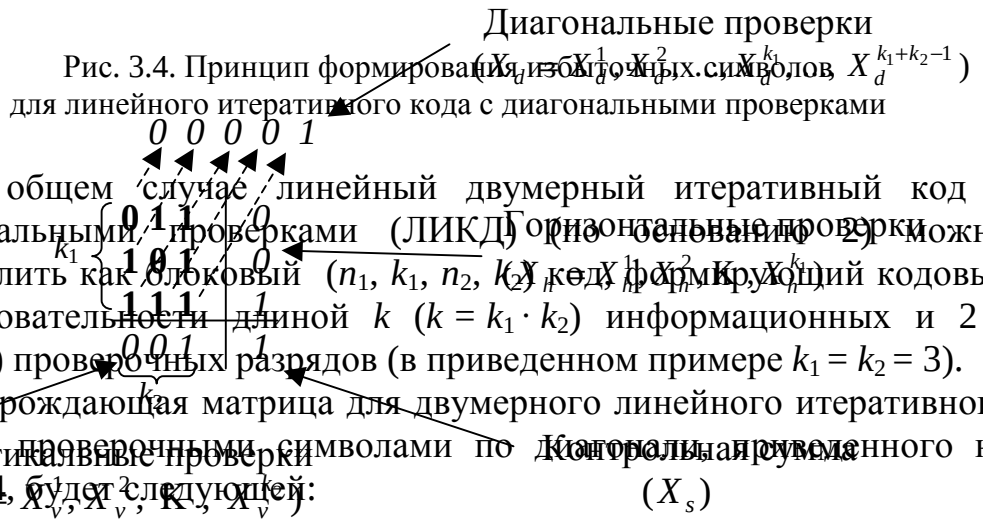

Рис. 3.4. Принцип формирования избыточных символов для линейного итеративного кода с диагональными проверками

В общем случае линейный двумерный итеративный код с диагональными проверками (ЛИКД) [по основанию 2] можно определить как блоковый (n_1, k_1, n_2, k_2) код, формирующий кодовые последовательности длиной k ($k = k_1 \cdot k_2$) информационных и $2 \cdot (k_1 + k_2)$ проверочных разрядов (в приведенном примере $k_1 = k_2 = 3$).

Порождающая матрица для двумерного линейного итеративного кода с проверочными символами по диагонали, приведенного на рис. 3.4, будет следующей:

$$G = [P \mid I] = \left[\begin{array}{c|c} 111000000 & 100000000000 \\ 000111000 & 010000000000 \\ 000000111 & 001000000000 \\ 100100100 & 000100000000 \\ 010010010 & 000010000000 \\ 001001001 & 000001000000 \\ 100000000 & 000000100000 \\ 010100000 & 000000010000 \\ 001010100 & 000000001000 \\ 000001010 & 000000000100 \\ 000000001 & 000000000010 \\ 111111111 & 000000000001 \end{array}\right]. \quad (3.18)$$

Рассмотрим некоторые основные параметры таких кодов.

Избыточность двумерного линейного итеративного кода r будет равна ($k = k_1 \cdot k_2$):

$$r = 2 \cdot (k_1 + k_2). \quad (3.19)$$

Минимальное кодовое расстояние двумерного линейного итеративного кода с диагональными проверками равно пяти ($d_{\min} = 5$). Следовательно, такой код позволяет обнаруживать и корректировать все ошибки, кратность которых не превышает двух.

Рассмотренный выше двумерный линейный итеративный код с диагональными проверками, прежде всего, может находить применение для коррекции ошибок в системах памяти. Для каналов передачи данных целесообразней использовать коды с максимальной кратностью обнаруживаемых ошибок. К таким относится двумерный линейный итеративный код с двойными диагональными проверками (ЛИКДД) (рис. 3.5).

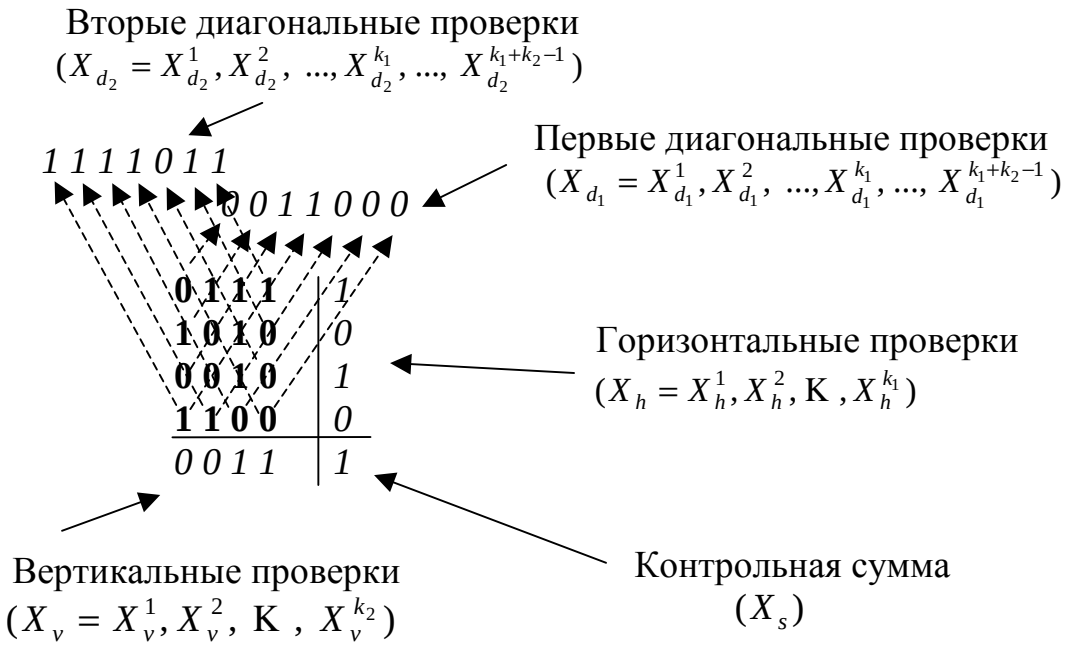

Рис. 3.5. Принцип формирования избыточных символов
для линейного итеративного кода с двойными диагональными проверками

Для определенного набора k двоичных информационных символов кодовое слово можно представить в виде k_1 n_2-разрядных кодовых слов строк, k_2 n_1-разрядных кодовых слов столбцов и $k_1 + k_2 - 1$ кодовых слов первых и вторых диагоналей, размерность которых не превышает меньшего из k_1 или k_2 ($n_1 = k_1 + 1$, $n_2 = k_2 + 1$).

В соответствии с этим, код, приведенный на рис. 3.5, можно представить в виде кодовых слов строк, столбцов и диагоналей (таблица).

Таблица

Кодовые слова строк, столбцов и диагоналей для ЛИКДД

Кодовые слова строк	Кодовые слова столбцов	Кодовые слова первых диагоналей	Кодовые слова вторых диагоналей
0111 *1*	**0101** *0*	000**0** *0*	000**1** *1*
1010 *0*	**1001** *0*	00**11** *0*	00**01** *1*
0010 *1*	**1110** *1*	0**001** *1*	0**100** *1*
1100 *0*	**1000** *1*	**1011** *1*	**0010** *1*
		0**110** *0*	0**110** *0*
		00**00** *0*	00**10** *1*
		000**0** *0*	000**1** *1*

Минимальное кодовое расстояние такого двумерного линейного итеративного кода равно шести ($d_{\min} = 6$), что дает возможность не только корректировать двойные ошибки, но и одновременно обнаруживать все тройные.

Однако известно, что код можно применять для одновременного исправления α и обнаружения β ошибок, причем $\alpha \le \beta$, а минимальное расстояние кода задается следующим выражением:

$$d = \alpha + \beta + 1. \qquad (3.20)$$

Поэтому итеративный код с $d_{\min} = 6$ при необходимости можно использовать в каналах передачи данных для обнаружения большего числа (вплоть до пяти) ошибок.

Оптимальными проверочными матрицами как с точки зрения быстродействия, так и минимума избыточности для всех видов итеративных кодов являются матрицы с одинаковым количеством строк и столбцов. При таких параметрах кода достигается и максимальная эффективность обнаружения и исправления ошибок высокой кратности. Для коррекции ошибок могут применяться различные алгоритмы декодирования: *синдромный* (как и для кодов Хемминга), *мажоритарный*, *с классификацией ошибок* и т. д.

Вопросы для самоконтроля

1. Дайте определения веса и кодового расстояния по Хеммингу. Приведите примеры.
2. Изобразите обобщенную структурную схему системы передачи информации с использованием помехоустойчивого кодирования данных. Поясните назначение и особенности элементов системы.
3. Как рассчитывается количество избыточных символов кодового слова? Приведите примеры.
4. Что такое проверочная матрица? Как она строится и каким условиям должна соответствовать? Приведите проверочную матрицу кода, если каждый символ исходной информации дублируется трижды (пять раз, шесть раз).
5. Постройте проверочную матрицу кода Хемминга и модифицированного кода Хемминга для $k = 2$ (3, 9, 11, 12).
6. Что такое итеративный код? Попробуйте изобразить проверочную матрицу для вычисления проверочных символов в соответствии с рис. 3.4.

Задание для самостоятельного выполнения

Создайте программное средство, с помощью которого следует произвести следующие операции:

1. На основе информационного сообщения, представленного символами русского/английского алфавитов, служебными символами и цифрами, содержащегося в некотором текстовом файле, сформируйте информационное сообщение в двоичном виде. Для выполнения этого задания применяйте коды ASCII.
2. Для полученного информационного слова постройте проверочную матрицу Хемминга (требуемое минимальное кодовое расстояние задает преподаватель).
3. Используя построенную матрицу, вычислите избыточные символы (слово X_r).
4. Проанализируйте ситуации, когда кодовое слово принимается без ошибок, с одной и с двумя ошибками. Позиция ошибки определяется случайным образом.
5. Для полученного слова, применяя известную проверочную матрицу Хемминга, вновь вычислите избыточные символы.
6. Определите и проанализируйте синдром: в случае, если анализ синдрома показал, что информационное сообщение было передано с одной ошибкой, сгенерируйте унарный вектор ошибки и исправьте ошибку.
7. Сделайте выводы.

Примечание.

Файл выходной информации должен содержать исходное информационное сообщение, значения величин k, r, n, проверочную матрицу Хемминга $H_{n,k}$, слова X_k, X_n, X_r, Y_n, Y_r, Y_r', синдром S, унарный вектор ошибки E. Программа не должна быть чувствительна к длине информационного сообщения.

4. ПРЕОБРАЗОВАНИЕ ИНФОРМАЦИИ НА ОСНОВЕ МЕТОДОВ КОМПРЕССИИ (СЖАТИЯ)

4.1. Цели, задачи и классификация методов

Методы компрессии информации преследуют три основные *цели*:

1) сокращение физического объема хранимой на носителях информации;
2) снижение стоимости передачи фиксированного объема информации (стоимости трафика);

3) повышение уровня конфиденциальности информации.

Основной особенностью методов сжатия является то, что количество передаваемых данных (после преобразования, т. е. после сжатия), как правило, меньше чем до преобразования: $n < k$. Важнейшая *задача* состоит в том, чтобы обеспечить реализацию отмеченного неравенства.

Существующие методы сжатия с точки зрения технологии их реализации можно условно разделить на три основных класса:

1) *символ-ориентированные* (или словарные) *методы* основаны на поиске и анализе повторяющихся символов (комбинаций) и их замене на другие комбинации меньшей длины;

2) *вероятностные методы* базируются на анализе вероятностных (частотных) характеристик используемого алфавита;

3) *комбинированные методы* объединяют первые и вторые, что может давать новые качественные свойства.

Второй классификационный признак основывается на соотнесении информации до компрессии (прямого преобразования: $X_k \rightarrow X_n$) и после декомпрессии (после обратного преобразования: $Y_n \rightarrow Y_k$; если между двумя отмеченными преобразованиями в X_n не возникли ошибки, то $X_n = Y_n$). В соответствии с этим методы сжатия можно разделить:

– на *сжатие без потерь информации*, при реализации которых объем и вид информации до сжатия и после декомпрессии полностью идентичны; такие методы должны применяться при сжатии текстовых документов, файлов баз данных, текстов программ, другой информации, где недопустимы потери;

– *сжатие с частичной потерей информации*, при которых исходная и декомпрессованная информации могут не совпадать (мультимедийные, звуковые и видеофайлы).

Основной численной характеристикой методов сжатия является коэффициент сжатия R. Указанный параметр рассчитывают двумя способами:

$$R_{\text{I}} = \frac{V_{\text{после\_сжатия}}}{V_{\text{до\_сжатия}}},$$

$$R_{\text{II}} = \frac{V_{\text{до\_сжатия}} - V_{\text{после\_сжатия}}}{V_{\text{до\_сжатия}}} = 1 - \frac{V_{\text{после\_сжатия}}}{V_{\text{до\_сжатия}}} = 1 - R_{\text{I}},$$

где $V_{\text{после\_сжатия}}$ − объем информации после сжатия; $V_{\text{до\_сжатия}}$ – объем информации до сжатия.

Пример 4.1. До компрессии объем файла ($V_{\text{до\_сжатия}}$) составлял 1 Мбайт, после сжатия ($V_{\text{после\_сжатия}}$) – 600 Кбайт. Коэффициент R_{I} = = 0,6 (60%), коэффициент R_{II} = 0,4 (40%).

Пример 4.2. Объем файла после преобразования не изменился. Очевидно, что при этом R_{I} = 1,0 (100%), а R_{II} = 0 (0%).

В силу очевидных причин на практике чаще используют второй способ вычисления.

4.2. Символ-ориентированные методы сжатия

4.2.1. Метод интервалов. Наиболее простым из анализируемой группы методов является метод интервалов. Суть метода заключается в поиске комбинаций одинаковых символов и замене их на специальную комбинацию *abc*, где *a* – специальный символ, *b* – один из символов анализируемой повторяющейся последовательности (интервала) и *c* – количество повторений. Как правило, принимается, что длина минимального интервала составляет два символа. При реализации метода сжатия важным является выбор специального символа (*a*). Его подбирают так, чтобы вероятность его появления в тексте стремилась к минимуму.

Пример 4.3. X_k = *mkkkcb0000fkff*. Можно выделить три интервала (подчеркнуты): X_k = *mkkkcb0000fkff.* Возьмем в качестве специального символ «*». После преобразования (сжатия) получим такую последовательность: X_n = *m*k3cb*04fk*f2*.

Как видно из примера, метод является эффективным при количестве повторений не ниже трех.

4.2.2. Метод Берроуза – Уиллера. Суть метода (Burrows – Wheller**)** состоит в том, что исходная информационная последовательность (X_k) разбивается на блоки произвольной, но одинаковой длины *k*. Преобразованию подвергается каждый блок в отдельности. На каждом этапе создается квадратная матрица W_1 размерности $k \times k$. Каждая последующая строка этой матрицы, начиная со второй, является циклическим сдвигом на один символ влево предыдущей строки. На основе первой матрицы создается вторая матрица (W_2) такой же размерности, содержащая строки первой матрицы, отсортированные в лексикографическом порядке.

Результатом преобразования является столбец h_k матрицы W_2 и число *N*, соответствующее номеру строки матрицы W_2, в которой записана исходная последовательность X_k.

Обратное преобразование основано на свойстве рекуррентности преобразования матриц W_1 и W_2. Задача состоит в воссоздании матрицы W_2. Эту процедуру начинаем с заполнения последнего столбца будущей матрицы. Дальнейшие действия иллюстрируются на основе используемого примера.

Пример 4.4. Пусть X_k = «*столб*». Как видим, $k = 5$. Строим, как показано ниже, соответствующие матрицы:

$$W_1 = \begin{bmatrix} с & т & о & л & б \\ т & о & л & б & с \\ о & л & б & с & т \\ л & б & с & т & о \\ б & с & т & о & л \end{bmatrix} \Rightarrow W_2 = \begin{bmatrix} б & с & т & о & л \\ л & б & с & т & о \\ о & л & б & с & т \\ с & т & о & л & б \\ т & о & л & б & с \end{bmatrix}.$$

Окончательно имеем h_5 = «*лотбс*», $N = 4$.

Информация после преобразования (условной компрессии) имеет больший объем, чем до преобразования (при неизменной длине сообщения дополнительно требуется передавать или хранить число N).

В силу этой особенности метод эффективен при сжатии информации, состоящей из ограниченного набора символов (например, тексты программ), при котором последний столбец имеет в своем составе практически одинаковые символы, которые могут быть заменены на иные, используя, например, предыдущий метод. С другой стороны, анализируемый метод также обеспечивает хороший уровень преобразования двоичных данных. На практике этот метод применяется совместно с другими.

Обратная процедура. Имеем Y_k = «*лотбс*», $N = 4$. Строим матрицу W:

$$W=\begin{bmatrix} л \\ о \\ т \\ б \\ с \end{bmatrix}\Rightarrow\begin{bmatrix} л & б \\ о & л \\ т & о \\ б & с \\ с & т \end{bmatrix}\Rightarrow\begin{bmatrix} л & б & с \\ о & л & б \\ т & о & л \\ б & с & т \\ с & т & о \end{bmatrix}\Rightarrow$$

$$\Rightarrow\begin{bmatrix} л & б & с & т \\ о & л & б & с \\ т & о & л & б \\ б & с & т & о \\ с & т & о & л \end{bmatrix}\Rightarrow\begin{bmatrix} л & б & с & т & о \\ о & л & б & с & т \\ т & о & л & б & с \\ б & с & т & о & л \\ с & т & о & л & б \end{bmatrix}\Rightarrow W=\begin{bmatrix} б & с & т & о & л \\ л & б & с & т & о \\ о & л & б & с & т \\ с & т & о & л & б \\ т & о & л & б & с \end{bmatrix}.$$

На каждой итерации происходит дополнение матрицы исходной последовательностью и лексикографическая сортировка строк. В результате получаем исходное слово X_k = «*столб*», которое размещается в четвертой строке восстановленной матрицы.

Ниже приведен фрагмент кода программы, позволяющей построить матрицу W_1 для данного метода (каждая новая строка матрицы –сдвиг предыдущей на один символ влево). Следует обратить внимание на то, что *работа идет со словом с заранее известной длиной*. В общем случае массивы *slovo[]* и *w1[][]* должны быть объявлены и заполнены динамически (листинг 4.1).

```
char slovo[20];      //массив, хранящий слово,
                     //считываемое из файла,
char ch;             //хранит символ,
                     //считываемый из файла
int i;
ifstream in("a");    //открытие файла для чтения
if (!in){
cout<<"can not open file\n";}
k=0;
while (in){          //пока не конец файла
in.get(ch);          //считывание очередного символа
slovo[i]=ch;         //и запись его в массив slovo[]
k++;                 //впоследствии от k следует
                     //отнять 1, чтобы не учитывать
                     //символ конца строки
}                    //объявление массива w1[][]
char w1[k - 1][k - 1];
for (int i=0;i<=k - 2;i++)
{                    //заполнение первой строки
```

```
                    //матрицы w1[][] элементами
                    //матрицы slovo[]
w1[0][i]=slovo[i]; }
int index=1;
while (index<=k - 2){
int par= - 1;
int check=index - 1;
                    //цикл заполнения остальных
                    //строк матрицы w1[][]
for (int j=0;j<=k - 2;j++)
{check++;
   if (check<=k - 2)
    {w1[index][j]=slovo[check];}
    if (check>k - 2)
   {  par++;
      w1[index][j]=slovo[par]; }
   }
index++;}
                    //вывод заполненного массива
                    //w1[][]
for (int i=0;i<=k - 2;i++){
 for (int j=0;j<=k - 2;j++){
  cout<<w1[i][j]; }
  cout<<endl;}
return 0;}
```

Листинг 4.1. Программная реализация кода
для построения матрицы W_1 по методу Берроуза – Уиллера

Вопросы для самоконтроля

1. Охарактеризуйте цели компрессии данных.
2. Дайте общую классификацию методов сжатия. Приведите известные вам сведения о реализованных методах в компьютерных архиваторах данных (RAR, ARJ, ZIP).
3. Как оценить эффективность того или иного метода сжатия?
4. Поясните сущность метода интервалов. Приведите примеры.
5. Обясните сущность метода Берроуза – Уиллера. Поясните на примерах (исходное слово имеет вид «сжатие», «символ», «алфавит», «метод»).

Задание для самостоятельного выполнения

1. Применяя алгоритм Берроуза – Уиллера, выполните сжатие произвольного информационного сообщения, которое может содержать символы базового алфавита (русский/английский), служебные символы, цифры. Результат сжатия сохраните в файл с названием *compress.txt*.

2. Используя информацию из файла, созданного в пункте 1, реализуйте процесс декомпрессии (обратного преобразования) и запишите результат этого преобразования в файл *decompress.txt*. Кроме того, файл *decompress.txt* должен отображать результат работы алгоритма декомпрессии после каждого этапа добавления кодового столбца и сортировки матрицы W_2.

3. Вычислите первый, второй и интегральный коэффициенты компрессии.

4. Оцените эффективность работы метода Берроуза – Уиллера на основе анализа полученных результатов.

Примечание.

Для формирования массива W_1 можно применять как правосторонний, так и левосторонний сдвиг. На работе метода это не должно отразиться.

Для проверки работоспособности программы можно выбирать слова и сообщения произвольной длины, содержащие символы русского и английского алфавитов, цифры и служебные символы (особую сложность в некоторых случаях создают слова и сообщения, содержащие пробелы или комбинации из символов нескольких алфавитов (русского/английского)). В связи с тем, что в алгоритм программы должна быть включена сортировка некоторого массива символов, и эта операция должна сортировать массив построчно: не только по первому, но и по остальным символам слова, то для проверки можно использовать слова-перевертыши, например, «казак», слова, содержащие несколько одинаковых символов, например, «фарфор», «молоко», «колокольчик», «колокол», «колосок» и т. д.

4.2.3. Словарные методы сжатия. В известном смысле таблицу кодов ASCII (American Standard Code for Information Interchange, американский кодовый стандарт для обмена информацией), в которой один символ исходного алфавита всегда представляется одним байтом двоичных символов, можно рассматривать как один из методов сжатия. Здесь таблица может выступать как своеобразный словарь. Словарь этот является полным и исчерпывающим. Однако на практике в словарь помещаются наиболее часто встречающиеся слова и словосочетания.

В общем случае словарь может содержать произвольное число комбинаций произвольной длины. Основным вопросом при этом является размер словаря. Некоторые особенности рассмотрим на примере.

Пример 4.5. Положим, что текст, подлежащий компрессии, строится на основе 4-символьных слов из 26 символов латинского алфавита и 6 специальных знаков (например, точка, тире и т. д).

По принципу кодов ASCII каждый из 32 символов может быть заменен бинарным кодом длиной 5 бит. Общее же количество всех возможных слов длиной 4 бит на основе этих символов составляет: $32^4 = 2^{20} = 10\,048\,567$. Вероятность того, что данное произвольное слово будет равно какой-то конкретной комбинации, составит $P = 1/10\,048\,567$ (принимая, что все слова равновероятны). Если все комбинации разместить в словаре, то каждой комбинации должен соответствовать бинарный код длиной 20 бит. Встает вопрос. Что лучше: **32** и **5** или **10 048 567** и **20**?

Положим, что из миллиона слов все-таки имеются те, которые встречаются реже, другие – чаще. Допустим, что в словаре размещено 256 наиболее часто встречающихся слов, т. е. весь массив из 4-разрядных слов разделен на 2 подмассива: подмассив $\{X_4\}^1$, состоящий из 2^8 комбинаций, и подмассив $\{X_4\}^2$, состоящий из остальных комбинаций и находящийся вне словаря.

Очевидно, что в силу принятого разделения принцип кодирования слов обоих подмассивов должен быть разным: каждому из слов $\{X_4\}^1$ должна соответствовать фиксированная бинарная последовательность из 8 бит плюс дополнительный символ (например 0), который дописывается впереди как флаг, что в сумме дает 9 бит. Аналогично рассуждая, слова второго подмассива можно заменить 20-разрядным кодом плюс флаг («1»), что в сумме дает 21 бинарный символ.

Если предположить, что вероятность появления в тексте каждого из слов словаря равна P_c, а каждого из остальных слов – $(P_c - 1)$, то можем дать вероятностно-символьную оценку принятому принципу построения словаря:

$$C = P \cdot 9 + (1 - P) \cdot 21 = 9 \cdot P + 21 - 21 \cdot P = 21 - 12 \cdot P.$$

В соответствии с этим можно определить, каким должно быть P, чтобы размещение определенных слов в словаре дало нужный эффект (в сравнении с методом размещения в словаре всех 2^{20} слов). Очевидно, если мы разместим все слова в словаре, то значение C составит 20. Следовательно, нужно решить простое неравенство:

$$21 - 12 \cdot P < 20,$$

следовательно, $P > 1/12 \approx 0.084.$

В общем случае при построении словаря исходят из следующих *критериев*:

1) требуемая скорость кодирования комбинаций (сжатия) и скорость обратного преобразования: чем больше словарь, тем скорость ниже;

2) размеры требуемой памяти: чем больше словарь, тем больше размер требуемой памяти для хранения словаря;

3) эффективность сжатия в первом приближении можно оценить параметром *C*.

Наилучшим является подход, при котором словарь строится под тип данных. Важнейшей проблемой при проектировании словарной схемы является выбор размера кодового словаря.

В связи с этим часто ограничение накладывается на размер фраз, помещаемых в словарь. Относительно размера фраз составление словаря может быть статическим, полуадаптивным и адаптивным.

Способ разбиения текста на фразы для их дальнейшего кодирования называется ***разбором текста***.

Статический (неизменный) ***словарь*** строится предварительно для разнообразных типов документов и практически не изменяется. ***Полуадаптивный алгоритм*** предполагает создание словаря под конкретный тип документов. ***Адаптивный алгоритм*** строит словарь под каждый документ.

Важнейшей особенностью адаптивного (и в определенной степени – полуадаптивного) словаря является необходимость его передачи вместе со сжатой информацией.

4.2.4. Метод Лемпеля – Зива. До появления метода, описанного в 1977 г. Лемпелем и Зивом (Lempel, Ziv), все методы были в основном статическими.

Важнейшей особенностью метода Лемпеля – Зива было то, что не требуется передавать словарь, создаваемый под каждый документ. Этот словарь формально становится частью сжатой информации.

Сейчас созданы и составляют основу всех известных компьютерных архиваторов многочисленные модификации первоначального метода, известного как LZ77. Основой метода является поиск комбинаций, которые уже встречались, и замена того повтора на комбинации символов (p, q), где p – индекс начала комбинации в передаваемом документе, а q – длина этой повторяющейся комбинации.

Преобразование информации заключается в «прохождении» текста через два окна и параллельном анализе символов с текущим словарем. Общая схема реализации алгоритма показана на рис. 4.1.

Как правило, n (общая длина двух окон, называемых буферами) составляет и тысячи символов. Окно 02 – буфер данных, в него «въезжают» данные (исходный текст), которые нужно сжимать, кодировать. Далее эти данные «въезжают» в окно 01 – словарь.

Преобразование осуществляется по принципу размещения и передачи исходного сообщения через буфер, который делится на две части: буфер данных (буфер кодирования) и словарь.

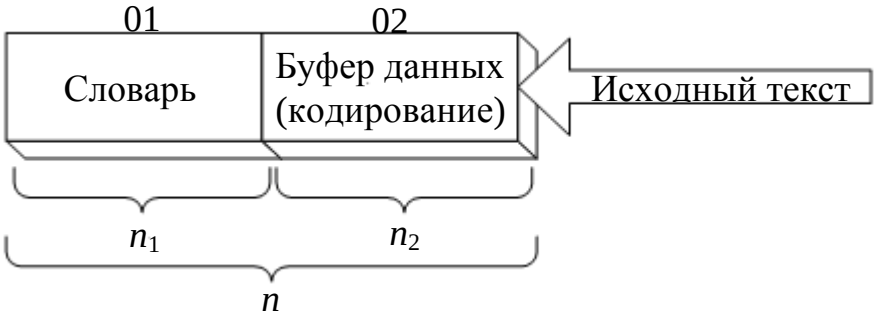

Рис. 4.1. Иллюстрация общего принципа преобразования данных по методу Лемпеля – Зива

Принцип состоит в том, чтобы выявить в тексте повторяющиеся комбинации. Анализируемые данные находятся в буфере кодирования, а обнаруженные ранее повторения – в словаре. Найденный повторяющийся ряд символов в буфере данных заменяется в основном парой (p, q) символов. Кроме этого, к данной паре добавляется еще один символ (c_i), который является частью потока X_k и следует за найденным повторением в буфере данных. Буфер работает по принципу регистра сдвига.

Пример 4.6. Используем для передачи данных алфавит, состоящий из четырех знаков: $A\{0, 1, 2, 3\}$. Необходимо сжать последовательность:

$X_k = 2000302013020130313031303133333333$.

Полагаем, что $n = 28$, $n_1 = 13$, $n_2 = 15$ символам.

Для обозначения p и q используем четверичную систему счисления, где $0_4 = 00$, $1_4 = 01$, $2_4 = 02$, $3_4 = 03$, $4_4 = 10$, $5_4 = 11$ и т. д.

На первом шаге производится анализ в соответствии с рис. 4.2.

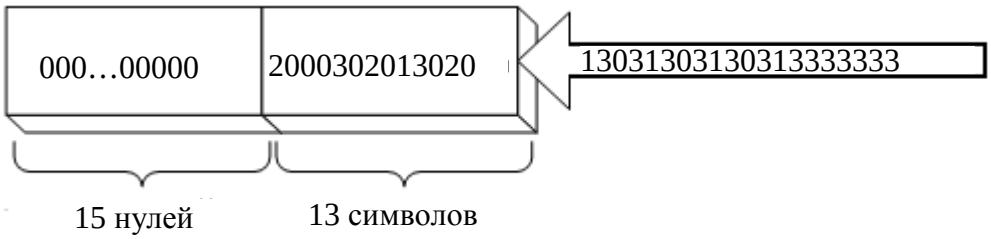

Рис. 4.2. Схематичное представление анализа на первом шаге преобразования

Анализируем первый символ в буфере кодирования на предмет соответствия (наличия) такого же символа или нескольких символов в словаре. Как видно из рис. 4.2, таких символов нет, следовательно, принимается $p = 0\,(00)$. Длина повторения $q = 0\,(00)$. Таким образом, имеем следующую триаду: $(p, q, c_i) = (00\ 00\ 2)$. Символ c_i (2) сдвигается в левое окно (на q + 1 символов).

На втором шаге исходное состояние показано на рис. 4.3.

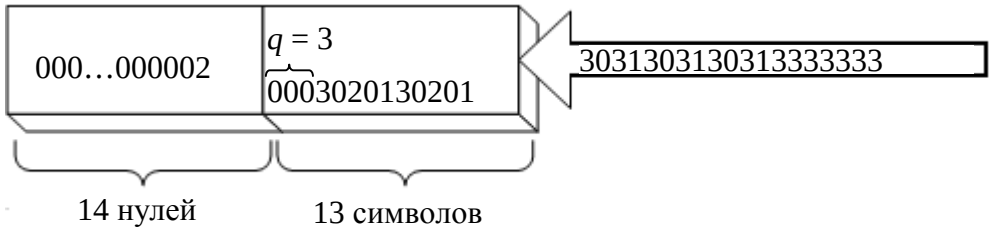

Рис. 4.3. Схематичное представление анализа на втором шаге преобразования

Находим повторение (000), длина этого повторения составляет три символа: $q = 3$. Поскольку в словаре нулевые символы записываются с 1-й по 14-ю позицию, то индексом p (началом повторения) может быть выбрано любое число от 1 до 12. Положим, выбрано 6: $p = 6$.

Итак, получим следующую триаду: $(p,\ q,\ c_i) = (6, 3, 3) \rightarrow (12\ 03\ 3)$. Содержимое буфера данных сдвигается на 4 позиции (на q + 1 символ).

На третьем шаге ситуация поясняется рис. 4.4.

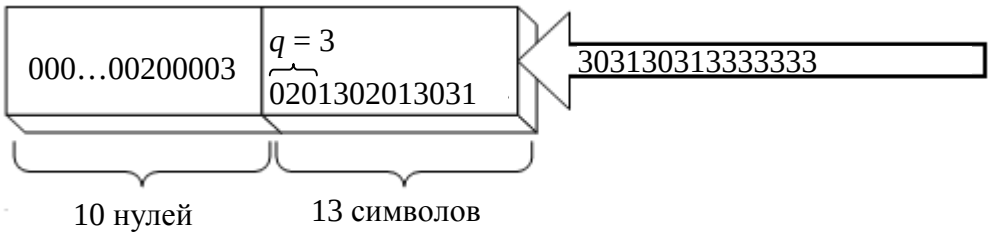

Рис. 4.4. Схематичное представление анализа на третьем шаге преобразования

В этой ситуации наиболее длинный повтор – 020. Следовательно, $q = 3,\ p = 10$.

Передвигаем анализируемую последовательность на $q + 1 = 4$ позиции влево. Выходная триада будет следующей: $(p, q, c_i) = (10, 3, 1) \rightarrow (22\ 03\ 1)$.

На четвертом шаге анализируется состояние в соответствии с рис. 4.5.

В этой ситуации наиболее длинный повтор выглядит так: 3020130, $q = 7, p = 11$. Осуществляем сдвиг на $q + 1 = 8$ символов.

Очередная триада: $(p, q, c_i) = (11,\ 7,\ 3) \rightarrow (23\ 13\ 3)$.

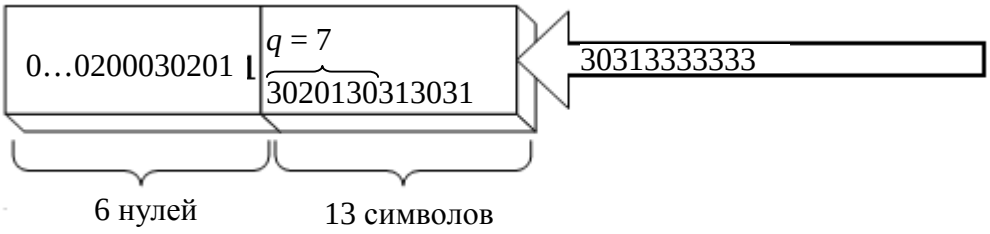

Рис. 4.5. Схематичное представление анализа на четвертом шаге преобразования

Подобным образом будет проанализирована вся входная последовательность. Полученные триады 00002 12033 22031 23133 30301 в принятой форме записи будут рассматриваться как входное сообщение на входе декомпрессора (обратного преобразования).

В исходном состоянии при *обратном преобразовании* в окно словаря записывают 15 нулей (рис. 4.6).

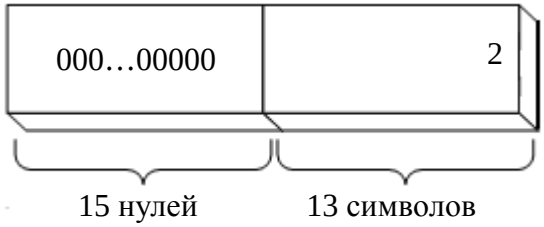

Рис. 4.6. Схематичное представление
исходного состояния окон при обратном
преобразовании данных по методу Лемпеля – Зива

Дальше анализируем очередные 5 символов (12033), из чего следует, что $q = 3$ и $p = 6$. Поскольку в словаре содержатся только одни нули, то делается вывод: повтором является комбинация из трех нулей (000). В словарь записывается 000, а в буфер кодирования – 3 (рис. 4.7). Подобным образом будет проанализирована вся входная

последовательность. Результатом преобразования (декомпрессии) является сдвигаемое содержимое словаря.

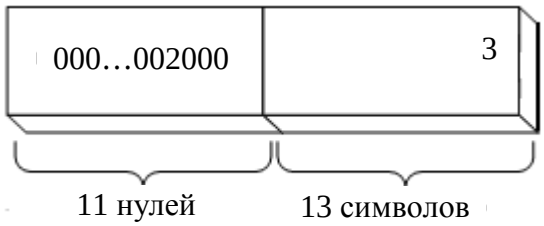

Рис. 4.7. Схематичное представление состояния окон после первого шага обратного преобразования

Вопросы для самоконтроля

1. Поясните особенность словарных методов сжатия данных.
2. При каких условиях словарный метод является эффективным?
3. В чем сущность метода Лемпеля – Зива? Приведите примеры известных вам модификаций базового метода (LZ77).
4. Используя метод Лемпеля – Зива, покажите порядок прямого и обратного преобразований, если исходное сообщение имеет вид: «информатизация и информированность», «кодовое слово», «компрессия и декомпрессия».

4.3. Статистические методы сжатия

Статистические методы используют вероятностные характеристики символов применяемого алфавита для построения стратегии сжатия. Среди этих методов наиболее известны метод Шеннона –Фано (Shannon – Fano) и метод Хаффмана (Huffman). Кратко рассмотрим особенности их применения.

Отметим, что *основная идея* методов состоит в замене символов исходного алфавита бинарными кодами (последовательностями) различной длины: символам с наибольшими вероятностями появления должны соответствовать коды наименьшей длины и наоборот.

Процесс генерации бинарных кодов должен отвечать двум основным *требованиям*, налагаемым на вид конечных кодовых комбинаций:

1) все комбинации должны быть различны;

2) любая комбинация меньшей длины не может быть началом любой комбинации большей длины (свойство префикса).

Прямое преобразование (сжатие) состоит в замене каждого исходного символа соответствующим бинарным кодом, обратное преобразование – в обратной замене.

4.3.1. Метод Шеннона – Фано. Отсортированные в последовательности от максимального до минимального вероятности появления всех символов алфавита (можно использовать вероятности, полученные при изучении энтропийных свойств алфавита) в тексте делят на две части в такой пропорции, чтобы сумма вероятностей в каждой из частей была максимально близка к 0,5. Символам верхней части приписывается старший символ бинарного кода – 1, нижней части – 0. Далее каждая из двух частей в свою очередь делится также на две части в такой пропорции, чтобы сумма вероятностей появления символов алфавита, составляющих каждую из частей, характеризовалась наименьшей разностью. Символам верхней из полученных частей приписывается второй символ кода – 1, нижней – также второй символ, но 0.

Процесс деления и генерации очередных символов бинарного кода продолжается до тех пор, пока после всех операций разделения массива полученная часть (подмассив) не будет состоять только из одного символа исходного алфавита.

Пример 4.7. Имеем исходный алфавит $A\{a_i\}$, $[i = 1\ldots10]$, т. е. мощность алфавита равна 10: $N(A) = 10$. Полагаем, что вероятности появления в документе каждого из символов алфавита соответствуют следующим значениям (помним, что сумма всех вероятностей равна 1):

$$p(a_1) = 0.05; \quad p(a_6) = 0.12;$$
$$p(a_2) = 0.10; \quad p(a_7) = 0.05;$$
$$p(a_3) = 0.03; \quad p(a_8) = 0.10;$$
$$p(a_4) = 0.20; \quad p(a_9) = 0.15;$$
$$p(a_5) = 0.15; \quad p(a_{10}) = 0.05.$$

Отсортируем эти вероятности в порядке убывания и разделим весь массив на две части с учетом вышеотмеченного правила. Положим, что верхнюю часть образуют символы (или соответствующие им вероятности) a_4, a_5, a_9, нижнюю – a_6, a_2, a_8, a_1, a_7, a_{10}, a_3. Верхним символам приписываем старший бит – 1, нижним – 0. В свою очередь, верхний подмассив делим на два очередных: один состоит из единственного элемента – a_4, другой – из двух остальных. Каждому из трех этих символов приписываем по одному соответствующему символу бинарного кода. В результате всех

делений и приписывания коду очередного двоичного символа получим следующие кодовые комбинации:

$p(a_4) = 0.20$ 11; $p(a_8) = 0.10$ 0011;

$p(a_5) = 0.15$ 101; $p(a_1) = 0.05$ 0010;

$p(a_9) = 0.15$ 100; $p(a_7) = 0.05$ 0001;

$p(a_6) = 0.12$ 011; $p(a_{10}) = 0.05$ 00000;

$p(a_2) = 0.10$ 010; $p(a_3) = 0.03$ 00001.

Как видим, коды соответствуют двум основным требованиям.

Блоки прямого и обратного преобразований осуществляют операции, используя полученную таблицу.

Пусть исходное (подлежащее сжатию) сообщение имеет вид: $a_5 a_4 a_4 a_9 a_6 a_1 a_1 a_8 a_7 a_9$. После выполнения операции сжатия имеем $a_5 a_4 a_4 a_9 a_6 a_1 a_1 a_8 a_7 a_9 \Rightarrow 10111111000110010001000110001100$.

Общая длина полученной двоичной последовательности составляет 32 бита. Если предположить, что при стандартной операции замены символов кодами одинаковой длины (по принципу кодов ASCII) используется код минимальной длины (требуется при этом минимум 4 битовых символа), то длина бинарного кода была бы равной 40 символам (4 · 10).

Для выполнения обратного преобразования необходимо знать таблицу кодировки, а также значения $l_{\min}$, $l_{\max}$ (минимальную и максимальную длину кода в битах соответственно). Для нашего примера $l_{\min} = 2, l_{\max} = 5$.

Операция осуществляется по шагам: анализируются первые $l_{\min}$ символов (10) путем их сравнения с таблицей кодировки. Если найдено совпадение, то на выходе будет первый символ (в примере такого совпадения нет) декомпрессированного сообщения и далее будут проанализированы очередные $l_{\min}$ бит. Если же соответствие не наблюдается, то анализируется последовательность длиной на единицу больше (в примере – 101) и т. д.

Как видно из выполненного анализа, метод характеризуется неоднозначностью, т. е. возможностью реализации различных вариантов деления посортированного исходного множества. Очевидно, что каждый из вариантов приводит к различным кодовым комбинациям.

Характеристикой эффективности алгоритма является так называемый интегральный коэффициент компрессии:

$$C = \sum_{i=1}^{n} p(i) \cdot l_i,$$

где l_i – количество символов кода, соответствующее символу алфавита a_i.

В нашем примере $C = \underbrace{0,05}_{a_1} \cdot 4 + \underbrace{0,10}_{a_2} \cdot 3 + \underbrace{0,03}_{a_3} \cdot 5 + ... + \underbrace{0,05}_{a_{10}} \cdot 5.$

Значение коэффициента отвечает среднему количеству символов кода на один символ алфавита (a_i).

Программная реализация сортировки массива вероятностей по убыванию может иметь следующий вид (листинг 4.2).

```
for ( int i=0;i<ind;i++)
 {
  for ( int j=(i+1);j<ind;j++)
  { if (ver[i]<ver[j])
                        //ver - массив вероятностей
     { a=ver[i];        //переменная a типа int
       ver[i]=ver[j];
       ver[j]=a;
       b[0]=use_alf[i];
                        //переменная b типа char
       use_alf[i]=use_alf[j];
                        //массив    use_alf[]    -    массив,
                        //хранящий все символы
                        //исходного алфавита, входящие
                        //в информационное сообщение
       use_alf[j]=b[0]; }
  }
 }
```

Листинг 4.2. Программная реализация сортировки массива вероятностей по убыванию

Чтобы разбить исходный массив вероятностей на две приблизительно равные части, необходимо вычислить сумму (*sum*) значений его элементов (в первом случае *sum* равна 1). Далее ввести некоторую вспомогательную переменную *flag*, определяемую как *sum*/2.

Двигаясь, например, сверху, необходимо складывать в *sum2* элементы исходного массива вероятностей до тех пор, пока *sum2* не станет больше переменной *flag*. Запомнить в переменную *index* индекс последнего элемента, прибавление которого к переменной *sum2* еще не вызвало превышение значения *sum2* над значением *flag*. Все элементы массива *Code* с 0 по *index* инициализировать единицами, а с *index+1* по *count* (общее число элементов в исходном

массиве) –нулями. Выполнить те же действия в отношении к каждому из вновь полученных массивов (подмассивов исходного).

Для реализации этого алгоритма необходимо написать функцию, рекурсивно вызывающую саму себя для каждого из вновь полученных подмассивов.

Вопросы для самоконтроля

1. Поясните сущность вероятностных методов сжатия данных.
2. Что является критериями корректного формирования кодовых таблиц?
3. Приведите свой вариант генерации кодов для исходных данных из примера 4.7. Сравните с описанным.
4. Постройте кодовую таблицу (по методу Шеннона – Фано) для равновероятного появления символов алфавита мощностью 10; 12; 15.
5. Постройте кодовую таблицу для случая следующего распределения вероятностей появления символов алфавита (в порядке возрастания индексов символов): 0.02, 0.04, 0.08, 0.009, 0.11, 0.04, 0.02, 0.15, 0.05, 0.05, 0.10, 0.20, 0.05. Рассчитайте интегральный коэффициент компрессии.
6. Постройте таблицу кодов для символов русского алфавита, используя ранее вычисленные вероятностные значения (при нахождении энтропии).

Задание для самостоятельного выполнения

1. На основе произвольного текстового файла постройте таблицу кодов, встречающихся в нем символов, используя метод Шеннона – Фано.
2. Закодируйте исходную информацию на основе построенной таблицы кодов.
3. Вычислите первый, второй и интегральный коэффициенты сжатия. Оцените эффективность работы метода.
4. Выполните операцию обратного преобразования.

Примечание.

Результаты компрессии, декомпрессии, таблицу кодов следует сохранить в различных файлах как результаты работы программы. Для оценки эффективности работы алгоритма необходимо исходный файл (используя коды ASCII) представить в двоичном виде.

Кроме того, поскольку будет формироваться не универсальная, а частная таблица кодов для конкретного информационного сообщения, необходимо учесть объем таблицы кодов, который должен передаваться получателю вместе со сжатой информацией для оценки эффективности работы метода.

4.3.2. Метод Хаффмана. Основная отличительная особенность метода в сравнении с предыдущим заключается в формировании бинарного кода. В последнем случае для этого нужно построить бинарное дерево объединением попарно отсортированных символов алфавита, начиная с двух нижних. Причем группировать следует символы с примерно одинаковыми вероятностями. Объединенные символы (пара) создают новый виртуальный символ с вероятностью, равной сумме вероятностей объединенных символов.

Объединение символов, таким образом, создает ветви и узлы дерева. Каждая из ветвей должна быть обозначена бинарным символом (в паре одному соответствует 1, другому – 0). Последние из объединенных символов образуют корень дерева.

Бинарным кодом, который соответствует определенному символу исходного алфавита, будет комбинация, полученная после прохождения по ветвям дерева от его корня до соответствующего символа исходного алфавита.

Сжатие и декомпрессия сообщений осуществляются по тем же принципам, что и для предыдущего метода.

Рассмотрим это на примере.

Пример 4.8. Исходными являются данные из примера 4.7:

$$p(a_1) = 0.05; \qquad p(a_6) = 0.12;$$
$$p(a_2) = 0.10; \qquad p(a_7) = 0.05;$$
$$p(a_3) = 0.03; \qquad p(a_8) = 0.10;$$
$$p(a_4) = 0.20; \qquad p(a_9) = 0.15;$$
$$p(a_5) = 0.15; \qquad p(a_{10}) = 0.05.$$

После сортировки попарно объединяем символы, начиная с двух нижних: a_3 и a_{10}. Верхней ветви (a_{10}) приписываем младший символ кода – 1, нижней (a_3) – 0. Полученный в результате объединения узел далее рассматривается как виртуальный символ с вероятностью появления, равной 0,08 (0,03 + 0,05).

Общий вид дерева представлен на рис. 4.8.

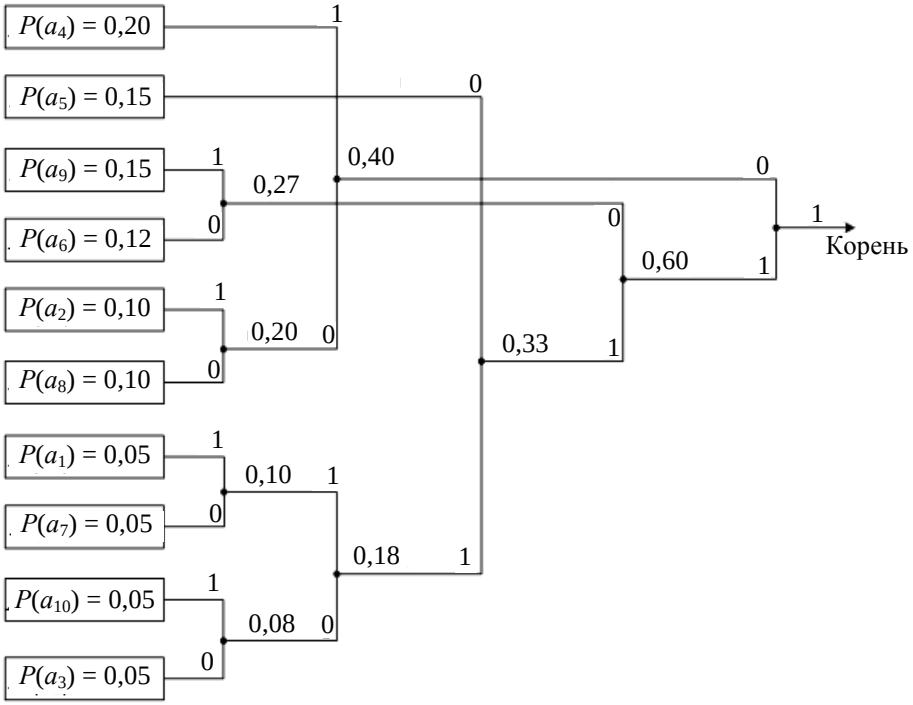

Рис. 4.8. Вариант бинарного дерева для рассматриваемого примера

Запишем соответствующие символам коды посредством обхода ветвей дерева от корня к этому символу:

$a_1 \rightarrow 11111$;
$a_2 \rightarrow 001$;
$a_3 \rightarrow 11100$;
$a_4 \rightarrow 01$;
$a_5 \rightarrow 110$;
$a_6 \rightarrow 100$;
$a_7 \rightarrow 11110$;
$a_8 \rightarrow 000$;
$a_9 \rightarrow 101$;
$a_{10} \rightarrow 11101$.

Нетрудно понять, что и данный метод характеризуется многовариантностью кодовых комбинаций. Критерием эффективности того или иного варианта является интегральный коэффициент компрессии. В обоих рассмотренных методах *наилучшему варианту построения кодовой таблицы соответствует наименьший коэффициент*.

Вопросы для самоконтроля

1. Поясните сущность сжатия данных по методу Хаффмана.
2. Что является критериями корректного формирования кодовых таблиц?
3. В чем состоит основное различие в алгоритмах на основе методов Шеннона – Фано и Хаффмана?

4. Приведите свой вариант генерации кодов для исходных данных из примера 4.8. Сравните с описанным.

5. Постройте кодовую таблицу (по методу Хаффмана) для равновероятного появления символов алфавита мощностью 10; 12; 15.

6. Постройте кодовую таблицу для случая следующего распределения вероятностей появления символов алфавита (в порядке возрастания индексов символов): 0.02, 0.04, 0.08, 0.009, 0.11, 0.04, 0.02, 0.15, 0.05, 0.05, 0.10, 0.20, 0.05. Рассчитайте интегральный коэффициент компрессии.

7. Постройте дерево для символов русского алфавита, используя ранее вычисленные вероятностные значения (при нахождении энтропии).

Задание для самостоятельного выполнения

1. На основе произвольного текстового файла постройте таблицу кодов (бинарных последовательностей), применяя алгоритм Хаффмана. Символы исходного текстового файла могут быть символами русского и английского алфавитов, цифрами, специальными символами.

2. Представьте исходную информацию в кодах на основе построенного бинарного дерева Хаффмана. Результат преобразования сохраните в отдельном файле.

3. Вычислите первый, второй и интегральный коэффициенты сжатия. Оцените эффективность работы метода.

4. Выполните операцию обратного преобразования. Сохраните результат в новом файле.

Примечание.

Результат компрессии, декомпрессии, таблицу кодов следует сохранить в различных файлах как результаты работы программы. Для оценки эффективности работы алгоритма необходимо исходный файл (применяя коды ASCII) представить в двоичном виде.

Для построения такого дерева Хаффмана можно использовать структуры.

5. КРИПТОГРАФИЧЕСКИЕ МЕТОДЫ ПРЕОБРАЗОВАНИЯ ИНФОРМАЦИИ

В течение последних 20 лет наблюдается бурное развитие открытых академических исследований в области криптографии. Пока обычные граждане использовали «классическую» криптографию, «компьютерная» криптография еще со времен Первой мировой войны применялась исключительно в военных целях.

Говоря об исторических аспектах научных исследований в области криптографии, необходимо отметить тот факт, что весь период с древних времен до 1949 г. можно назвать донаучным, когда средства шифрования информации не имели строгого математического обоснования. Поворотным моментом, придавшим криптографии научность и выделившим ее в отдельное направление математики, явилась публикация в 1949 г. статьи К. Шеннона «Теория связи в секретных системах».

Современная криптография широко используется и за стенами военных ведомств. Рядовые пользователи информационных технологий получили возможность защититься от всемогущих вредителей (не только от хакеров или кракеров). Ниже проанализируем основные особенности и оценим эффективность криптографических методов преобразования (защиты) информации.

5.1. Основные понятия и определения

Криптография является одной из двух ветвей общего научного направления – ***криптологии***. Второй ветвью криптологии является ***криптоанализ***. Цели криптографии и криптоанализа прямо противоположны.

Криптографические методы нашли широкое применение в практической информатике для решения многочисленных проблем *информационной безопасности*. В проблематике современной криптографии можно выделить следующие три типа основных задач:

1) обеспечение *конфиденциальности* (*секретности*);

2) создание условий для *анонимности* (*неотслеживаемости*);

3) обеспечение *аутентификации* информации и источника сообщения.

Первый тип задач относится к защите информации от несанкционированного доступа по секретному ключу. Доступ к информации (информационным ресурсам) имеют только обладатели ключа. Второй и третий типы задач обязаны своей постановкой массовому применению электронных способов обработки и передачи информации (банковская сфера, электронная коммерция, каналы межличностной коммуникации и др.).

В дополнение к некоторым определениям из рассматриваемой предметной области, сформулированным в подразделе 1.1, поясним значение вновь введенных.

Конфиденциальность – свойство информации быть известной только допущенным и прошедшим авторизацию субъектам системы (пользователям, программам, процессам).

Авторизация – предоставление субъектам доступа к объектам системы.

Аутентификация – проверка *идентификации* пользователя, устройства или другого компонента в системе (обычно для принятия решения о разрешении доступа к ресурсам системы). Частным

вариантом аутентификации является установление принадлежности сообщения конкретному автору.

Криптографическое преобразование, как и вышерассмотренные типы (помехоустойчивое кодирование и сжатие), состоит из двух этапов: прямого и обратного. Прямое преобразование называют *шифрованием* (в соответствии со стандартом ISO 7492-2 – *зашифрованием*, encrypt), обратное – *дешифрованием* (*расшифрованием*, decrypt). Процесс передачи зашифрованных сообщений иллюстрирует рис. 5.1.

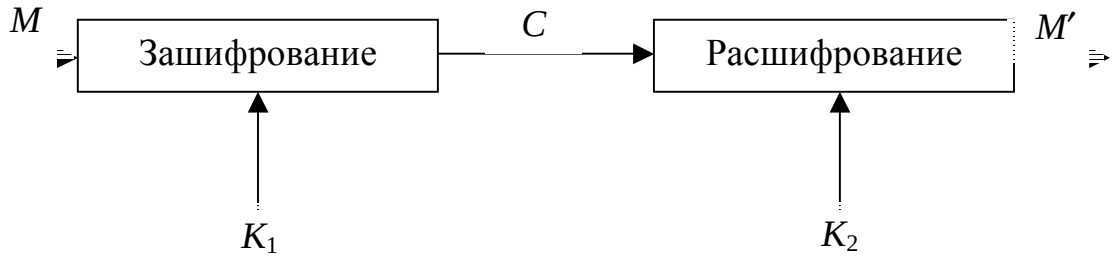

Рис. 5.1. Зашифрование и расшифрование информации

Исходное сообщение называется открытым текстом (M от англ. message). Зашифрованное сообщение – *шифртекстом*, или *шифрограммой* (C от англ. cipfer). После обратного преобразования получаем исходный (или приближенный к нему) документ (M'). Оба преобразования осуществляются на основе ключей (K_1 и K_2).

Ключ – секретный параметр, управляющий ходом преобразования. Ключ определяет конкретный вариант преобразования.

Если значения ключей K_1 и K_2 полностью совпадают, то такие криптосистемы называют *симметричными*, в противном случае – *асимметричными*.

Если пользоваться символьными обозначениями, введенными нами ранее (X_k – данные до прямого преобразования, X_n – данные после прямого преобразования), т. е. положить, что $X_k = M$ и $X_n = C$, то сравнительной характеристикой криптографических методов будет равенство $k = n$. Это означает, *что длина открытого и зашифрованного сообщений не меняется* (исключение составляют методы шифрования с использованием электронной цифровой подписи).

Голландский криптограф Керкхофф (1835–1903) впервые сформулировал *правила оценки стойкости шифра* перед его взломом (раскрытием), в соответствии с которым:

1) весь механизм преобразования считается известным злоумышленнику (интрузу);

2) криптостойкость (надежность) алгоритмов преобразования определяется только неизвестным значением ключа.

Интруз – физическое лицо или процесс, которые реализуют неразрешенный, или несанкционированный, доступ к информации (атаку на систему).

В современном криптоанализе (вспомним, что целью криптоанализа является взлом шифра) рассматриваются атаки на засекречивающие системы на основе следующих известных данных:

– шифртекста;
– открытого текста и соответствующего ему шифртекста;
– выбранного открытого текста;
– выбранного шифртекста;
– адаптированного открытого текста;
– адаптированного шифртекста.

5.2. Базовые криптографические алгоритмы

Криптографический алгоритм, называемый также шифром, в наиболее общем виде представляет собой математическую функцию, которая используется для зашифрования и расшифрования.

В предыдущем подразделе упоминалось о двух классах криптосистем, реализующих подобные алгоритмы: симметричные и асимметричные системы. ***Симметричность*** означает, что ключи, задающие пару взаимно обратных преобразований, идентичны либо могут быть получены один из другого путем простых преобразований. ***Асимметричность*** подразумевает, что ключи различны и знание одного (K_1) не должно позволять определению другого (по крайней мере, нахождение другого должно быть чрезвычайно трудным).

Зашифрование и расшифрование с помощью симметричного алгоритма записывается следующим образом:

$$E_k(M) = C,\ D_k(C) = M \text{ (или } M'). \qquad (5.1)$$

Предполагается, что возможно $E_k = D_k = K$.

Симметричные алгоритмы подразделяются, в свою очередь, на два подкласса: потоковые и блочные. *Потоковые алгоритмы* обрабатывают открытый текст побитово, *блочные* – группы битов (блоки) открытого текста.

В асимметричных алгоритмах невозможно выполнение равенств $E_k = D_k = K$.

В некоторых случаях сообщения шифруются закрытым (D_k) ключом, а расшифровываются – открытым. Такой подход используется в цифровых подписях.

До появления компьютеров криптография основывалась на *текстовых алгоритмах*. Основой их были операции замены одних символов другими либо перестановка символов местами. Первые алгоритмы относятся к классу *подстановочных*, другие – *перестановоч-ных*. Современные криптосистемы используют как подстановки, так и перестановки символов.

5.2.1. Подстановочные шифры. Сущность подстановочного шифрования состоит в том, что, как правило, исходный (M) или зашифрованный текст (C) используют один и тот же алфавит, а ключом является алгоритм подстановки. Такой шифр называется простым, или моноалфавитным.

Примером такого шифра является знаменитый шифр Цезаря, в котором каждый символ открытого текста заменяется символом, находящимся тремя символами правее (по модулю 26 или по принципу кольца): «A» меняется на «D», «B» – на «E», «W» – на «Z», «X» – на «A» и т. д. (в некоторых случаях во внимание принимается 27-й символ – пробел). Для расшифрования необходимо выполнить обратную замену.

Пример 5.1. Имеем открытый текст M = «cba». На основе шифра Цезаря C = «gfd».

Простой подстановочный шифр применяется, например, в системе UNIX: простая программа шифрования (ROT13) использует смещение на 13 позиций, т. е. символ «A» заменяется на «N» и т. д.

Анализируемые шифры взламываются без труда, поскольку не скрывают частоту (вероятность) применения различных символов в открытом тексте.

5.2.2. Перестановочные шифры. Перестановочные шифры используют перестановку символов исходного сообщения в соответствии с установленным правилом, открытый текст остается неизменным, но символы в нем «перетасовываются» (подвергаются пермутации). Так, в простом вертикальном перестановочном шифре

открытый текст пишется по горизонтали на разграфленном листе бумаги фиксированной длины, а шифртекст считывается по вертикали. Рассмотрим это на примере.

Пример 5.2. *М* = «ВАСЯ ЛЮБИТ МАШУ». Запишем этот текст как показано на рис. 5.2.

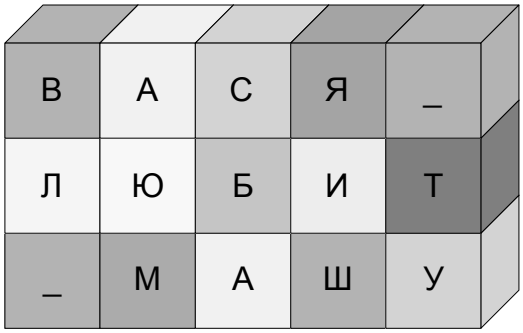

Рис. 5.2. Использование простого
вертикального перестановочного шифра

Считывание по столбцам снизу вверх приводит к такому шифртексту: *С* = «_ЛВМЮААБСШИЯУТ_».

Принцип записи исходного сообщения и порядок считывания символов может быть различным. Обратимся к следующему примеру.

Пример 5.3. Открытый текст возьмем из предыдущего примера, а запишем его так, как показано на рис. 5.3.

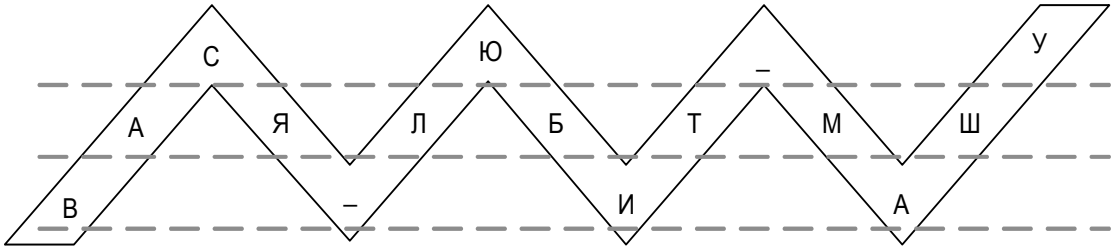

Рис. 5.3. Использование перестановочного шифра

Осуществляя считывания по уровням, начиная с верхнего, получим следующий шифртекст: *С* = «СЮ_УАЯЛБТМШВ_ИА».

Существуют еще более сложные перестановочные шифры, но компьютеры достаточно быстро справляются с ними. При этом использование данных шифров требует большого объема памяти.

5.3. Характеристики и реализация симметричных и асимметричных алгоритмов

Криптографическое преобразование в большинстве современных информационных систем основано на решении задачи *дискретного логарифмирования в конечном поле*. Сущность задачи состоит в следующем. Если есть целые положительные числа *a*, *x*, *n*, легко можно вычислить:

$b = a^x \bmod n$.

Однако, если известны *b*, *a* и *n*, то *x* можно найти на основе вычисления логарифма. Оказывается, что при некоторых значениях *b*, *a* и *n* определить *x* является большой проблемой.

Например, если $3^x = 15 \bmod 17$, то $x = 6$. Однако нетрудно заметить, что у следующего уравнения нет решений: $3^x = 7 \bmod 13$. Еще труднее решить задачу для 1024-битовых чисел.

Особый интерес представляют дискретные *логарифмы мультипликативных групп полей простых чисел*: *GF*(*p*). Если *p* – простое число, используемое в качестве модуля, то сложность поиска дискретных логарифмов в *GF*(*p*) соответствует разложению на множители числа *n* того же размера, где *n* – произведение двух простых чисел примерно равной длины, т. е. вычисление дискретных логарифмов тесно сопряжено с разложением на множители. Если стоит вопрос дискретного логарифмирования, то решается задача разложения на множители (истинность обратного пока никем не доказана).

С. Полиг (S. Pohlig) и М. Хеллман (M. Hellman) нашли способ быстрого вычисления дискретных логарифмов в поле *GF*(*p*) при условии, что $p - 1$ раскладывается только на малые простые множители. По этой причине в криптографии используются только такие поля, в которых у числа $p - 1$ есть хотя бы один большой простой множитель.

Описанные особенности имеют отношение в основном к асимметричным алгоритмам. Далее рассмотрим наиболее известные алгоритмы, относящиеся к обоим типам шифрования.

5.3.1. Симметричный алгоритм DES. В симметричных системах отправитель и получатель используют один и тот же ключ, который должен быть известен только им.

Для понимания существа анализируемого алгоритма целесообразно рассмотреть следующий простой пример.

Пример 5.4. Пусть открытое сообщение в двоичной форме имеет следующий вид: $M_{(0,1)}$ = 10101100. Считаем, что выбран

симметричный ключ $K_1 = K_2 = K = 1010$. Используется самая простая операция шифрования:

$$C_i = M_i \oplus K$$

и операция расшифрования:

$$M_i' = C_i \oplus K,$$

где i – i-й блок шифрования; M_i – i-я часть сообщения.

Нетрудно убедиться, что $C = C_1 C_2 = 00000110$, а сложение каждых четырех бит шифртекста с ключом восстанавливает исходное сообщение (здесь $\oplus$ – операция суммирования по модулю 2).

Алгоритм DES (Data Encryption Standard), как и другие симметричные и асимметричные алгоритмы, использует множественные арифметическо-логические преобразования исходного текста.

Стандарт шифрования DES был разработан в 1970-х г. Национальным институтом стандартизации США (ANSI) и называется алгоритмом DEA (Data Encryption Algorithm).

Основные идеи алгоритма были предложены компанией IBM еще в 1960-х гг. и базировались на идеях, описанных Клодом Шенноном в 1940-х г.

DEA оперирует с блоками данных размером 64 бит и использует ключ длиной 56 бит. Каждый 8-й бит ключа (всего – 8) применяется для контроля четности, т. е. предназначен для контроля ошибок. Такая длина ключа соответствует 10^{17} комбинациям, что обеспечивало до недавнего времени достаточный уровень безопасности.

Входной блок данных, состоящий из 64 бит, преобразуется в выходной блок идентичной длины. В алгоритме широко используются перестановки битов текста после первоначальной перестановки на правую и левую половины длиной по 32 бита. Затем выполняются 16 раундов одинаковых действий.

Для реализации упомянутых действий вводится функция *f*, которая оперирует с 32-разрядными словами исходного текста (*A*) и использует в качестве параметра 48-разрядный ключ (*J*; в каждом из фиксированного множества преобразований биты ключа сдвигаются и затем из 56 битов ключа выбираются 48).

Схема работы функции *f* показана на рис. 5.4.

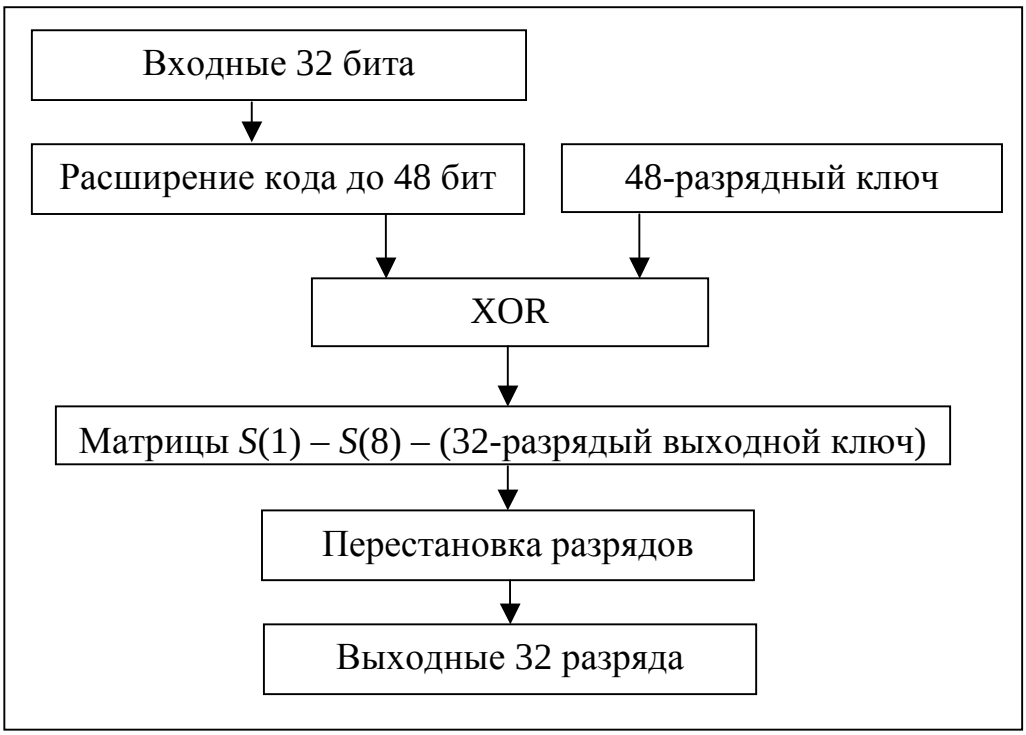

Рис. 5.4. Схема реализации функции *f*

Сначала 32 входных разряда расширяются до 48, при этом некоторые разряды повторяются. Результирующий 48-разрядный код преобразуется в 32-разрядный с помощью *S*-матриц. На выходе *S*-матриц осуществляется перестановка символов, согласно рекомендуемой схеме перестановок.

Преобразование начинается с перестановки (пермутации) бит в 64-разрядном блоке исходных данных: 58-й бит становится первым, 50-й – вторым и т. д.

Полученный блок делится на две 32-разрядные части: L_0 и R_0. Далее 16 раз (раундов) повторяются 4 процедуры в соответствии с рис. 5.5.

Пусть $A = L_i$, а J – преобразованный ключ. С помощью функции $f(A, J)$ генерируется 32-разрядный выходной код. Выполняется операция XOR для R_i $f(A, J)$, результат обозначается R_{i+1}. Осуществляется операция присвоения: $L_{i+1} = R_i$.

Преобразования ключей $K_n(n = 1, \ldots,16; K_n = KS(n, key)$, где n – номер итерации) выполняется согласно алгоритму, показанному на рис. 5.6.

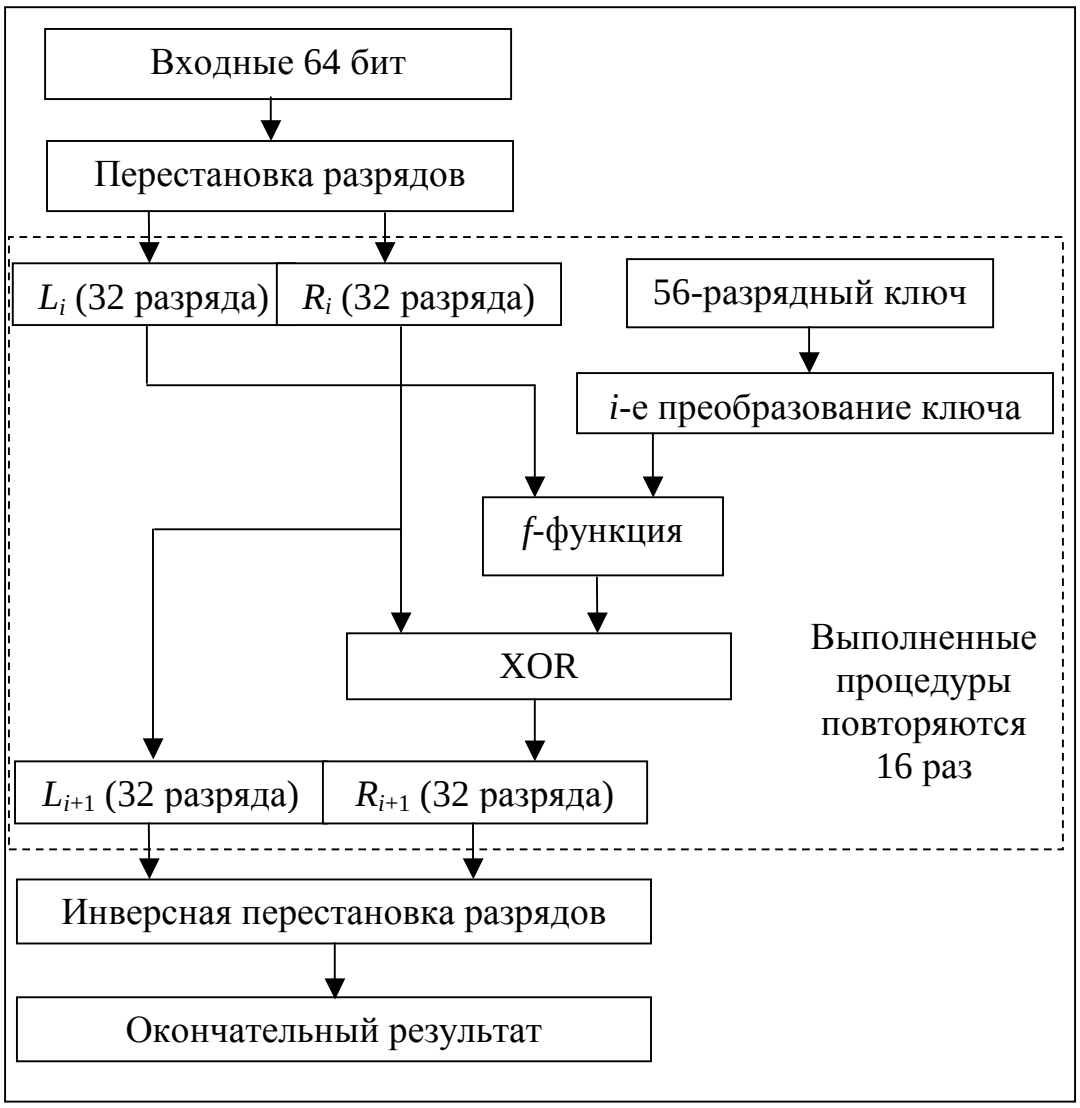

Рис. 5.5. Структурная схема реализации алгоритма DEA

Для описания алгоритма вычисления ключей K_n (функция *KS*) достаточно определить структуру «Выбора 1» и «Выбора 2», а также задать схему сдвигов влево. «Выбор 1» и «Выбор 2» представляют собой перестановки битов ключа (PC-1 и PC-2; табл. 5.1). При необходимости биты 8, 16, …, 64 могут использоваться для контроля четности.

Для вычисления очередного значения ключа таблица делится на две части: C_0 и D_0. В часть C_0 войдут биты 57, 49, 41, …, 44 и 36, а в D_0 – 63, 55, 47, …, 12 и 4. Так как схема сдвигов задана (табл. 5.2), C_1,nD_1, C_n, D_n и так далее могут быть легко получены из C_0 и D_0. Так, например, C_3 и D_3 получаются из C_2 и D_2 циклическим сдвигом влево на 2 разряда.

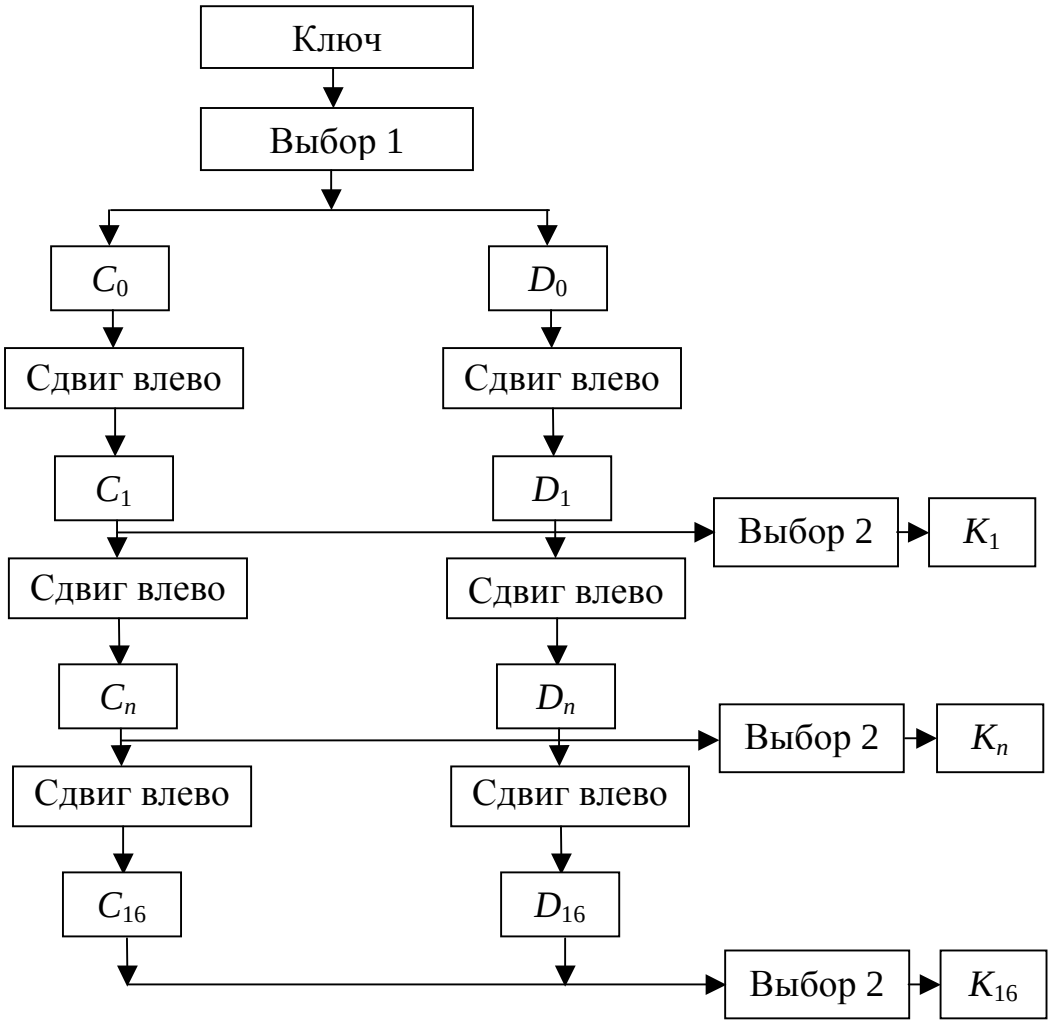

Рис. 5.6. Алгоритм вычисления последовательности ключей K_n

Таблица 5.1

Схема перестановки битов ключа

PC-1 (Выбор 1)	PC-2 (Выбор 2)
57 49 41 33 25 17 9	14 17 11 24 1 5
1 58 50 42 34 26 18	3 28 15 6 21 10
10 2 59 51 43 35 27	23 19 12 4 26 8
19 11 3 60 52 44 36	16 7 27 20 13 2
63 55 47 39 31 23 15	41 52 31 37 47 55
7 62 54 46 38 30 22	30 40 51 45 33 48
14 6 61 53 45 37 29	44 49 39 56 34 53
21 13 5 28 20 12 4	46 42 50 36 29 32

После всех подстановок, перестановок, операций XOR, циклических сдвигов можно подумать, что алгоритм расшифрования сложен и должен отличаться от алгоритма зашифрования. Напротив,

различные компоненты DES подобраны так, чтобы выполнялось очень полезное свойство: для зашифрования и расшифрования используется один и тот же алгоритм.

Таблица 5.2

Схема реализации сдвигов

Номер итерации	Число сдвигов влево
1	1
2	1
3	2
4	2
5	2
6	2
7	2
8	2
9	1
10	2
11	2
12	2
13	2
14	2
15	2
16	1

В последнее время принято считать, что алгоритм DES не обладает достаточной криптоскойкостью.

5.3.2. Криптографические системы с открытым (публичным) ключом. Асимметричное шифрование. Математическое решение проблемы перехода на асимметричное шифрование нашли в 1976 г. Диффи (Diffie) и Хеллман (Hellman)[2]. Основное отличие асимметричных систем от симметричных заключается в том, что ключ для зашифрования отличается от ключа для расшифрования сообщений ($K_1 \neq K_2$). Более того, ключ для расшифрования не может быть вычислен из ключа зашифрования.

После 1976 г. было создано множество криптоалгоритмов, использующих концепцию открытых ключей. И лишь небольшая их часть пригодна для практического применения. Как правило, такие алгоритмы основываются на решении одной из трудных математических задач, которая относится к так называемой *проблеме*

[2] Diffie, W. New Directions in Cryptography / W. Diffie, L. M. Hellman // IEEE Trans. on Inform Theory. – 1976. – V.IT–22. – P. 644–654.

дискретного логарифмирования (см. с. 68): если, к примеру, известны целые числа *a*, *x* и *n*, то можно легко вычислить значение функции $y = a^x \bmod n$, однако при известных *y*, *a* и *n* нахождение *x* может вызвать затруднение; при этом очевидно, что определение *x* основывается на операции *логарифмирования*.

Основой асимметричных систем является *модулярная арифметика*, т. е. вычисления по модулю. Общая задача состоит в определении такого значения *x*, при котором

$$a^x \bmod n = 1. \qquad (5.2)$$

Последнее равенство можно также записать в виде:

$$a^{-1} \equiv x \bmod n. \qquad (5.3)$$

В (5.3) параметр a^{-1} называют *обратным значением*. Задача вычисления обратных значений намного сложнее обычной. Рассмотрим это на примере.

Пример 5.5. Перепишем (5.2) в следующем виде:

$$4 \cdot x = 1 \bmod 7.$$

Решение этого уравнения эквивалентно поиску таких значений *x* и *k*, при которых справедливо:

$$4 \cdot x = 7 \cdot k + 1,$$

где *x* и *k* – целые числа.

Иногда задача обратных вычислений имеет решение, иногда – нет. Например, обратное значение 5 по модулю 14 равно 3. С другой стороны, число 2 не имеет обратного значения по модулю 14.

В общем случае уравнение (5.3) имеет единственное решение, если *a* и *n* взаимно просты. Если же *a* и *n* не являются взаимно простыми, то (5.3) решений не имеет.

Следует также иметь в виду, что *если число является простым, то любое число от 1 до n – 1 взаимно простое с n, имеет только одно обратное значение по модулю n.*

Обратное значение по модулю *n* можно вычислить на основе двух методов: *алгоритма Евклида* (или расширенного алгоритма Евклида) и *функции Эйлера*.

Алгоритм Евклида на языке C++ представлен в листинге 5.1.

```
#define isEven (x) ((x & 0x01) == 0)
#define isOdd (x) (x & 0x01)
#define swap (x,y) (x^= y, y^= x, x^= y)
Void ExtBinEuclid (int *u, int *v, int *u1, int *u2, int *u3) {
```

```
                          //предупреждение: u и v будут
                          //переставлены, если u<v
int k, t1, t2, t3;
    if (*u < *v) swap (*u, *v) ;
    for (k = 0; isEven (*u) && isEven (*v); ++k {
       *u >>= 1; *v >>= 1;
    }
    *u1 = 1; *u2 = 0; *u3 = *u; t1 = *v; t2 = *u-1; t3 = *v;
    do {
      do {
          if (isEven (*u3)) {
              if (isOdd (*u1) || isOdd (*u2)) {
                  *u1 += *v; *u2 += *u;
                  }
                  *u1 >>= 1; *u2 >>= 1; *u3 >>= 1;
          }
          if (isEven (t3) || *u3 < t3 ) {
                  swap (*u1, t1); swap (*u2, t2); swap (*u3,
                  t3);
          }
      } while (isEven (*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;
}
main (int argc, char **argv) {
int a, b, gcd;
    if (argc < 3) {
        cerr << "Использование: xeuclid u v" << endl;
        }
        int u = atoi (argv [1] );
        int v = atoi (argv [2] );
        if (u <= 0 || v <= 0 ) {
            cerr << "Аргументы должны быть положительными! " <<
            endl;
            return -2;
        }
                          //предупреждение: u и v будут
                          //переставлены, если u<v
        ExtBinEuclid (&u, &v, &a, &b, &gcd);
        cout << a <<" * " << u << " + (-"
             << b << ") * " << v << " = " << gcd << endl;
        if (gcd == 1)
             cout << "Обратное значение" << v << " mod " << u
             << " равно
        "
                  << u - b << endl;
        return 0;}
```

Листинг 5.1. Программная реализация алгоритма Евклида

Функция Эйлера, которую иногда называют функцией «фи» Эйлера и записывают как $\varphi(n)$, указывает количество положительных целых чисел, меньших n и взаимно простых с n для любого n, большего 1.

Если n – простое число, то $\varphi(n) = n - 1$. При $n = p \cdot q$, где p и q – простые числа, то $\varphi(n) = (p-1) \cdot (q-1)$. Эти числа используются в некоторых системах с открытым ключом, в том числе и в наиболее известной и широко используемой системе RSA.

5.3.3. Криптографический алгоритм RSA. Из всех предложенных алгоритмов для асимметричных систем алгоритм RSA проще всего понять и реализовать. Названный в честь трех изобретателей Рона Ривеста (Ron Rivest), Ади Шамира (Adi Shamir) и Леонарда Адлемана (Leonard Adleman), этот алгоритм многие годы противостоит многочисленным попыткам криптоаналитического вскрытия.

Безопасность алгоритма основана на трудоемкости разложения на множители больших чисел. Открытый и закрытый ключи являются функциями двух больших простых чисел разрядностью 100–200 десятичных цифр. Предполагается, что *восстановление открытого текста по шифртексту и открытому ключу равносильно разложению числа на два больших простых множителя*.

Ключ состоит из тройки больших целых чисел e, d, n. Пара чисел (e и n) является не секретной и образует *публичный ключ*. Число d является секретным, и пара (d и n) образует *тайный ключ*, известный только данному пользователю. Проблема верификации пользователей на основе их открытых ключей является одной из важных.

Основные операции алгоритма:

1. Для генерации двух ключей применяются два больших случайных простых числа (p и q). Для большей криптостойкости алгоритма эти числа должны иметь равную длину.

2. Рассчитывается произведение $n = p \cdot q$ и вычисляется функция $\varphi(n) = (p-1) \cdot (q-1)$.

3. Случайным образом выбирается число e такое, что e и $\varphi(n)$ являются взаимно простыми числами.

4. С помощью расширенного алгоритма Евклида вычисляется число d такое, что $e \cdot d = 1 \bmod \varphi(n)$, другими словами,

$$d = e^{-1} \bmod \varphi(n).$$

Подразумевается, что эти шаги выполняет лицо, которое генерирует для себя (или по просьбе другого лица для этого другого лица) соответствующие ключи.

Ниже приведен фрагмент программы (листинг 5.2), позволяющей найти все простые числа из диапазона (3, 100).

```
int numbers[20];
int a=0;
int flag=0;
for (i=3;i<100;i++)
                        //определяется массив простых чисел
{
for (j=2;j<i;j++)       //проверка: является ли элемент
                        //простым
  {                     //поиск значения, на которое он
                        //не делится
    ost=i%j;            //в переменную ost записывается
                        //остаток от деления
     if (ost==0)        //если он равен 0,
     {                  //то этот элемент не простой
       flag++;          //флаг = 1
          j=i;
     }
  }
if (flag==0)            //если оказалось, что у этого
                        //элемента нет делителей,
 {
   numbers[a]=i;        //значение записывается в массив
   a++;
 }
flag=0;                 //обнуление флага и переход к
                        //следующему элементу
}
//Найденные числа хранятся в массиве numbers[].
   cout<<"Массив простых чисел:\n";
   for (a=0;a<20;a++)
   cout<<numbers[a]<<"\t";
   flag=0;
   cout<<"\n";
   cout<<"\nВыберите из массива два простых числа\n";
                        //выбор из массива numbers[] двух
                        взаимно простых чисел p и q
   while (flag!=1)      //ввод p до тех пор, пока вводимое
                        //значение p не совпадет с одним из
                        //элементов массива простых чисел
    {
      cout<<"p = ";
      cin>>p;
      for (a=0;a<20;a++)
      {
        if (p==numbers[a])
        {
           flag=1;
         }
             }
      }
      flag=0;
      while (flag!=1)
```

```
                //ввод q до тех пор, пока вводимое
                //значение q не совпадет с одним из
                элементов массива простых чисел
{cout<<"q = ";
  cin>>q;
  for (a=0;a<20;a++)
   {
     if ((q==numbers[a])&&(q!=p))
                //при этом p и q не должны быть
                //равны друг другу
     {
      flag=1;
       }
    }
}
cout<<"\n";
long FEl=1;
                // вычисление функции Эйлера
FEl=(p - 1)*(q - 1);
cout<<"n = "<<n<<"\n";
cout<<"Funkcija Eilera = "<<FEl<<"\n";
i=2;
long d;         // поиск d
ost=0;
while (i!=FEl)
{
   ost=FEl%i;   //d и функция Эйлера должны
                //быть взаимно простыми
   if (ost!=0)
      {
       d=i;
       i=FEl - 1;
      }
 i++;
}
cout<<"d = "<<d<<"\n";
flag=0;
long e=0;
while (flag!=1)
                //исходя из формулы (e*d)mod(FEl)=1
                //подбор e
{
 e++;           //перебор e
                //пока не выполнится следующее
                //равенство
  if (((e*d)%FEl)==1)                    flag=1;
}
cout<<"e = "<<e<<"\n";
cout<<"\nОткрытый ключ: "<<e<<"  "<<n<<"\n";
cout<<"Закрытый ключ: "<<d<<"  "<<n<<"\n";
```

Листинг 5.2. Формирование открытого и закрытого ключей

Отметим, что числа d и n также являются взаимно простыми. Открытый и закрытый ключи составляют вышеуказанные пары чисел: e и n, d и n соответственно.

Зашифрование сообщения $M = m_1 m_2 \ldots m_k$ (сообщение делится на k блоков) в соответствии с (5.1) выглядит просто:

$$c_i = (m_i)^e \bmod n. \qquad (5.4)$$

При расшифровании каждого блока m_i сообщения C производится следующая операция:

$$m_i = (c_i)^d \bmod n.$$

Точно также сообщение может быть зашифровано с помощью пары d и n и расшифровано – с помощью чисел e и n. Именно такой подход используется в системах электронной цифровой подписи.

Пример 5.6. Пусть сообщение M представляется числом: 688232. Предполагаем, что длина ключа составляет три десятичных цифры. Проиллюстрируем использование алгоритма RSA для передачи зашифрованного сообщения.

Выбираем числа $p = 47$ и $q = 71$. Имеем $n = p \cdot q = 47 \cdot 71 = 3337$. Вычисляем $\varphi(n) = (p-1) \cdot (q-1) = 46 \cdot 70 = 3220$.

Число e не должно иметь общих сомножителей с числом 3220; выбираем (случайным образом) e, равным 79.

Вычисляем d: $d = 79^{-1} \bmod 3220 = 1019$.

Имеем открытый ключ – числа 79 и 3337 (его можно разместить в общедоступных источниках) и закрытый ключ – числа 1019 и 3337 (как видим, *секретным является только число d*; в нашем случае – это число 1019).

Для зашифрования сообщения M разбиваем его на блоки длиной, равной длине ключа, т. е. по 3 символа: $m_1 = 688$, $m_2 = 232$. Первый блок шифруется как $688^{79} \bmod 3337 = 1570 = c_1$; второй блок – $232^{79} \bmod 3337 = 2756 = c_2$.

Шифртекст C сообщения M выглядит следующим образом: $C = 1570\ \ 2756$. Для обратного преобразования нужно выполнить похожие операции, однако с использованием числа 1019 в качестве степени:

$m_1 = 1570^{1019} \bmod 3337 = 688,$

$m_2 = 2756^{1019} \bmod 3337 = 232.$

5.4. Электронная цифровая подпись

Основываясь на идее *односторонней функции*, Диффи и Хеллман предложили структуру криптосистемы с открытыми ключами для сети со многими абонентами. Эта идея может быть применена для получения и использования электронной цифровой подписи (ЭЦП). ЭЦП выполняет те же функции, что и обычная «ручная» подпись:

– удостоверяет,

– подтверждает,

– идентифицирует.

Рассмотрим это поподробнее:

1) подпись должна быть достоверна, это означает, что подписавший документ человек сделал это осознанно, без принуждения;

2) подпись неподдельна: особа, подписавшая документ, является автором подписи, от которой нельзя отречься;

3) подпись невозможно использовать повторно, мошенник не должен иметь возможность переносить подпись без ведома подписавшегося.

5.4.1. Электронная цифровая подпись на основе симметричных криптосистем и посредника. Пусть имеется абонент *A*, который пересылает сообщение абоненту *B*. Основная проблема заключается в механизме передачи и механизме хранения ключа. Хранителем ключей является третья сторона – посредник *П*, которому абсолютно доверяют абоненты *A* и *B*.

Для организации пересылки сообщения от *A* к *B* абонент *A* пользуется услугой *П*. Схематично эта процедура выглядит следующим образом:

1. Абонент *A* шифрует с помощью собственного ключа K_A (ключи обоих абонентов известны *П*, однако каждый из абонентов не знает «чужих» ключей) сообщение *M*: $C = K_A(M)$.

2. Абонент *A* отсылает зашифрованное сообщение *П*. Посредник расшифровывает сообщение с помощью ключа K_A, что является подтверждением того, что документ прислан именно абонентом *A*.

3. Посредник *П* дополняет расшифрованное сообщение подтверждением того, что оно получено от *A*, и это дополнение (*D*) выполняет функцию ЭЦП. *П* шифрует это дополненное сообщение ключом *B*, т. е. K_B: $C = K_B(M, D)$, и пересылает его *B*.

4. Абонент *B* расшифровывает известным ему ключом полученное от посредника *П* сообщение *A*.

Участие *П* позволяет выполнить основные из предъявляемых к подписи *требования*:

− подпись действительно достоверна, поскольку посредник является абсолютно достоверным лицом;

− подпись неподдельна; если кто-то попытается выдать себя за абонента *A*, то это должен установить посредник (при условии, что абонент *A* соблюдает конфиденциальность своего ключа);

− подпись невозможно использовать повторно; если, к примеру, четвертая сторона захочет это сделать, то *A* и *B* легко это могут опровергнуть;

− невозможно изменить подпись с учетом того, что это можно легко обнаружить;

− подпись невозможно отрицать: вступает в силу принцип мажоритарности.

Основная проблема реализации рассмотренного метода – хранение ключей множества пользователей.

5.4.2. Электронная цифровая подпись на основе асимметричных криптосистем. Р. Ривест разработал алгоритм и метод генерации ЭЦП на основе алгоритма RSA. Данный метод предполагает, что для обмена подписанными сообщениями *A* и *B* должны сгенерировать (см. предыдущий подраздел) личные ключи: E_A, D_A, N_A; E_B, D_B, N_B соответственно. Для обмена подписанными сообщениями каждая из сторон использует свой тайный ключ для зашифрования сообщения. В свою очередь, другая сторона применяет публичный ключ отправителя для расшифрования сообщения.

5.4.3. Электронная цифровая подпись на основе асимметричных криптосистем и однонаправленных хэш-функций. В августе 1991 г. в США был разработан и утвержден стандарт цифровой подписи DSS (Digital Signature Standard), основанный на алгоритме DSA (Digital Signature Algorythm). Этот стандарт предусматривает использование принципов асимметричных систем и однонаправленных хэш-функций (hash-functions): $H(M)$.

Хэш-функции. В математике и информатике применяются такие функции, которые принимают на входе строку данных произвольной длины и преобразовывают ее в строку меньшей длины. Они называются значением *хэш-функции*, или дайджестом (digest). Функция, которая используется для преобразования, должна быть односторонней, т. е. значение функции не является «ключом» для определения ее аргумента.

Как правило, хэш-функция является *конкатенацией* вычисленных определенным образом составных частей или блоков:

$$H = h_1 \| h_2 \| ... \| h_i \|.$$

В современных системах i меняется от 3 до 6–7, причем расчет каждого последующего блока хэш-функции зависит не только от данных, но и от значений предыдущих блоков:

$$h_j = F(M, h_{j-1}),$$

где j меняется от 2 до i.

На практике однонаправленность функции контролируется на основе принципа сжатия (*сжимающей функции*). Принцип сжимаемости функции означает, что рассматриваются данные не целиком, а по частям.

Наиболее известными алгоритмами вычисления хэш-функций являются алгоритмы класса MD (Message Digest): MD2, MD4, MD5 и алгоритмы класса SHA (Secure Hash Algorythm).

Особенности алгоритма MD4. Алгоритм MD4 разработан Роном Ривестом (Ron Rivest). Функция возвращает 128-разрядное двоичное значение, вычисленное на основе исходного сообщения произвольной длины.

Как формируются эти 128 бит? Сообщение M должно быть предварительно преобразовано для того, чтобы длина входной строки (двоичной последовательности длиной k символов) равнялась 0 по модулю 512. Для этого к k символам должны быть добавлены двоичные символы (M') длины r и 64 бита: $(k + r + 64) \bmod 512 = 0$, где r является двоичным представлением числа k (например, при $k = 1024$ бита $r = 10$), а 64 – двоичное число, состоящее из одной единицы и 63 нулей.

Общая длина сообщения (M, M') разбивается на блоки по 512 бит, т. е. получается m блоков по 512 бит. В свою очередь, каждый из блоков разбивается еще на 16 блоков по 32 бита. Следовательно, имеем: $512 \cdot m = (16 \cdot 32) \cdot m$ бит.

Длина, соответствующая каждой части хэш-функции (h_j), также составляет 32 бита, и чтобы получить функцию длиной 128 бит, нужно подсоединить 4 такие части. Эти части формируются на основе общего алгоритма, включающего 3 так называемых раунда.

В каждом раунде выполняется 16 различных операций, каждая операция основывается на использовании известных нелинейных функций: функции F (в первом раунде), G (во втором) и H (в третьем) – над тремя переменными из массива a, b, c, d.

Алгоритм MD5 является модификацией алгоритма MD4 с целью повышения криптостойкости. Общая схема реализации алгоритма MD5 показана на рис. 5.7.

Хэш-функция алгоритма SHA имеет длину 160 бит и реализуется за 5 раундов операций над словами длиной 32 бита $(32 \cdot 5 = 160)$.

Из всех хэш-функций SHA1 является наиболее криптостойкой.

ЭЦП на основе хэш-функции генерируются абонентом *A* и используется абонентом *B* следующим образом. Абонент *A* производит вычисление хэш-функции, применяя оговоренный алгоритм. Затем, используя свой тайный ключ (D_A, N_A), шифрует хэш-функцию $D_A(H(M))$ и присоединяет зашифрованный хэш к передаваемому сообщению *M*, т. е. абонент *B* получает сообщение *M* с присоединением зашифрованной тайным ключом *A* ЭЦП: *M*, $D_A(H(M))$.

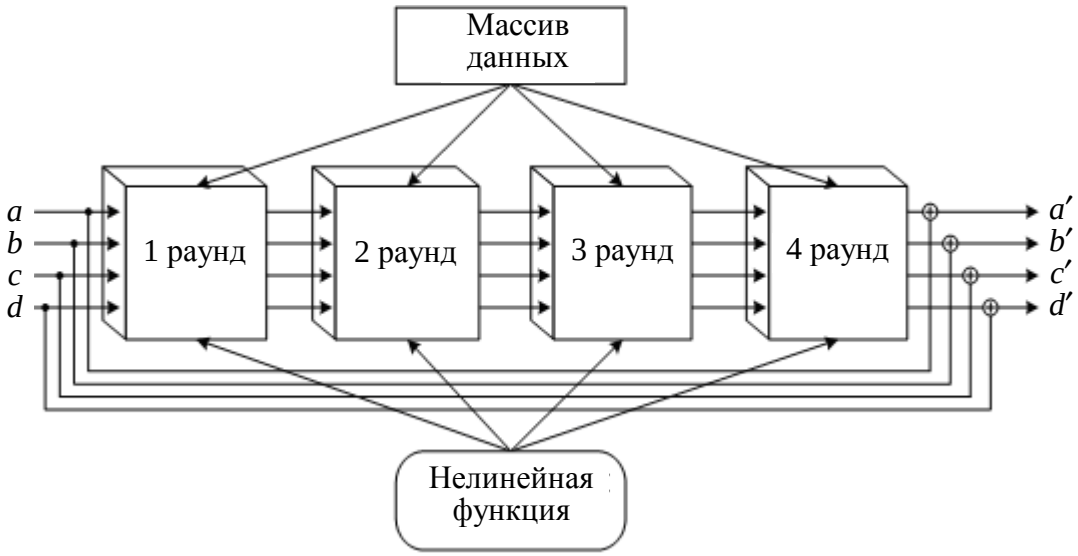

Рис. 5.7. Общая схема реализации алгоритма MD5

Получив подписанное сообщение *M*, абонент *B* вычисляет по известному алгоритму значение хэш-функции (обозначим его $H'(M)$) полученного сообщения. Кроме того, *B* расшифровывает зашифрованную ЭЦП ($D_A(H(M))$), используя публичный ключ абонента *A* $E_A(D_A(H(M))) = H(M)$ и далее производит сравнение хэш-функций $H(M)$ и $H'(M)$. При их совпадении делается заключение о том, что полученный документ в процессе передачи не претерпел изменений и его автором действительно является абонент *A*.

Сообщение с присоединенной к нему ЭЦП (M, $D_A(H(M))$) может быть дополнительно зашифровано по симметричному или асимметричному алгоритмам.

Вопросы для самоконтроля

1. Поясните сущность криптографических методов преобразования информации.
2. Дайте определения следующим понятиям:
а) конфиденциальность,
б) авторизация,
в) аутентификация,
г) зашифрование,
д) расшифрование,
е) ключ.
3. В чем состоят основные различия методов симметричного и асимметричного криптопреобразования?
4. Подстановочные и перестановочные шифры. Приведите примеры собственных алгоритмов.
5. В чем состоит основное отличие блочных и потоковых шифров?
6. Зашифруйте шифром Цезаря свои фамилию и имя.
7. Поясните сущность алгоритма DES.
8. Как вы понимаете проблему дискретного логарифма?
9. Объясните сущность алгоритма RSA. Приведите конкретный пример.
10. Поясните сущность основных методов вычисления обратных значений по модулю n.
11. Что такое ЭЦП и для чего она используется?
12. Охарактеризуйте основные методы генерации ЭЦП.
13. Что такое хэш-функция и в чем ее особенности?
14. Охарактеризуйте алгоритм использования ЭЦП на основе хэш-функции.
15. Представьте в формализованной форме передачу подписанного и зашифрованного сообщений на основе симметричного и асимметричного шифрования.

Задание для самостоятельного выполнения

Создайте приложение, с помощью которого покажите пример передачи зашифрованных собственных фамилии, имени и отчества, полагая, что букве «а» соответствует число 01, букве «б» – 02 и т. д. Для преобразования применяйте алгоритм RSA, длина ключа устанавливается по указанию преподавателя. В качестве базового алфавита используйте русский алфавит (с учетом регистра символа).

ЛИТЕРАТУРА

1. Скляр, Б. Цифровая связь. Теоретические основы и практическое применение / Б. Скляр; пер. с англ. – М.: Издательский дом «Вильямс», 2003. – 1104 с.

2. Мак-Вильямс, Ф. Дж. Теория кодов, исправляющих ошибки / Ф. Дж. Мак-Вильямс, Н. Слоэн; пер. с англ. – М.: Связь, 1979. – 744 с.

3. Урбанович, П. П. Избыточность в полупроводниковых интегральных микросхемах памяти / П. П. Урбанович, В. Ф. Алексеев, Е. А. Верниковский. – Минск: Навука і тэхніка, 1994. – 320 с.

4. Щербаков, Н. С. Достоверность работы цифровых устройств / Н. С. Щербаков. – М.: Машиностроение, 1989. – 224 с.

5. Черкесов, Г. Н. Надежность аппаратно-программных комплексов: учеб. пособие / Г. Н. Черкесов. – СПб.: Питер, 2005. – 479 с.

6. Хофман, Л. Современные методы защиты информации / Л. Хофман; под ред. В. А. Герасименко; пер. с англ. – М.: Сов. радио, 1980. – 264 с.

7. Леонов, А. П. Безопасность автоматизированных банковских и офисных систем / А. П. Леонов, К. А. Леонов, Г. В. Фролов. – Минск: НКП Беларуси, 1996. – 280 с.

8. Шнайдер, Б. Прикладная криптология. Протоколы, алгоритмы, исходные тексты на языке Си / Б. Шнайдер. – М.: ТРИУМФ, 2003. – 816 с.

9. Молдовян, Н. А. Введение в криптосистемы с открытым ключом / Н. А. Молдовян, А. А. Молдовян. – СПб.: БХВ-Петербург, 2005. – 288 с.

10. Молдовян, Н. А. Криптография: от примитивов к синтезу алгоритмов / Н. А. Молдовян, А. А. Молдовян, М. А. Еремеев. – СПб.: БХВ-Петербург, 2004. – 446 с.

11. Харин, Ю. С. Математические основы криптологии / Ю. С. Харин, В. И. Берник, Г. В. Матвеев. – Минск: БГУ, 1999. – 319 с.

12. Математические и компьютерные основы криптологии / Ю. С. Харин и [др.]. – Минск: Новое знание, 2003. – 381 с.

13. Ховард, М. Защищенный код / М. Ховард, Д. Лебланк; пер. с англ. – 2-е изд., испр. – М.: Издательский дом «Русская Редакция», 2005. – 704 с.

СОДЕРЖАНИЕ

Учебное издание

Урбанович Павел Павлович
Романенко Дмитрий Михайлович
Романцевич Елена Владимировна

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ И НАДЕЖНОСТЬ СИСТЕМ

Учебно-методическое пособие

Редактор *Е. С. Ватеичкина*
Компьютерная верстка *О. Ю. Шантарович*

Подписано в печать 20.08.2007. Формат 60×84$^{1}/_{16}$.
Бумага офсетная. Гарнитура Таймс. Печать офсетная.
Усл. печ. л. 5,2. Уч.-изд. л. 5,4.
Тираж 70 экз. Заказ .

Учреждение образования
«Белорусский государственный технологический университет».
220050. Минск, Свердлова, 13а.
ЛИ № 02330/0133255 от 30.04.2004.

Отпечатано в лаборатории полиграфии учреждения образования
«Белорусский государственный технологический университет».
220050. Минск, Свердлова, 13.
ЛП № 02330/0056739 от 22.01.2004.
Учреждение образования
«БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ»

П. П. Урбанович, Д. М. Романенко, Е. В. Романцевич

ИНФОРМАЦИОННАЯ БЕЗОПАСНОСТЬ И НАДЕЖНОСТЬ СИСТЕМ

Минск БГТУ 2007