Министерство образования и науки Украины
Черниговский государственный технологический университет

БЕЗОПАСНОСТЬ ИНФОРМАЦИИ

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
для самостоятельной работы
и для выполнения лабораторных работ
по дисциплине
“БЕЗОПАСНОСТЬ ПРОГРАММ И ДАННЫХ ”
для студентов направления подготовки
6.050102 “Компьютерная инженерия” и
6.050103 “Программная инженерия”

Обсуждено и рекомендовано
на заседании кафедры
информационных и компьютерных систем
Протокол № 10 от 18 апреля 2013 г.

Чернигов ЧГТУ 2013

Безпека інформації. Методичні вказівки для самостійної роботи та для виконання лабораторних робіт з дисципліни “Безпека програм та даних” для студентів за напрямом підготовки 6.050102 “Комп’ютерна інженерія” та 6.050103 “Програмна інженерія” / Укладачі: Соломаха В.В., Тевкун М.В. – Чернігів: ЧДТУ, 2013. - 106 с. Рос. мовою.

Составители: Соломаха Валерий Владимирович, старший преподаватель,
Тевкун Мария Вадимовна, аспирант

Ответственный за выпуск: Казимир В.В., заведующий кафедрой информационных и компьютерных систем, доктор технических наук, профессор

Рецензент: Нестеренко С.О., кандидат технических наук, доцент кафедры информационных и компьютерных систем Черниговского государственного технологического университета

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

На сегодняшний день успешная работа предприятий и организаций зависит все в большей мере от информационных ресурсов. Ценность информации определяется её достоверностью, актуальностью и конфиденциальностью. Широкое применение компьютерных средств в человеческой деятельности привело к возникновению важной задачи обеспечения эффективной эксплуатации систем хранения и обработки информации. На первый план выходит вопрос защиты информации от несанкционированных воздействий.

Курс «Безопасность и защита информации в компьютерных сетях» ставит своей целью получение студентами теоретических знаний, практических навыков в области решения задачи защиты информации в составе компьютерных информационных систем. Студентам предлагается изучить как классические, так и современные методы и средства, применяемые для защиты информации.

Применение компьютерной системы изучения методов и средств аппаратно-программной защиты информации, далее компьютерной системы, позволяет повысить эффективность обучения за счет графического представления модели информационной системы с учетом воздействий угроз нарушения безопасности, возможности изучения имитационной модели средств и методов защиты информации. Система позволяет исследовать криптографические алгоритмы и протоколы, формальные политики безопасности, является настраиваемой и расширяемой за счет возможности редактирования элементов информационной системы, подсистемы защиты и внесения блоков, реализующих новые методы и средства защиты.

Для закрепления теоретических сведений по дисциплине предназначен цикл лабораторных работ. методические указания к которым приводятся ниже.

1 ЛАБОРАТОРНАЯ РАБОТА № 1

АНАЛИЗ ТРЕБОВАНИЙ БЕЗОПАСНОСТИ

1.1 Теоретические сведения

Для достижения эффективной защиты информации требуется правильно спроектировать подсистему защиты и применить соответствующие методы и средства защиты. Правильность проектирования определяется выполнением всех требований к подсистеме защиты, поэтому на начальном этапе необходимо сформулировать требования безопасности к системе защиты. Затем на основе этих требований выбираются и разрабатываются решения, применяемые для защиты информации.

Результатом анализа требований является перечень ресурсов, угрозы и уязвимости возникающие в системе, требования к применяемым механизмам защиты, предварительная оценка риска и достигаемой защищенности.

Для изучения процесса анализа используется математическая модель информационной системы. Модель информационной системы расширена моделями элементов, используемыми при построении средств защиты информации: угрозы, средства защиты и др.

Математическая модель представляет статическое состояние и описывает следующие элементы:

- информационные ресурсы;
- активные процессы, выполняющие обработку и доступ к данным;
- уязвимости, присутствующие в информационной системе;
- угрозы нарушения безопасности;
- средства и методы защиты информации.

1.2 Анализ информационной системы

Для выбора механизмов защиты информации требуется в первую очередь выделить ресурсы информационной системы, а также внешние элементы, с которыми осуществляется взаимодействие.

Ресурсами могут быть средства вычислительной техники, программное обеспечение, данные в информационной системе. В качестве внешних элементов выступают внешние сервисы, сетевые ресурсы.

Доступ к ресурсам информационной системы выполняют субъекты. В качестве субъектов выступают программные процессы, пользователи системы, другие системы, которые используют внешнюю информацию.

1.2.1 Угрозы нарушения безопасности

Проведение анализа риска и формулировка требований к системе защиты производятся на основе информации о возможных угрозах и их характеристик.

По природе возникновения выделяют угрозы:

а) естественные;

б) искусственные угрозы, которые в свою очередь делятся на:

- непреднамеренные;
- преднамеренные.

По виду угрозы делят на:

- стихийные бедствия и аварии;
- сбои и отказы оборудования;
- последствия ошибок проектирования и разработки компонентов информационной системы;
- ошибки эксплуатации;
- действия нарушителей и злоумышленников.

Естественные угрозы вызваны воздействиями стихийными бедствиями и авариями, независящими от человека.

Искусственные угрозы вызваны деятельностью человека. Преднамеренные угрозы отличаются о непреднамеренных корыстными целями нарушителя.

Непреднамеренные угрозы являются следствиями непреднамеренных ошибок проектирования, разработки, эксплуатации.

В зависимости от источника угрозы делят на:

- внутренние;
- внешние.

Различают такие виды угроз: угроза нарушения конфиденциальности информации, целостности информации, отказа в обслуживании (нарушения доступности).

При воздействии угроз нарушения защиты нарушается нормальный поток информации (рисунок 1.1).

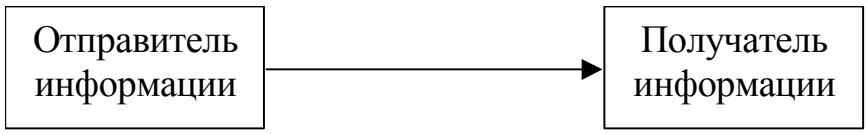


Рисунок 1.1 — Нормальный поток информации

При анализе воздействия нарушений следует различать такие виды угроз:

- прерывание (рисунок 1.2);
- перехват (рисунок 1.3);
- модификация (рисунок 1.4);
- фальсификация (рисунок 1.5).

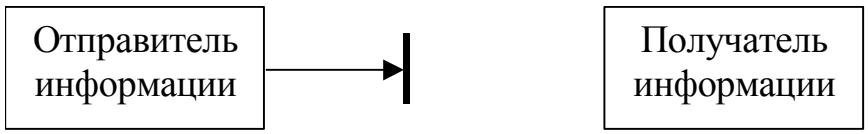


Рисунок 1.2 — Поток информации при воздействии угрозы прерывания

Угроза прерывания вызывает нарушение доступности информации.

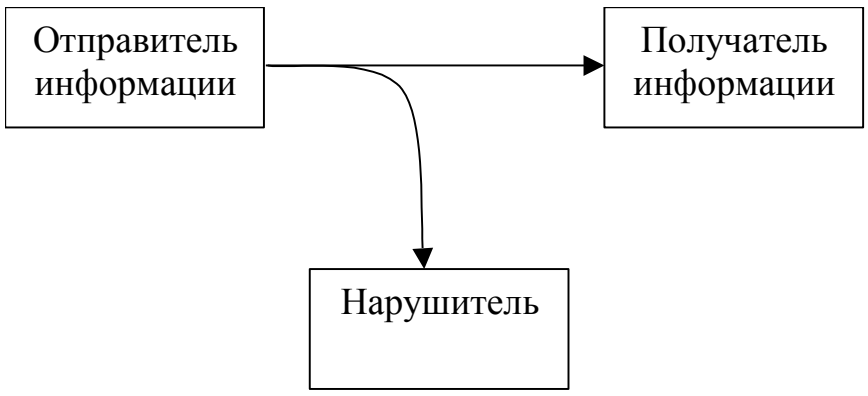


Рисунок 1.3 — Поток информации при воздействии угрозы перехвата

Угроза перехвата приводит к нарушению конфиденциальность информации.

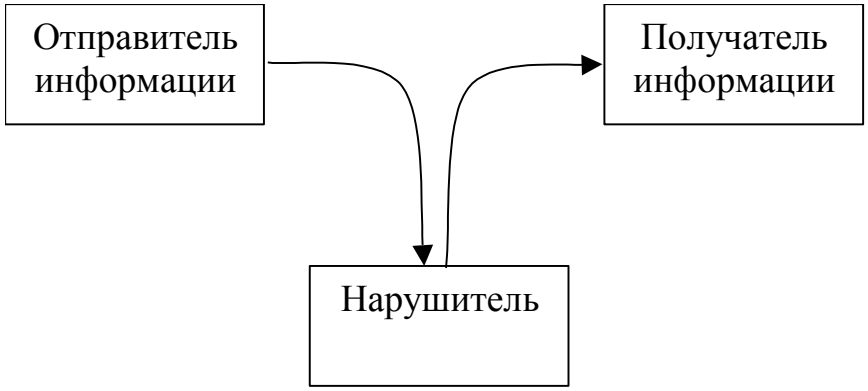


Рисунок 1.4 — Поток информации при воздействии угрозы модификации

При воздействии угрозы модификации нарушается конфиденциальность и целостность информации.

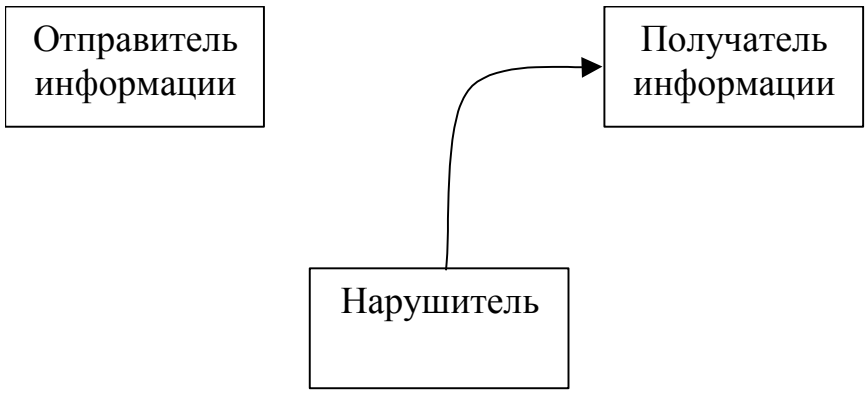


Рисунок 1.5 — Поток информации при воздействии угрозы фальсификации

Фальсификация приводит к нарушению аутентичности информации.

Выбор средств и методов защиты зависит от видов возникающих угроз.

Осуществление угроз возможно благодаря существованию уязвимостей в информационных системах.

Уязвимость — это любая характеристика информационной системы, использование которой может привести к реализации угрозы.

Существуют три пути возникновения уязвимостей программного и аппаратного обеспечения:

- ошибки реализации;
- ошибки проектирования;
- ошибки конфигурации.

Ошибки проектирования — наиболее серьезные и сложные в исправлении.

Ошибки конфигурации являются самыми распространенными и легче всего исправляются.

1.2.2 Методы и средства защиты информации

Для противодействия угрозам используется совокупность мер обеспечения безопасности:

- правовые или законодательные меры;
- морально-этические меры;
- административные меры;
- физические;
- аппаратно-программные меры.

К аппаратно-программным методам относят электронные устройства и программы, которые в комплексе позволяют противостоять указанным угрозам.

Различают два дополняющих друг друга подхода защиты информационных систем: формальное моделирование политики безопасности и криптографические методы защиты информации. Формальные модели безопасности предоставляют разработчикам систем основополагающие принципы, определяющие концепции построения систем.

Криптография даёт методы обеспечения конфиденциальности, целостности, ограничения доступа к данным.

Формальная модель безопасности является очень важной для получения безопасной системы, так как она определяет основные принципы и концепции обеспечения безопасности, используемые на уровне взаимодействия элементов системы.

Формальная модель строится на основе требований, предъявляемых к системе. Для теоретического подтверждения безопасности системы следует проводить формальное доказательство безопасности.

Криптография представляет собой совокупность алгоритмов и протоколов защиты информации.

Рассмотрим виды криптографических блоков, которые используются в системах защиты информации:

- идентификация;
- аутентификация;
- шифрование;
- контроль целостности.

Назначением идентификации является разграничение доступа пользователей или процессов к данным в системе.

Аутентификация используется для проверки подлинности идентифицированного пользователя.

Метод шифрования является основополагающим для построения защищенных систем, на его основе реализуется большинство других методов обеспечения безопасности.

Метод контроля целостности предназначен для противостояния угрозам нарушения целостности.

Защищенная система строится на основе применения всех видов криптографических методов, и только комплекс мер позволяет с требуемой уверенностью противостоять определенным угрозам.

В криптографии используются алгоритмы:

- симметричного шифрования данных;
- шифрования с открытым ключом;
- хэширования данных.

Алгоритм симметричного шифрования данных основан на том, что используется один секретный ключ, который передаётся по защищенному каналу отправителю и получателю.

В случае шифрования с открытым ключом не требуется передача по защищенному каналу секретного ключа.

Используется две пары ключей, шифрование производится с помощью открытого ключа, а обратное преобразование — с помощью секретного ключа.

Алгоритмы хэширования предназначены для получения сжатого образа данных, по которым нельзя восстановить данные, но можно проверить их целостность.

Ключи, используемые в криптографических алгоритмах, должны удовлетворять определенным требованиям, и для этого разработаны алгоритмы генерации ключей.

При рассмотрении современных решений требуется выяснить схему работы, основные возможности и характеристики, достоинства и недостатки средств и применяемых методов для защиты информации.

1.2.3 Анализ защищенности

Оценка защищенности является крайне важной при разработке подсистемы защиты информации, так как такая оценка позволяет выявить ошибки в конфигурации, опасные уязвимости, оценить риск нарушения доступа к информации, возможные потери при использовании различного рода средств и методов защиты информации.

Существуют различные подходы к оценке защищенности:

- статистический;
- метод минимизации затрат на средства защиты и суммы ущерба при успешном осуществлении угроз;
- неформальные методы.

Указанные выше подходы анализа защищенности различаются по эффективности, имеют существенные ограничения в применении.

Статистический подход основывается на вычислении рисков, связанных с успешной реализацией угроз.

Недостатком статистического подхода является сложность определения вероятностных характеристик событий в информационной системе, а также сложность оценки величин ущерба и степени сопротивляемости механизмов защиты.

Минимизация затрат позволяет разработать оптимальную систему защиты по суммарным затратам. В ряде случаев, когда безопасность информации имеет большее значение, нежели стоимость внедрение средств защиты нецелесообразно применять метод минимизации затрат.

Методы анализа рисков и минимизации стоимости являются формальными и позволяют на основе известных исходных данных о элементах системы определить точные значения рисков, затрат, защищенности.

Неформальные методы основаны на классификации и категорировании элементов в системе.

Подход не дает точных оценок защищенности, а позволяет определить категорию защищенности.

Методы, основанные на классификации, имеют более широкий круг применимости, чем формальные методы.

Для изучения процесса анализа защищенности применяются формальные методы.

Эти методы позволяют на основе модели системы выполнить анализ, расчет и сравнение защищенности.

Статическая модель системы, которая учитывает влияние угроз на элементы информационной системы, применение механизмов защиты, описывается следующими определениями:

а) система состоит из 6 множеств $I = < S, O, T, V, M, B, R >$;

б) $S = \{s_0, s_1, ..., s_n\}$ — множество субъектов;

в) $O = \{o_0, o_1, ..., o_m\}$ — множество ресурсов;

г) $T = \{t_0, t_1, ..., t_l\}$ — множество угроз;

д) $V = \{v_0, v_1, ..., v_u\}$ — множество уязвимостей;

е) $M = \{m_1, m_2, ..., m_h\}$ — множество механизмов защиты;

ж) $B = \{b_1, b_2, ..., b_g\}$ — множество барьеров;

з) $R = \{r_0, r_1, ..., r_k\}$ — множество отношений между элементами;

и) отношение $r = (s_i, o_j, t_k, v_l, m_x, b_y)$, принадлежащие множеству R, в качестве значений t_k, v_l, m_x, b_y могут иметь пустое значение $\varnothing$, значения s_i, o_j — не могут быть пустыми;

к) если в отношении $r = (s_i, o_j, t_k, v_l, m_x, b_y)$ $t_k \neq \varnothing$, то и $v_l \neq \varnothing$;

л) если в отношении $r = (s_i, o_j, t_k, v_l, m_x, b_y)$ $v_l \neq \varnothing$ и $m_x \neq \varnothing$, то и $b_y \neq \varnothing$, причем $b_y = (v_l, m_x)$.

Модель системы, которая включает описанные в разделе элементы, представлена на рисунке 1.6.

Барьер образуется применением средства защиты, которое препятствует воздействию угрозы.

Для достижения полного перекрытия угроз требуется, чтобы для каждой уязвимости v_l существовал барьер b_y такой, что $r = (s_i, o_j, t_k, v_l, m_x, b_y)$, $b_y = (v_l, m_x)$.

Для расчета требуется определить следующие характеристики элементов статической модели:

S — степень сопротивляемости механизма защиты;

P — вероятность возникновения угрозы;

L — величина ущерба при успешном осуществлении угрозы.

Степень сопротивляемости механизма защиты является обратной величиной для вероятности преодоления защиты T нарушителем

$$S = 1 - T$$

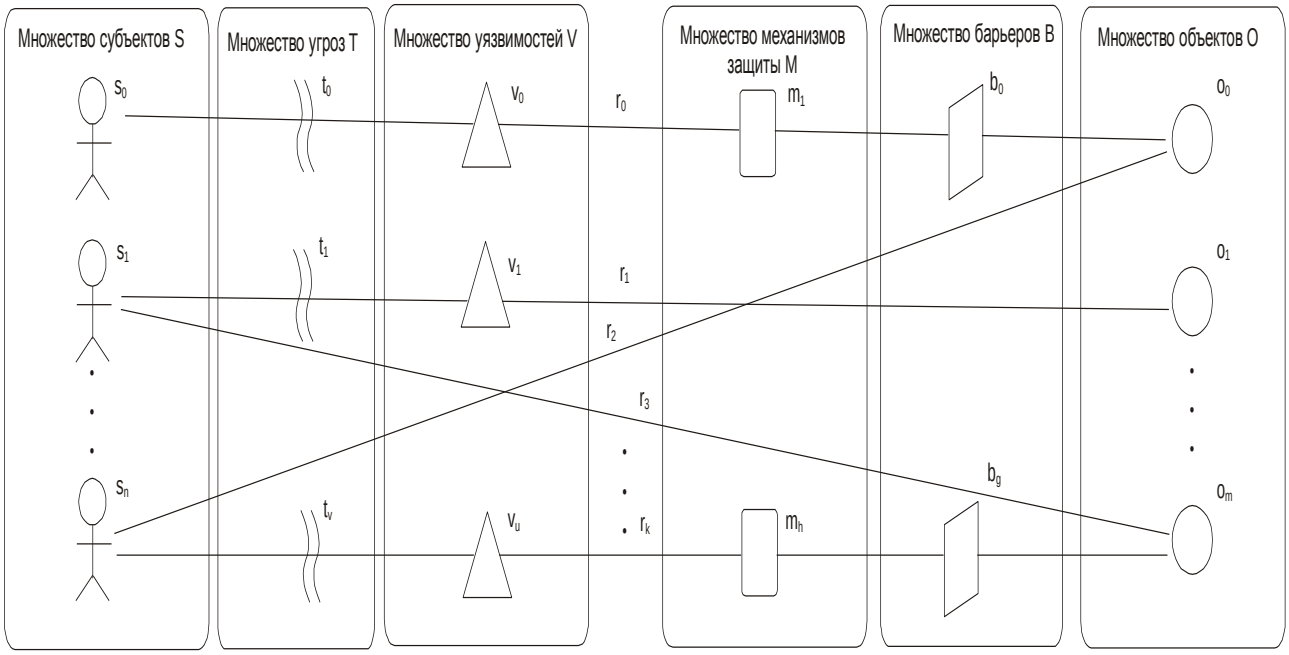


Рисунок 1.6 — Статическая модель информационной системы и снстемы защиты

Пусть информационный ресурс o_j при доступе субъекта s_i подвержен угрозе t_k.

Вероятность возникновения угрозы t_k обозначена P. Возникновение угрозы возможно благодаря наличию уязвимости v_l. Величина ущерба при удачном осуществлении угрозы L. Для исключения уязвимости применяется механизм защиты m_x со степенью сопротивляемости S. В результате при доступе к ресурсу уязвимость исключается барьером $b_y = (v_l, m_x)$.

Для расчета остаточного риска при доступе субъекта s_i к объекту o_j используется формула

$$Risk_i = PL(1 - S),$$

где $i = 0..k$ — номер отношения $r_i = (s_i, o_j, t_k, v_l, m_x, b_y) \in R$.

Если уязвимость не исключена механизмом защиты, степень сопротивляемости считается равной 0 и остаточный риск рассчитывается по формуле

$$Risk_i = PL$$

Остаточный риск для всей системы защиты состоит из рисков, связанных с возможностью осуществления всех угроз и вычисляется по формуле

$$Risk_\Sigma = \sum_{i=0..k} Risk_i ,$$

где $i = 0..k$ — номер отношения $r_i = (s_i, o_j, t_k, v_l, m_x, b_y) \in R$,

$Risk_i$ — остаточный риск при доступе субъекта s_i к ресурсу o_j.

Суммарный риск показывает вероятный ущерб при эксплуатации информационной системы с разработанными средствами защиты.

Степень защищенности системы — это величина обратная суммарному риску системы.

Стоимостной анализ позволяет сбалансировать уровень затрат на внедрение средств защиты с масштабом угроз.

Затраты на установку средств защиты относятся к убыткам и носят постоянный характер

$$C_\Sigma = \sum_{i=0..h} C_i ,$$

где $i = 0..h$ — номер средства защиты,

C_i — затраты на установку средства защиты m_i.

Убытки при использовании средств защиты и действии угроз суммируются

$$U = C_\Sigma + Risk_\Sigma ,$$

где C_Σ — затраты на установку средств защиты,

$Risk_\Sigma$ — суммарный риск.

Типичной является ситуация (рисунок 1.7), когда при небольших затратах на средства защиты, убытки будут большими из-за большой вероятности успешного осуществления угроз; при оптимальном значении затрат на систему безопасности достигается минимальные убытки; при дальнейшем увеличении затрат C_Σ суммарные убытки также увеличиваются.

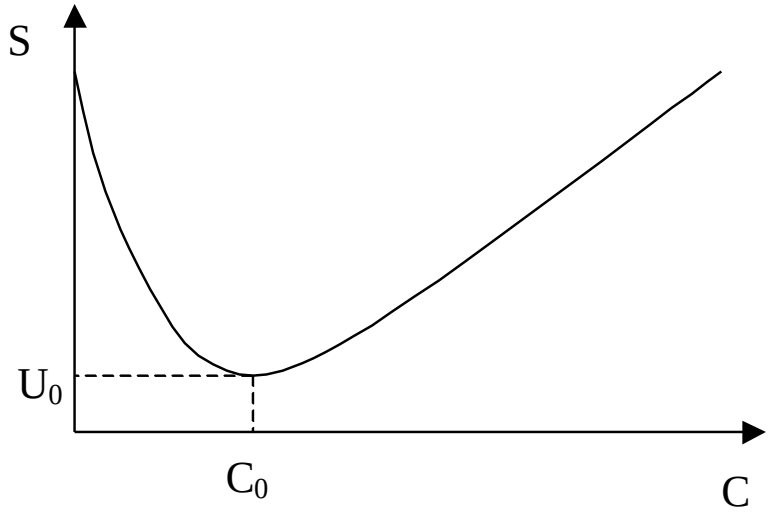


Рисунок 1.7 — Зависимость суммарных убытков при эксплуатации информационной системы от затрат на установку средств защиты

На графике C_0 — оптимальный уровень затрат на установку средств защиты, U_0 — наименьший уровень убытков, который достигается в точке C_0.

1.3 Применение компьютерной системы для анализа требований безопасности

В качестве примера рассмотрим упрощенную информационную систему предприятия.

В результате ознакомления с информационной системой выделены ресурсы:

- база данных предприятия, содержащая данные о клиентах, заказах, хозяйственных операциях и др.;
- файловый сервер предприятия, предназначенный для хранения и обмена конфиденциальной информацией.

Анализ пользователей системы показал разделение всего персонала на 2 вида:

- менеджеры;
- управляющий персонал (руководитель, главный бухгалтер).

Доступ к файловому серверу необходим только управляющему персоналу, причем следует учесть возможность обращения руководителя к

ресурсам база данных и файловый сервер через общедоступные глобальные сети.

Доступ менеджеров к базе данных осуществляется только внутри локальной сети предприятия.

При доступе субъектов менеджер и руководитель к ресурсам системы благодаря существованию разнообразных уязвимостей в программном и аппаратном обеспечении существует возможность реализации угроз нарушения доступа.

В таблице 1.1 представлены данные для проведения анализа защищенности системы.

Таблица 1.1 — Пример исходных данных для проведения анализа защищенности

Отношение доступа	**Тип угрозы**	**Вероятность возникновения угрозы**	**Величина ущерба**
менеджер – база данных	прерывание	0,01	100
руководитель – база данных	прерывание, перехват	0,09	1000
руководитель – файловый сервер	прерывание, перехват, фальсификация	0,20	10000

В соответствии с выявленными угрозами определим требуемые методы и средства защиты (см. таблицу 1.2).

Таблица 1.2 — Пример применения методов и средств защиты

Отношение доступа	**Тип угрозы**	**Методы и средства защиты**	**Сопротивля-емость механизма защиты**
менеджер – база данных	прерывание	—	0,00
руководитель – база данных	прерывание, перехват	шифрование	0,95

Продолжение таблицы 1.2

руководитель – файловый сервер	перехват, фальсификация	шифрование, электронная цифровая подпись	0,97

При доступе менеджера к базе данных вероятность возникновения угрозы и величина ущерба небольшие, поэтому применение средств для защиты информации может быть неэффективным с точки зрения затрат на разработку и простоты работы с системой.

Статическая модель системы представлена на рисунке 1.8.

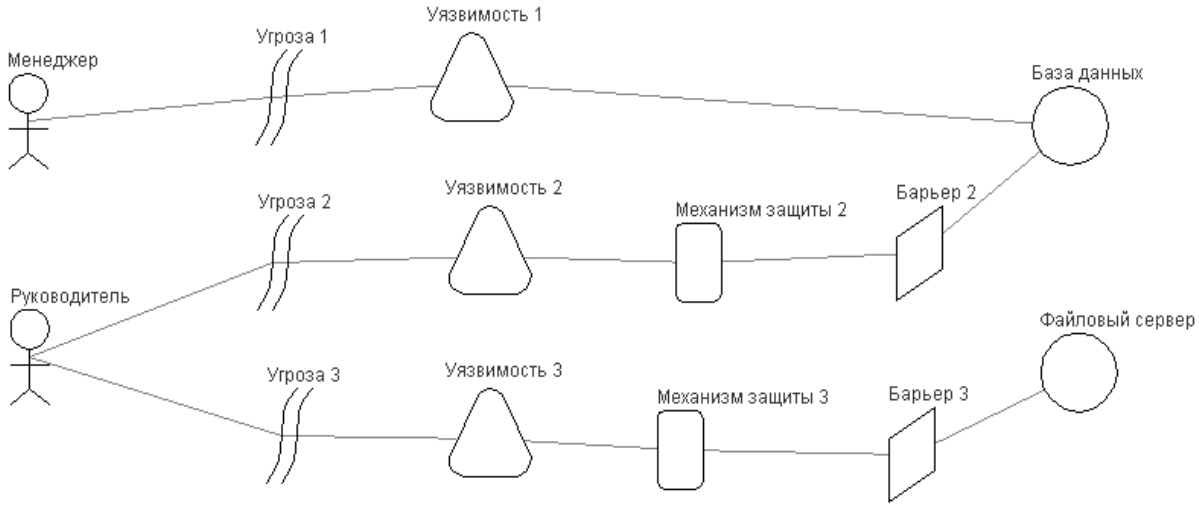


Рисунок 1.8 — Статическая модель информационной системы предприятия с учетом угроз и механизмов защиты

Остаточные риски барьеров представлены в таблице 1.3.

Таблица 1.3 — Остаточные риски барьеров

Отношение доступа	**Барьер**	**Риск**
Менеджер – База данных	—	1

Продолжение таблицы 1.3

Руководитель – База данных	Барьер 2	4,5
Руководитель – Файловый сервер	Барьер 3	60

Остаточный риск всей системы вычисляется как сумма всех остаточных рисков $Risk_{\Sigma} = 65{,}5$.

Суммарная защищенность равна $S_{\Sigma} = 0{,}015267$.

Наибольшее значение риска предполагается при доступе субъекта Руководитель к ресурсу Файловый сервер, данное значение значительно влияет на суммарный риск.

1.4 Выполнение работы

1. Запустить серверное приложение. Для этого требуется выполнить файл runServer.bat.

2. Для формирования множеств элементов системы открыть диалоговое окно «Элементы системы».

Для этого следует выполнить команду меню «Модель\Статическая».

3. В окне редактирования статической модели (см. рисунок 1.1) в соответствии с вариантом задания (см. таблицу 1.4) на вкладке «Ресурсы» добавить ресурсы информационной системы.

4. Перейти на вкладку «Субъекты» и ввести субъекты.

При редактировании субъектов (см. рисунок 1.10) задать доступные ресурсы в соответствии с вариантом задания.

5. Во вкладке «Угрозы» ввести угрозы нарушения доступа и вероятности их возникновения (см. рисунок 1.11) в соответствии с вариантом.

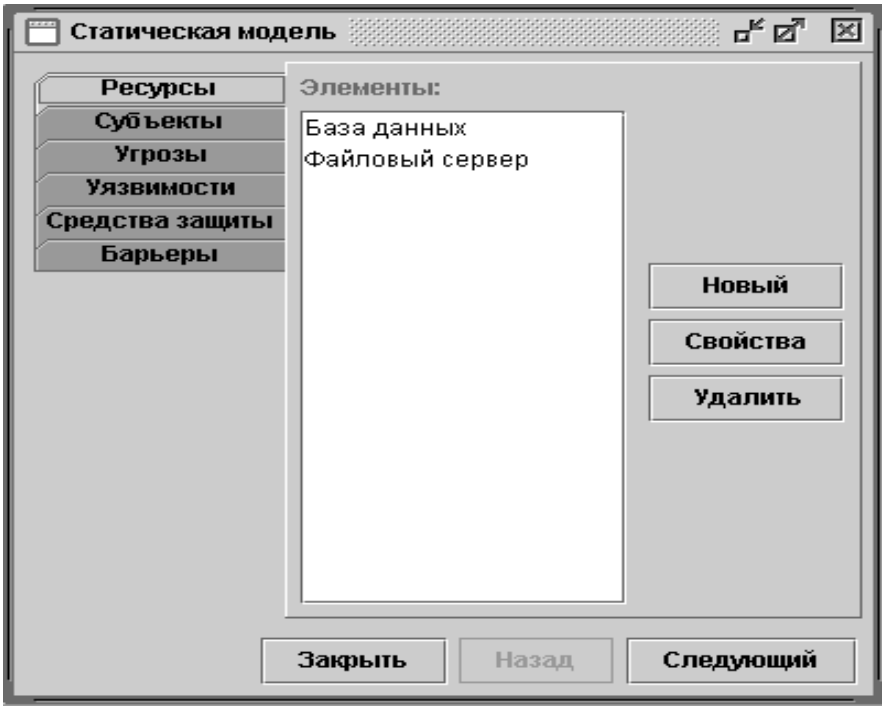


Рисунок 1.9 — Окно редактирования статической модели

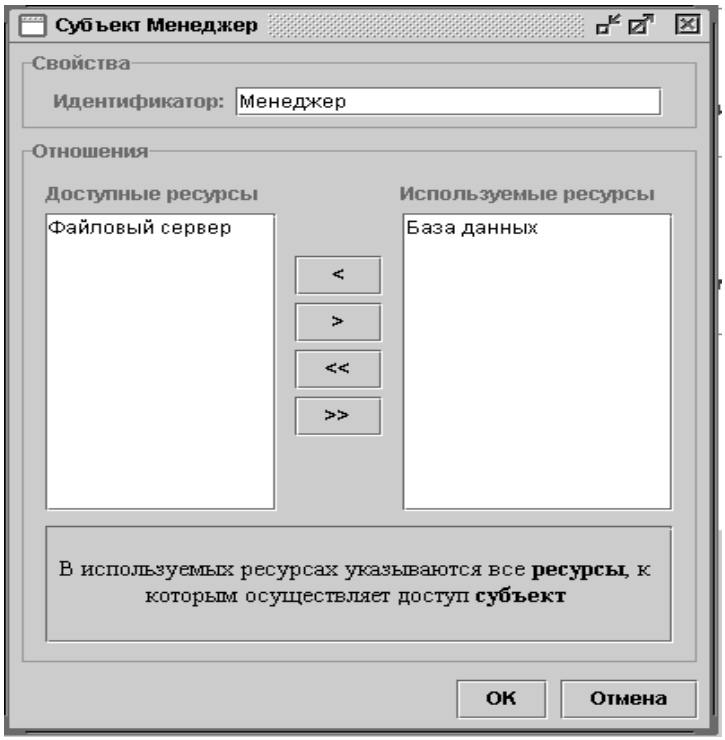


Рисунок 1.10 — Окно редактирования свойств субъекта

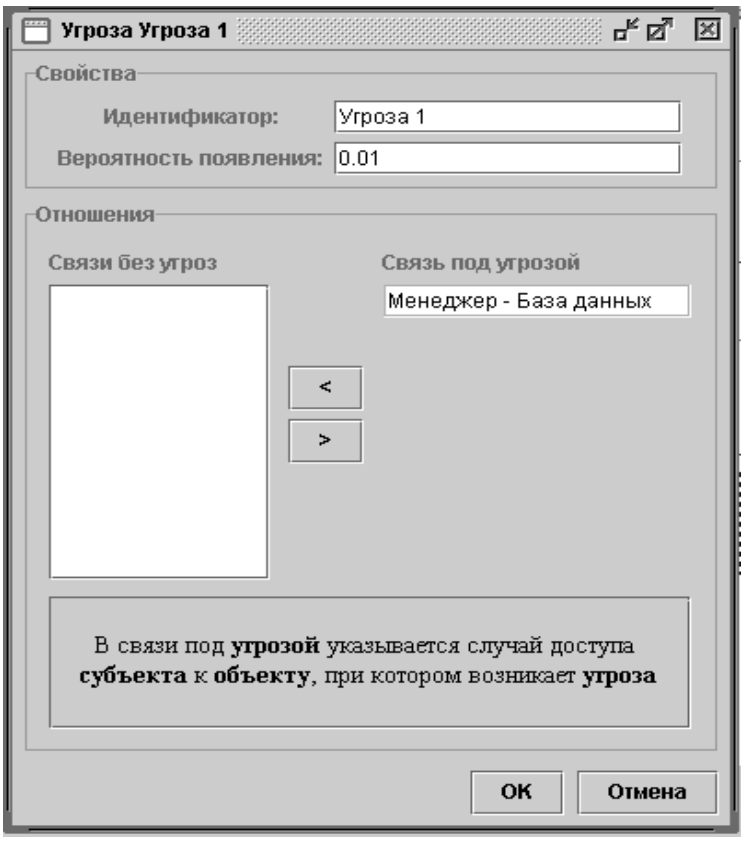


Рисунок 1.11 — Окно редактирования угрозы

6. Во вкладке «Уязвимости» ввести величины ущерба для всех уязвимостей.

Для всех отношений доступа выбрать методы защиты информации, которые будут препятствовать возникающим угрозам.

Добавить во вкладке «Средства защиты» механизмы защиты с выбранными самостоятельно степенями сопротивляемости (см. рисунок 1.12).

7. Открыть графическое представление статической модели двойным щелчком мыши на строке «Диаграмма статической модели» (см. рисунок 1.13) в древовидном списке. По диаграмме проанализируйте правильность задания элементов и связей.

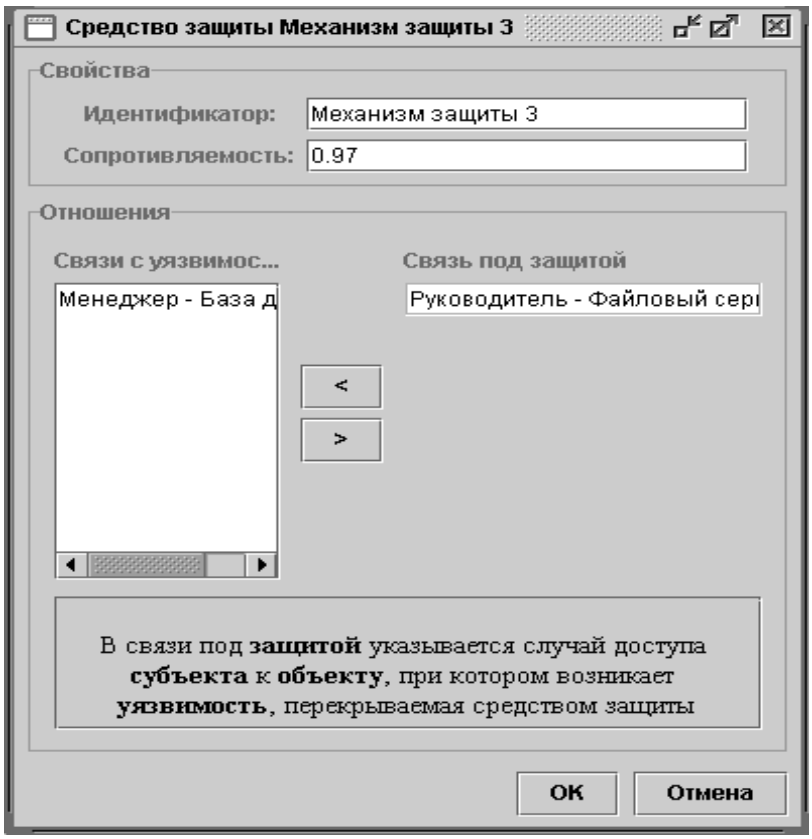


Рисунок 1.12 — Окно редактирования свойств механизма защиты

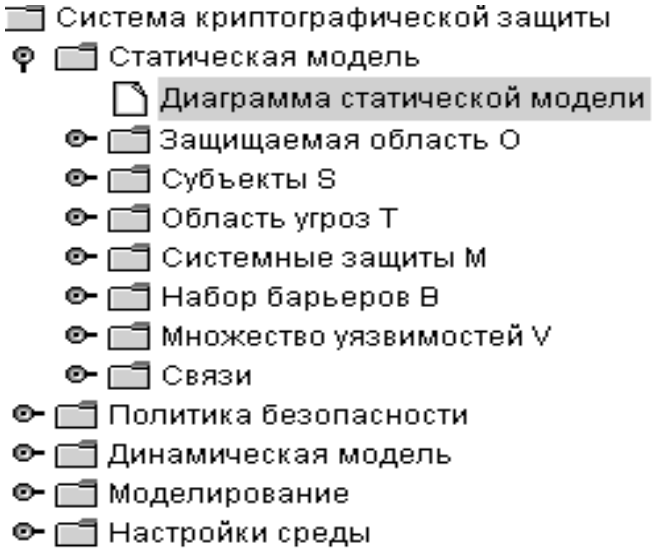


Рисунок 1.13 — Вид древовидного списка системы изучения методов и средств защиты информации

1. Для просмотра рисков откройте окно анализа защищенности (пункт меню «Анализ\Защищенность системы»). Вид данного окна представлен на рисунке 1.14

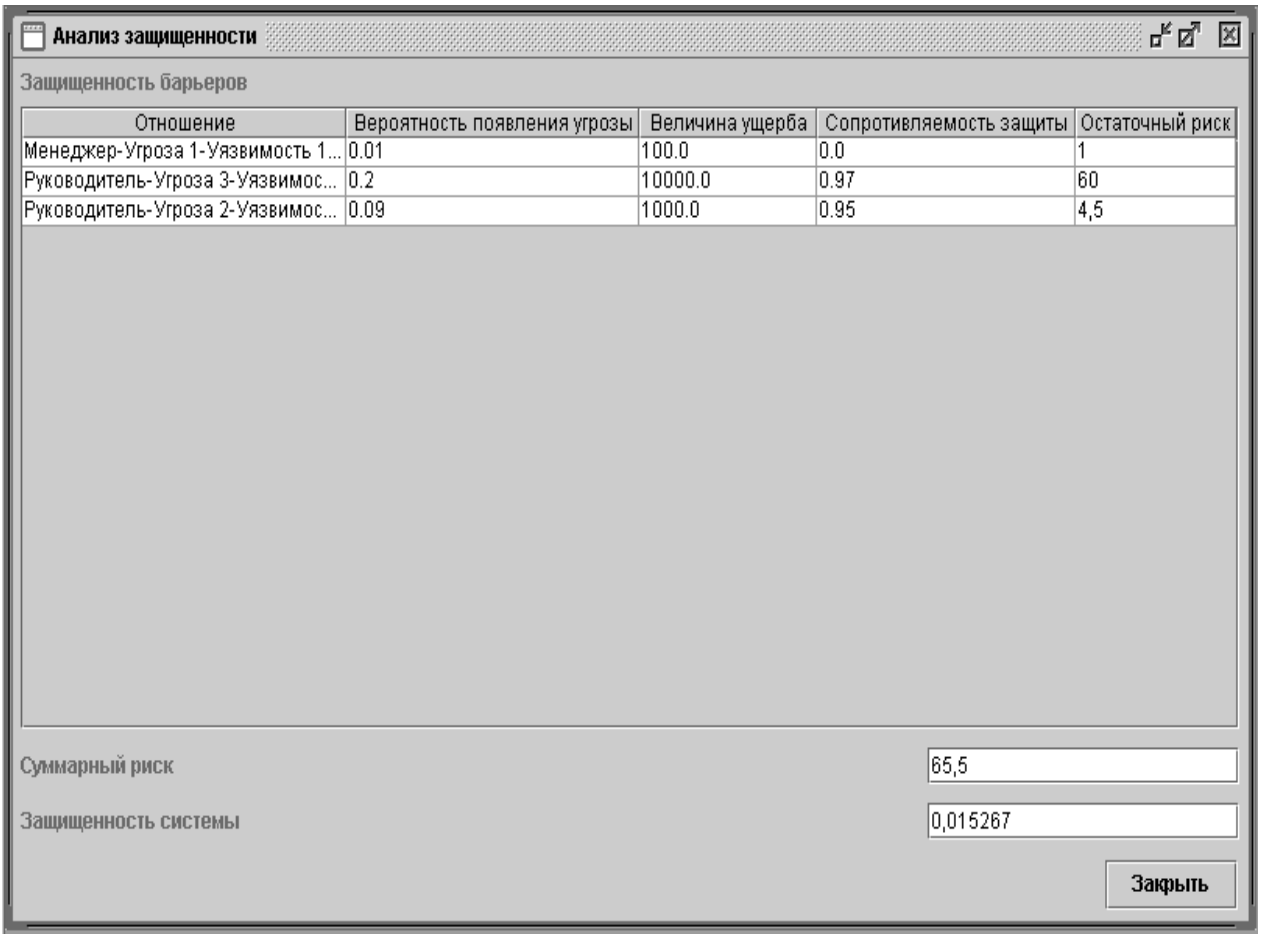


Рисунок 1.14 — Окно анализа защищенности

9. Проанализируйте и определите наиболее ценные ресурсы с точки зрения возможного ущерба.

10.Сделайте выводы о том, каким образом влияют вероятности и характеристики угроз, уязвимостей и механизмов защиты на общую защищенность системы, изменяя параметры в окне анализа защищенности.

Таблица 1.4 — Варианты заданий

Вариант	***Отношение доступа***	***Угроза***	***Параметры***	
			Вероятность возникновения угрозы	***Величина ущерба***
1	*Менеджер-База данных*	*перехват*	*0,1*	*100*
	Руководитель-Почтовый сервер	*фальсификация*	*0,3*	*50*
	Руководитель-База данных	*модификация*	*0,2*	*1000*
2	*Менеджер-База данных*	*перехват*	*0,4*	*200*
	Руководитель-Почтовый сервер	*перехват, фальсификация*	*0,3*	*520*
	Руководитель-База данных	*модификация, фальсификация*	*0,7*	*130*
3	*Менеджер-База данных*	*модификация, перехват*	*0,1*	*10*
	Менеджер–Почтовый сервер	*перехват, модификация*	*0,15*	*200*
	Руководитель-База данных	*модификация*	*0,2*	*620*
	Руководитель-Почтовый сервер	*фальсификация*	*0,2*	*540*
4	*Менеджер–Почтовый сервер*	*перехват, модификация*	*0,15*	*200*
	Менеджер-База данных	*перехват*	*0,1*	*120*
	Руководитель-Почтовый сервер	*фальсификация*	*0,3*	*2000*
	Руководитель-База данных	*модификация*	*0,2*	*10*

1.5 Содержание отчета

Отчет выполняется один на бригаду и должен включать:

1. Наименование и цель работы.
2. Краткие теоретические сведения.
3. Статическую модель согласно варианта.
4. Окна редактирования ресурсов, субъектов, угроз, уязвимостей, средств защиты.
5. Анализ защищенности
6. Выводы.

1.6 Контрольные вопросы к главе 1

1. Какие элементы описывает математическая модель информационной системы?
2. Какие существуют угрозы нарушения безопасности и их характеристика?
3. Меры обеспечения безопасности системы.
4. Подходы к оценке защищенности системы.
5. Из каких множеств состоит статическая модель системы?

2 ЛАБОРАТОРНАЯ РАБОТА № 2

СОВРЕМЕННЫЕ АЛГОРИТМЫ СИММЕТРИЧНОГО ШИФРОВАНИЯ

2.1 Теоретические сведения

Исторически первыми появились симметричные алгоритмы шифрования. Основой секретности этих алгоритмов является секретность симметричного ключа, который должен быть известен отправителю сообщения и получателю. Схема шифрования с использованием симметричного ключа изображена на рисунке 2.1.

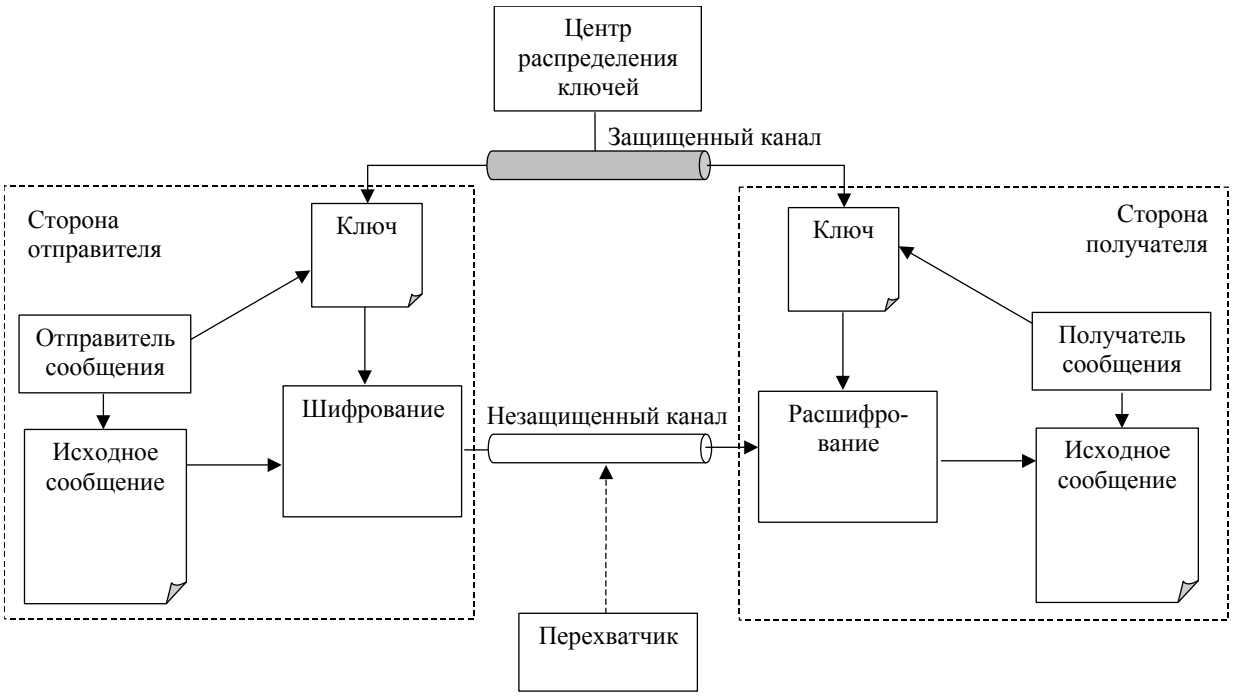


Рисунок 2.1 — Схема симметричного шифрования

Среди современных алгоритмов шифрования с симметричным ключом известны и широко используются Data Encryption Standard, IDEA, ГОСТ 28147-89, Blowfish, RC5 и др.

Симметричные алгоритмы шифрования основаны на применении двух способов преобразования бит данных:

- диффузия;
- конфузия.

Диффузия выполняет роль рассеивания статистических особенностей открытого текста по широкому диапазону статистических характеристик шифрованного текста.

Это достигается тем, что значение каждого элемента открытого текста влияет на значения многих элементов шифрованного текста или, что оказывается эквивалентным сказанному, любой из элементов шифрованного текста зависит от множества элементов открытого текста.

В результате применения диффузии частотные характеристики использования отдельных символов и последовательностей символов должны становиться близкими к равномерным.

Конфузия представляет собой механизм сложных подстановок, которые затрудняют установление статистической взаимосвязи между шифрованным текстом и ключом.

Целью применения конфузии является противостояние определению ключа при известных статистических характеристиках закрытого текста.

Достоинством симметричных алгоритмов является высокое быстродействие и малая длина ключа по сравнению с ключами у алгоритмов с открытым ключом.

2.1.1 Алгоритм шифрования DES

Алгоритм DES использует комбинацию подстановок и перестановок. DES осуществляет шифрование 64-битовых блоков данных с помощью 64-битового ключа, в котором значащими являются 56 бит (остальные 8 бит — проверочные биты для контроля на четность).

Дешифрование в DES является операцией, обратной шифрованию, и выполняется путем выполнения операции шифрования в обратной последовательности.

Процесс шифрования заключается в начальной перестановке битов 64-битового блока, 16 циклов шифрования и, наконец, в конечной перестановке битов.

Следует отметить, что все приводимые таблицы являются стандартными и должны включаться в реализацию алгоритма DES в неизменном виде.

Все перестановки и коды в таблицах подобраны разработчиками таким образом, чтобы максимально затруднить процесс расшифровки путем подбора ключа.

При описании алгоритма применены следующие обозначения:

L и *R* — последовательности битов. (левая и правая).

LR — конкатенация последовательностей *L* и *R*, то есть такая последовательность битов, длина которой равна сумме длин *L* и *R*; в последовательности *LR* биты последовательности *R* следуют за битами последовательности *L*.

Пусть из файла исходного текста считан 64-битовый блок *T*. Этот блок преобразуется с помощью матрицы начальной перестановки *IP*.

Таблица 2.1 — Матрица начальной перестановки IP

58	50	42	34	26	18	10	2
60	52	44	36	28	20	12	4
62	54	46	38	30	22	14	6
64	56	48	40	32	24	16	8
57	49	41	33	25	17	9	1
59	51	43	35	27	19	11	3
61	53	45	37	29	21	13	5
63	55	47	39	31	23	15	7

Биты входного блока *T* (64 бита) переставляются в соответствии с матрицей *IP*: бит 58 входного блока *T* становится битом 1, бит 50 — битом 2 и т. д..

Эту перестановку можно описать выражением $T_0=IP(T)$. Полученная последовательность битов T_0 разделяется на 2 последовательности: L_0 — левые или старшие биты, R_0 — правые или младшие биты, каждые из которых содержат 32 бита.

Затем выполняется итеративный процесс шифрования, состоящий из 16 шагов (циклов). Пусть T_i — результат i-той операции:

$T_i = L_iR_i$,

где $L_i=t_1t_2\ldots t_{32}$ (первые 32 бита),

$R_i=t_{33}t_{34}\ldots t_{64}$ (последние 32 бита).

Тогда результат i-той операции описывается следующими формулами:

$$L_i = R_{i-1};$$

$$R_i = L_{i-1} \oplus f(R_{i-1}, K_i).$$

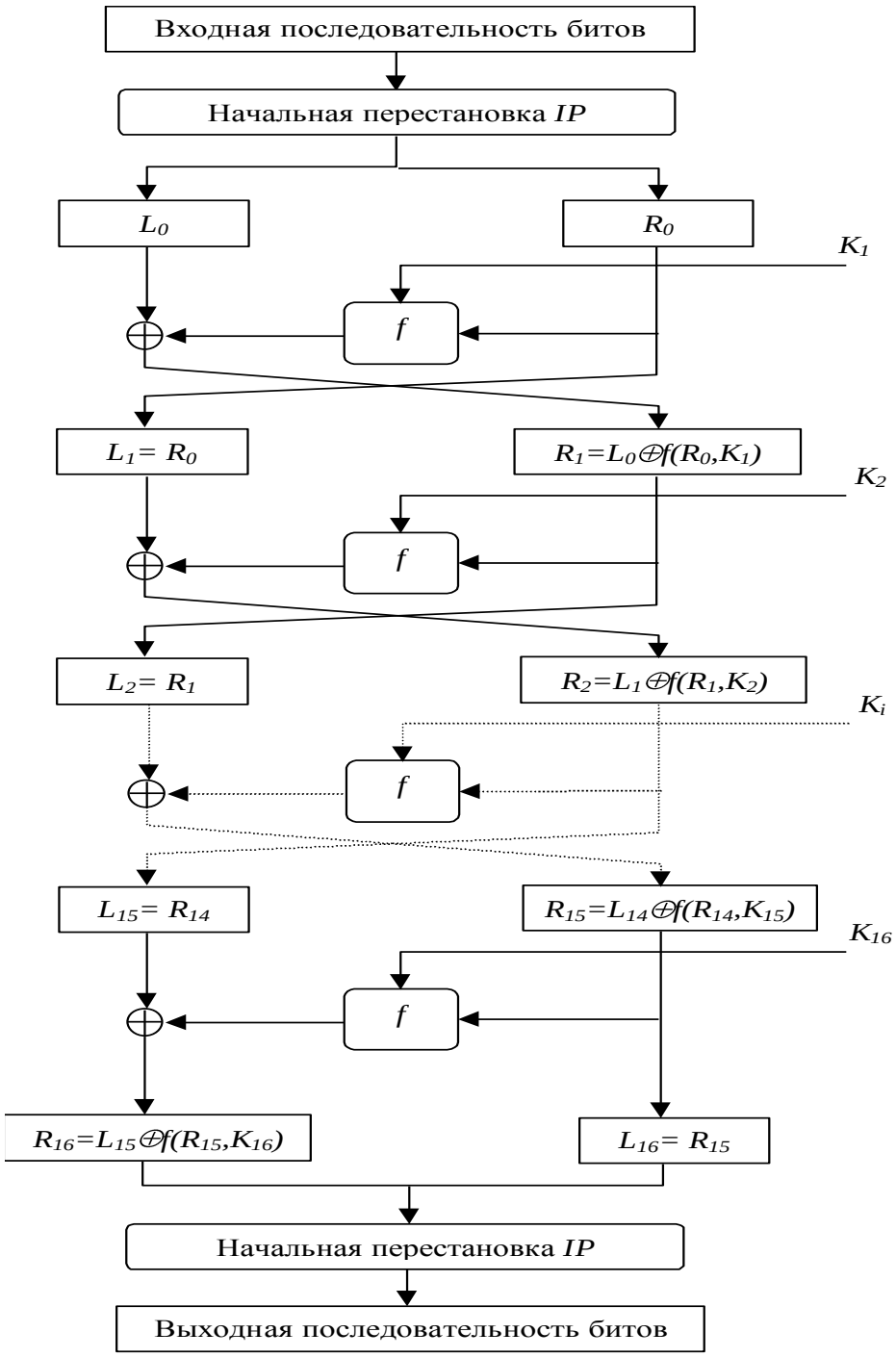


Рисунок 2.2 — Структура алгоритма DES

Функция *f* называется функцией шифрования.

Её аргументами является последовательность R_{i-1}, получаемые на предыдущем шаге итерации и 48-битовый ключ K_i, который является результатом преобразования 64-битового ключа шифра *K*.

На последнем шаге итерации получают последовательности R_{16} и L_{16} (без перестановки местами), которые конкатенируются в 64-битовую последовательность $R_{16}L_{16}$.

По окончанию шифрования осуществляется восстановление позиции битов с помощью матрицы обратной перестановки IP^{-1}.

Таблица 2.1 — Матрица обратной перестановки IP^{-1}

40	8	48	16	56	24	64	32
39	7	47	15	55	23	63	31
38	6	46	14	54	22	62	30
37	5	45	13	53	21	61	29
36	4	44	12	52	20	60	28
35	3	43	11	51	19	59	27
34	2	42	10	50	18	58	26
33	1	41	9	49	17	57	25

Процесс расшифрования данных является инверсным по отношению к процессу шифрования. Все действия должны быть выполнены в обратном порядке.

Теперь рассмотрим, что скрывается под преобразованием, обозначенным буквой *f* .

Схема вычисления функции шифрования $f(R_{i-1},K_i)$ показана ниже.

Для вычисления функции *f* используются:

- функция *E* (расширение 32 бит до 48);
- функции S_1, S_2, ... S_8 (преобразование 6-битового числа в 4-битовое);
- функция *P* (перестановка битов в 32-битовой последовательности).

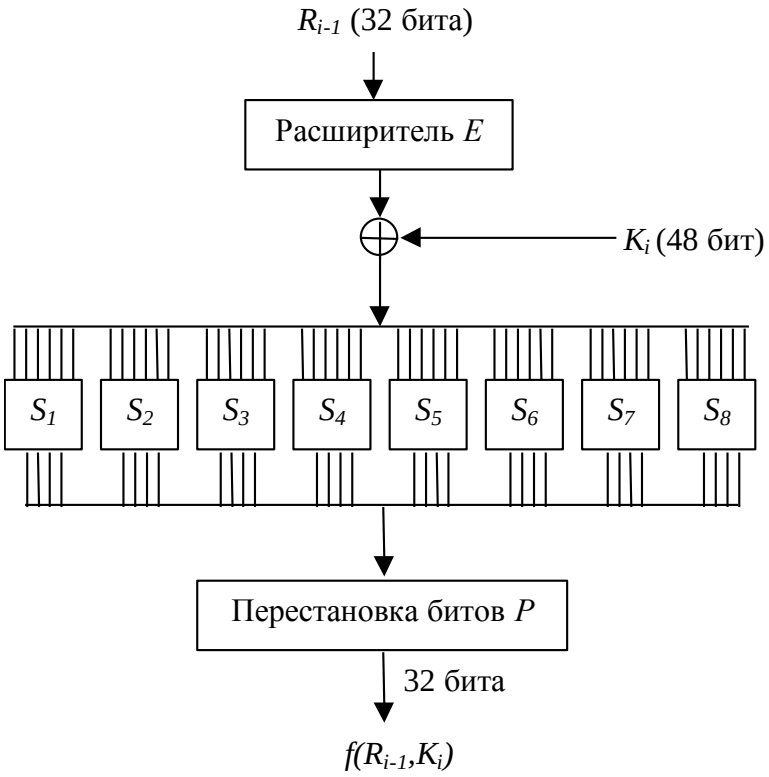


Рисунок 2.3 — Схема вычисления функции шифрования

Таблица 2.2 — Функция расширения E

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

Таблица 2.3 — Функция P перестановки битов

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

Таблица 2.4 — Функции преобразования S_1, S_2, … S_8

		Номер столбца															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Номер строки	0	14	4	13	1	2	15	11	8	3	10	6	12	5	9	0	7
	1	0	15	7	4	14	2	13	1	10	6	12	11	9	5	3	8
	2	4	1	4	8	13	6	2	11	15	12	9	7	3	10	5	0
	3	15	12	8	2	4	9	1	7	5	11	3	14	10	0	6	13
	0	15	1	8	14	6	11	3	4	9	7	2	13	12	0	5	10
	1	3	13	4	7	15	2	8	14	12	0	1	10	6	9	11	5
	2	0	14	7	11	10	4	13	1	5	8	12	6	9	3	2	15
	3	13	8	10	1	3	15	4	2	11	6	7	12	0	5	14	9
	0	10	0	9	14	6	3	15	5	1	13	12	7	11	4	2	8
	1	13	7	0	9	3	4	6	10	2	8	5	14	12	11	15	1
	2	13	6	4	9	8	15	3	0	11	1	2	12	5	10	14	7
	3	1	10	13	0	6	9	8	7	4	15	14	3	11	5	2	12
	0	7	13	14	3	0	6	9	10	1	2	8	5	11	12	4	15
	1	13	8	11	5	6	15	0	3	4	7	2	12	1	10	14	9
	2	10	6	9	0	12	11	7	13	15	1	3	14	5	2	8	4
	3	3	15	0	6	10	1	13	8	9	4	5	11	12	7	2	14
	0	2	12	4	1	7	10	11	6	8	5	3	15	13	0	14	9
	1	14	11	2	12	4	7	13	1	5	0	15	10	3	9	8	6
	2	4	2	1	11	10	13	7	8	15	9	12	5	6	3	0	14
	3	11	8	12	7	1	14	2	13	6	15	0	9	10	4	5	3

Продолжение таблицы 2.5

0	12	1	10	15	9	2	6	8	0	13	3	4	14	7	5	11
1	10	15	4	2	7	12	9	5	6	1	13	14	0	11	3	8
2	9	14	15	5	2	8	12	3	7	0	4	10	1	13	1	6
3	4	3	2	12	9	5	15	10	11	14	1	7	6	0	8	13
0	4	11	2	14	15	0	8	13	3	12	9	7	5	10	6	1
1	13	0	11	7	4	9	1	10	14	3	5	12	2	15	8	6
2	1	4	11	13	12	3	7	14	10	15	6	8	0	5	9	2
3	6	11	13	8	1	4	10	7	9	5	0	15	14	2	3	12
0	13	2	8	4	6	15	11	1	10	9	3	14	5	0	12	7
1	1	15	13	8	10	3	7	4	12	5	6	11	0	14	9	2
2	7	11	4	1	9	12	14	2	0	6	10	13	15	3	5	8
3	2	1	14	7	4	10	8	13	15	12	9	0	3	5	6	11

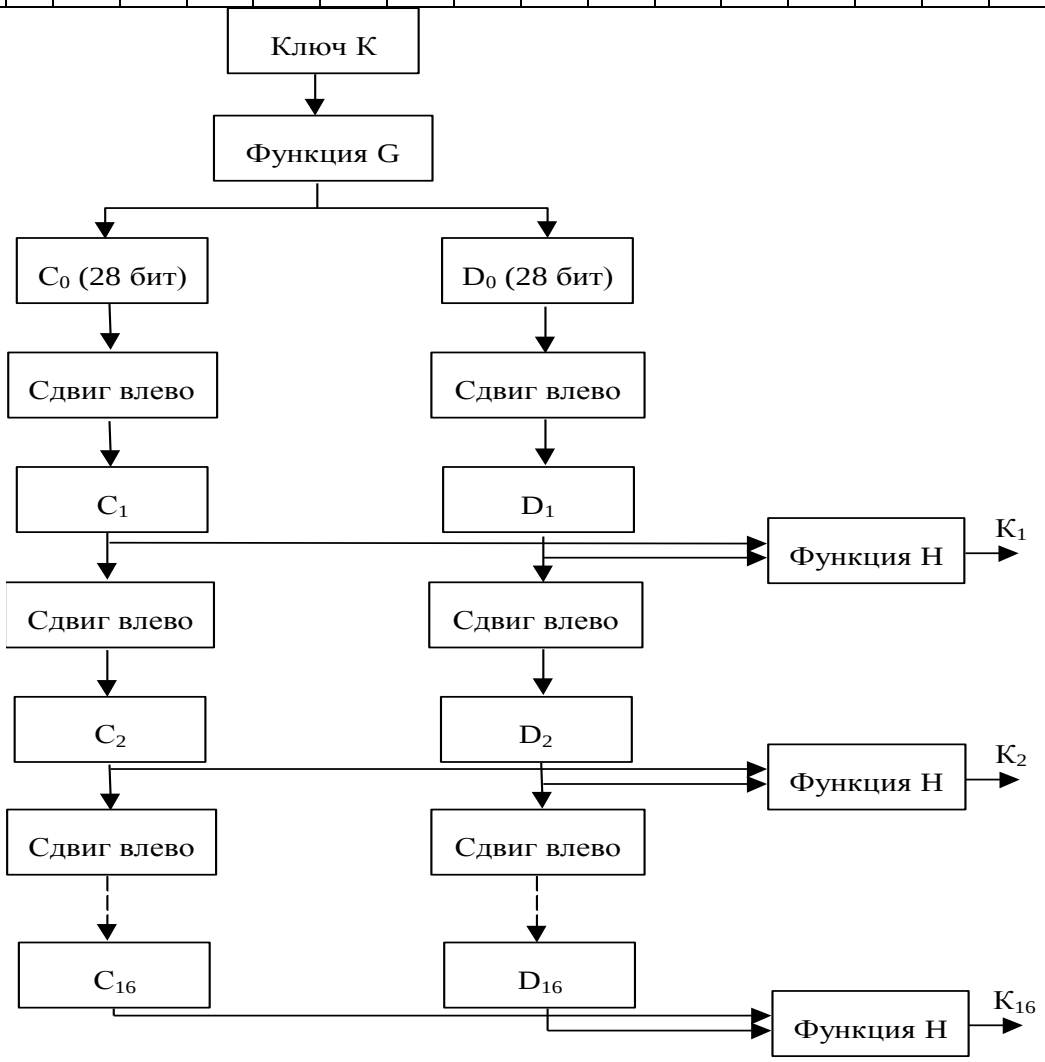


Рисунок 1.4 — Схема алгоритма вычисления ключей K_i

Таблица 2.5 — Функция G первоначальной подготовки ключа

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

Данная таблица разделена на две части.

Результат преобразования *G(K)* разбивается на две половины C_0 и D_0 по 28 бит каждая.

Первые четыре строки матрицы *G* определяют, как выбираются биты последовательности C_0 .

Следующие четыре строки матрицы *G* определяют, как выбираются биты последовательности.

Как видно из таблицы, для генерации последовательности C_0 и D_0 не используются биты 8, 16, 24, 32,40, 48, 56 и 64 ключа шифра.

Эти биты не влияют на шифрование и могут служить для других целей (например, для контроля по четности).

Таким образом, в действительности ключ шифра является 56-битовым.

После определения C_0 и D_0 рекурсивно определяются C_i и D_i , $i = 1,2…16$.

Для этого применяются операции циклического сдвига влево на один или два бита в зависимости от номера шага итерации.

Операции сдвига выполняются для последовательностей C_i и D_i независимо.

Таблица 2.6 — Таблица количества сдвигов

Номер итерации	**Количество Si сдвигов влево, бит**	**Номер итерации**	**Количество Si сдвигов влево, бит**
1	1	9	1
2	1	10	2
3	2	11	2
4	2	12	2
5	2	13	2
6	2	14	2
7	2	15	2
8	2	16	1

Ключ , определяемый на каждом шаге итерации, есть результат выбора конкретных битов из 56-битовой последовательности C_iD_i и их перестановки.

Другими словами, ключ $K_i = H(C_iD_i)$, где функция H определяется матрицей, завершающей обработку ключа.

Таблица 2.7 — Функция H завершающей обработки ключа

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

2.1.2 Режимы работы блочных шифров

Чтобы воспользоваться блочным алгоритмом для решения разнообразных криптографических задач, разработаны четыре рабочих режима:

- электронная кодовая книга ECB (Electronic Code Book);
- сцепление блоков шифра CBC (Cipher Block Chaining);
- обратная связь по шифротексту CFB (Cipher Feed Back);
- обратная связь по выходу OFB (Output Feed Back).

2.1.2.1 Режим «Электронная кодовая книга»

Длинный файл разбивается на N-битовые отрезки (блоки) по n байтов.

Каждый из этих блоков шифруют независимо с использованием одного и того же ключа шифрования (см. рисунок 2.5).

Основное достоинство — простота реализации.

Недостаток — относительно слабая устойчивость против квалифицированных криптоаналиков.

Из-за фиксированного характера шифрования при ограниченной длине блока N битов возможно проведение криптоанализа «со словарем». Блок такого размера может повторится в сообщении вследствие большой избыточности в тексте на естественном языке.

Это приводит к тому, что идентичные блоки открытого текста в сообщении будут представлены идентичными блоками шифротекста.

Что дает криптоаналитику некоторую информацию о содержании сообщения.

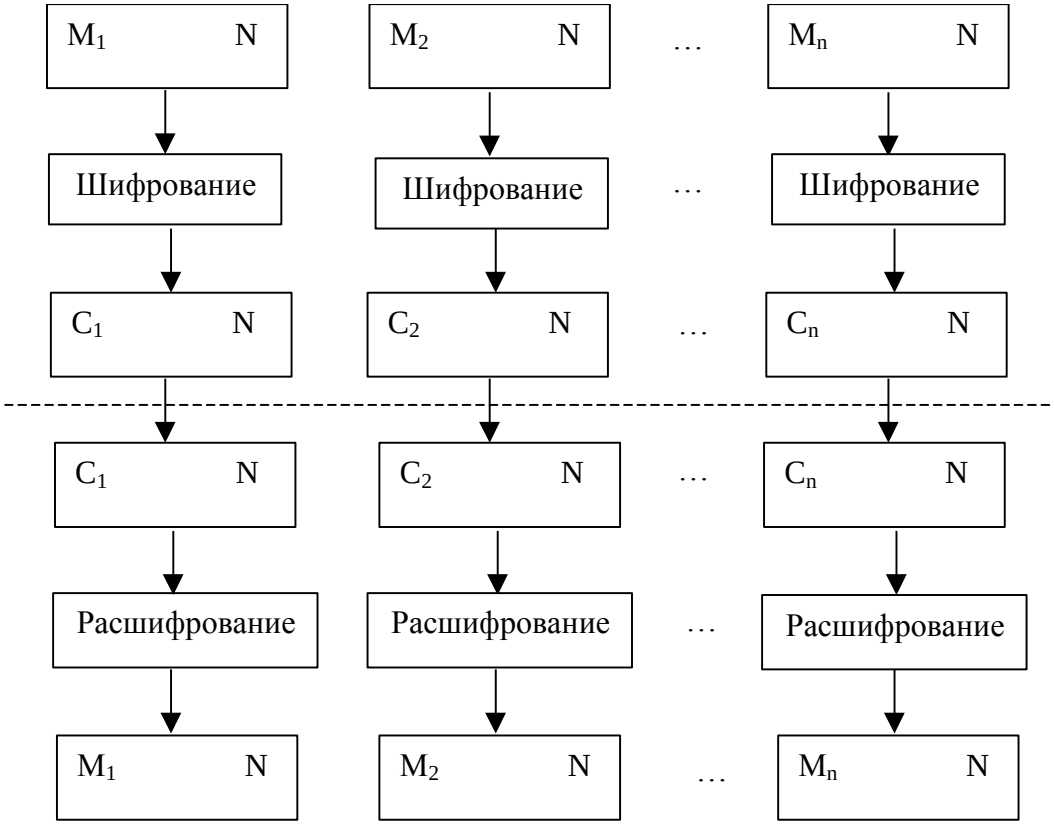


Рисунок 2.5 — Схема блочного алгоритма в режиме электронной кодовой книги

2.1.2.2 Режим «Сцепление блоков шифра»

В этом режиме исходный файл M разбивается на N-битовые блоки: $M=M_1M_2…M_n$.

Первый блок M_1 складывается по модулю 2 с N-битовым начальным вектором IV, который меняется ежедневно и держится в секрете.

Полученная сумма затем шифруется с использование ключа блочного алгоритма, известного и отправителю, и получателю информации. Полученный N-битовый шифр C_1 складывается по модулю 2 со вторым блоком текста, результат шифруется и получается второй N-битовый шифр C_2, и т.д.

Процедура повторяется до тех пор, пока не будут обработаны все блоки текста.

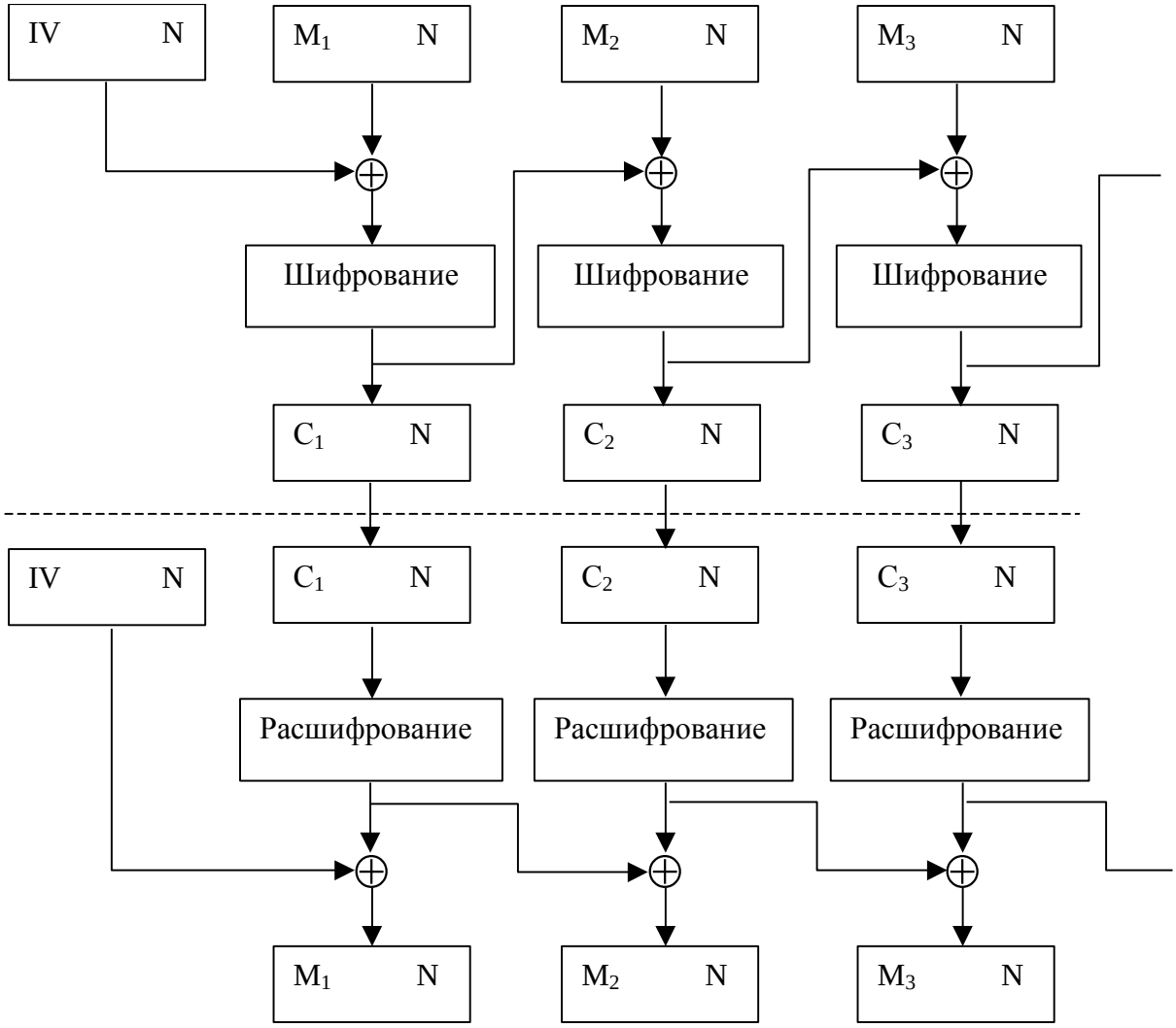


Рисунок 2.6 — Схема блочного алгоритма в режиме счепления блоков шифра

Таким образом, для всех i = 1…n (n — число блоков) результат шифрования C_i определяется следующим образом: $C_i = E (M_i \oplus C_{i-1})$, где $C_0 = IV$ — начальное значение шифра, равное начальному вектору (вектору инициализации).

Очевидно, что последний N-битовый блок шифротекста является функцией секретного ключа, начального вектора и каждого бита открытого текста независимо от его длины.

Этот блок шифротекста называют кодом аутентификации сообщения (КАС).

Код КАС может быть легко проверен получателем, владеющим секретным ключом и начальным вектором, путем повторения процедуры, выполненной отправителем. Посторонний, однако, не может осуществить

генерацию КАС, который воспринялся бы получателем как подлинный, чтобы добавить его к ложному сообщению, либо отделить КАС от истинного сообщения для использования его с измененным или ложным сообщением.

Достоинство данного режима в том, что он не позволяет накапливаться ошибкам при передаче.

Блок M_i является функцией только C_{i-1} и C_i. Поэтому ошибка при передаче приведет к потере только двух блоков исходного текста.

2.1.2.3 Режим «Обратная связь по шифру»

Файл, подлежащий шифрованию (расшифрованию), считывается последовательными блоками длиной k-битов (k=1…N).

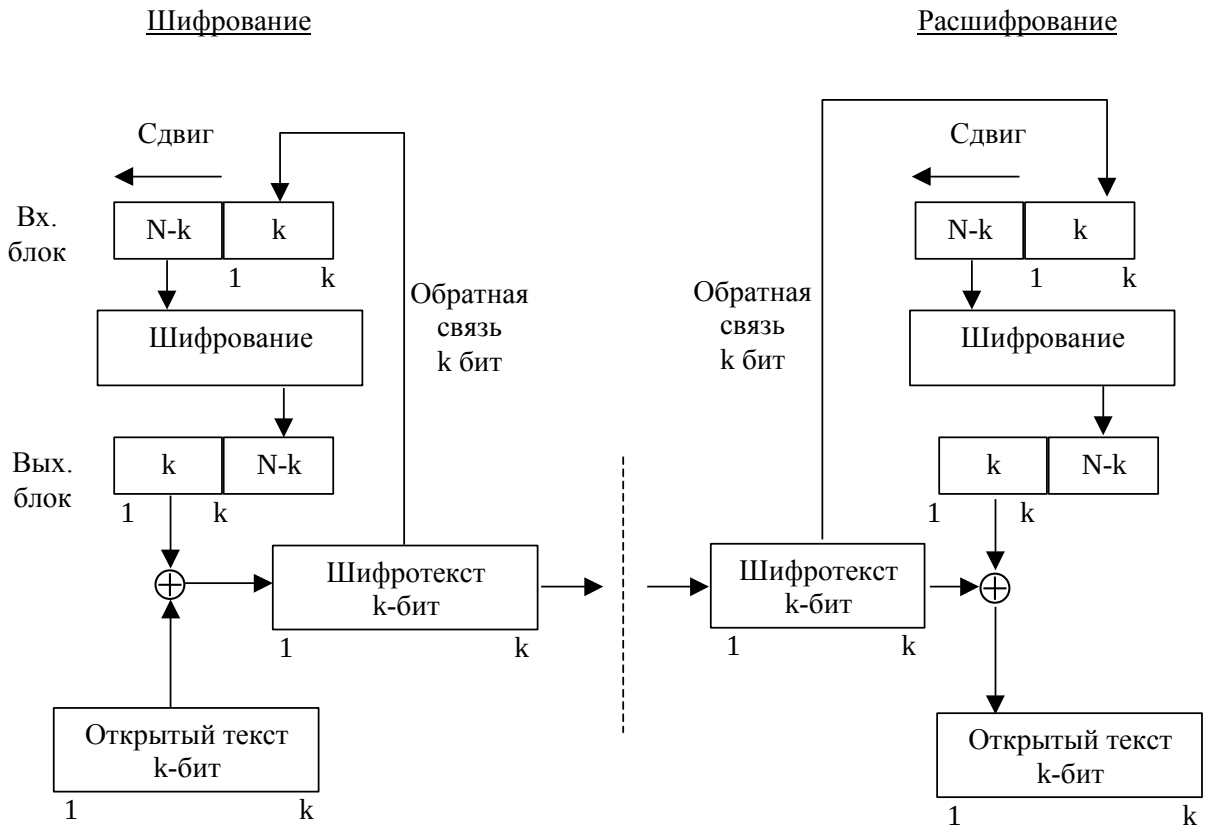


Рисунок 2.7 — Схема блочного алгоритма в режиме обратной связи по шифру

Входной блок (N-битовый регистр сдвига) вначале содержит вектор инициализации, выровненный по правому краю.

Предположим, что в результате разбиения на блоки мы получили n блоков длиной k битов каждый (остаток дописывается нулями или пробелами). Тогда для любого i=1…n блок шифротекста

$$C_i = M_i \oplus P_{i-1},$$

где P_{i-1} обозначает k старших битов предыдущего зашифрованного блока.

Обновление сдвигового регистра осуществляется путем удаления его старших k битов и записи C_i в регистр.

Восстановление зашифрованных данных также выполняется относительно просто: P_{i-1} и C_i вычисляются аналогичным образом и

$$M_i = C_i \oplus P_{i-1}.$$

2.1.2.4 Режим «Обратная связь по выходу»

Этот режим тоже использует переменный размер блока и сдвиговый регистр, инициализируемый так же, как в режиме CFB, а именно — входной блок вначале содержит вектор инициализации IV, выровненный по правому краю.

При этом для каждого сеанса шифрования данных необходимо использовать новое начальное состояние регистра, которое должно пересылаться по каналу открытым текстом.

Положим

$$M = M_1 M_2 \ldots M_n.$$

Для всех i = 1…n

$$C_i = M_i \oplus P_i,$$

где P_i обозначает k старших битов операции блочных алгоритмов.

Отличия от режима обратной связи по шифротексту состоит в методе обновления сдвигового регистра.

Это осуществляется путем отбрасывания старших k битов и дописывания справа P_i.

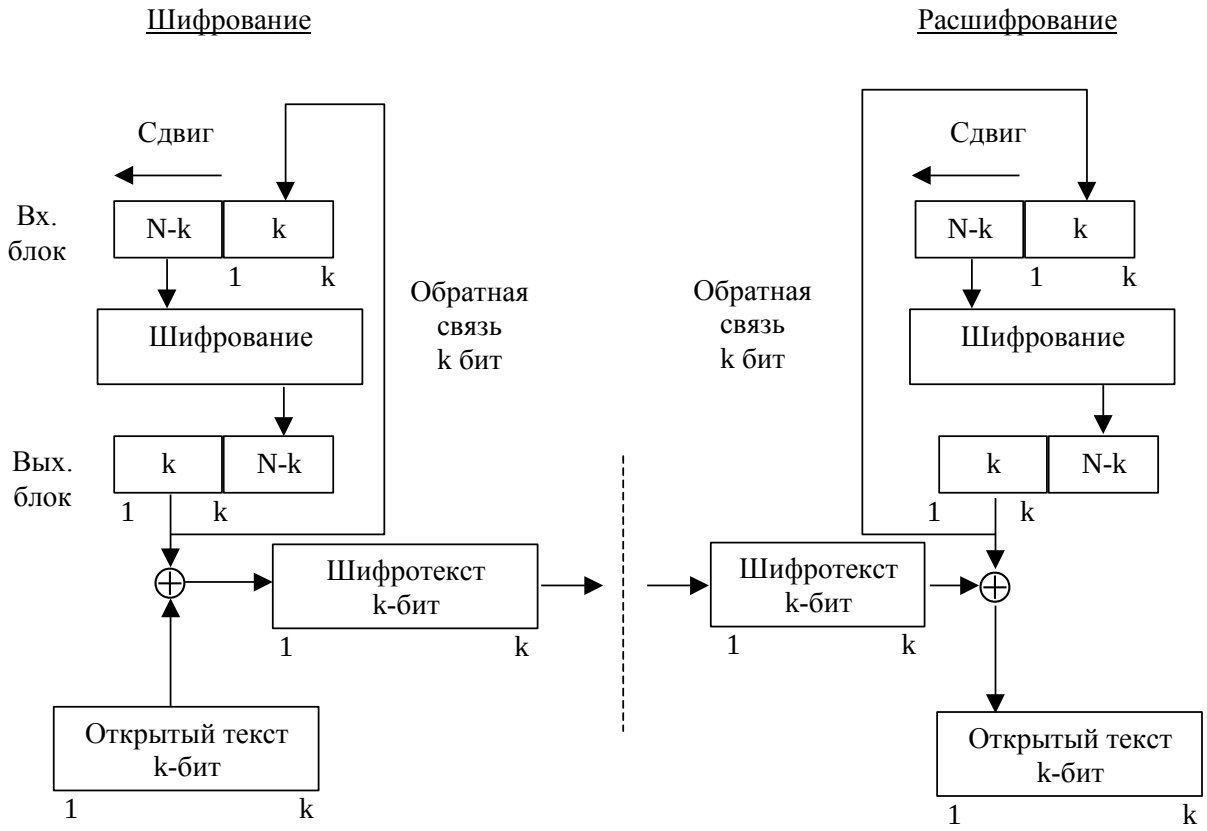


Рисунок 2.8 — Схема блочного алгоритма в режиме обратной связи по выходу

2.2 Применение компьютерной системы для изучения симметричных алгоритмов шифрования

В качестве примера рассмотрим задачу конфиденциальной передачи данных от одного процесса источнику сообщения другому процессу получателю сообщения.

В компьютерной системе изучения методов и средств аппаратно-программной защиты информации разработана динамическая модель. Графическое представление модели взаимодействия процессов изображено на рисунке 2.9.

Рисунок 2.9 — Диаграмма взаимодействия процессов при конфиденциальной передаче данных

Процесс источник сообщения S выполняет шифрование данных с помощью ключа K, после чего передает шифротекст по каналу процессу получателю R.

Процесс R обладает ключом K, с помощью которого выполняется дешифрование шифротекста и получения исходного сообщения.

В компьютерной системе реализованы следующие блоки симметричного шифрования:

- DES;
- IDEA.

Для описание алгоритмов работы процессом применяется язык Java.

Алгоритмы IDEA и DES реализованы в виде классов, предоставляющих следующие методы:

- конструктор — выполняет инициализацию объекта;
- void setIV(byte IV[]) — метод устанавливающий начальные вектор для режимов сцепления шифровальных блоков, обратной связи по шифротексту и по выходу;
- void resetFB() — выполняет сброс регистра сдвига для режимов обратной связи по шифротексту и по выходу в начальное состояние;
- boolean setKey(byte[] key) — метод для установки ключа;
- byte[] encode(byte[] key, byte[] data) — метод, выполняющий шифрование одного блока данных data с помощью ключа key;
- byte[] decode(byte[] key, byte[] data) — метод, выполняющий дешифрование одного блока данных data с помощью ключа key;
- byte[] encodeBlock(byte[] data) — метод выполняет шифрование одного блока данных data с помощью ключа, установленного функцией setKey;

- byte[] decodeBlock(byte[] data) — метод выполняет дешифрование одного блока данных data с помощью ключа, установленного функцией setKey;
- byte[] encodeData(byte data[], int mode) — выполняет шифрование данных data произвольной длины одним из режимов работы (см. таблицу 2.9);
- byte[] decodeData(byte data[], int mode) — выполняет дешифрование данных data произвольной длины одним из режимов работы (см. таблицу 2.9).

Таблица 2.9 — Константы режимов работы блочных шифров

Режим работы блочного шифра	**Константа**	**Примечание**
Электронной шифровальной книги	MODE_ECB	
Сцепления шифровальных блоков	MODE_CBC	
Шифрованной обратной связи	MODE_CFB	Длина данных в методах encodeData и decodeData должна быть не больше длины стандартного блока данных
Обратной связи по выходу	MODE_OFB	Длина данных в методах encodeData и decodeData должна быть не больше длины стандартного блока данных

В примере в качестве алгоритма шифрования используется алгоритм IDEA и режим сцепления шифрованных блоков (CBC).

Для шифрования определим метод процесса S do_send, блок-схема метода представлена на рисунке 2.10.

Рисунок 2.10 — Блок-схема работы метода do_send процесса S

При возникновении сообщения в узле процесса R сервер будет вызван метод onRecieve процесса R.
Блок-схема работы метода onRecieve процесса R представлена на рисунке 2.11.

Для сохранения значений переменных в процессе моделирования существуют методы:

- void logMessage(String message) — выполняет сохранение сообщения message в списке сообщений моделирования;
- void logDataMessage(String message, data) — выполняет сохранение сообщения message и связанных с сообщением данных data в журнале сообщений моделирования.

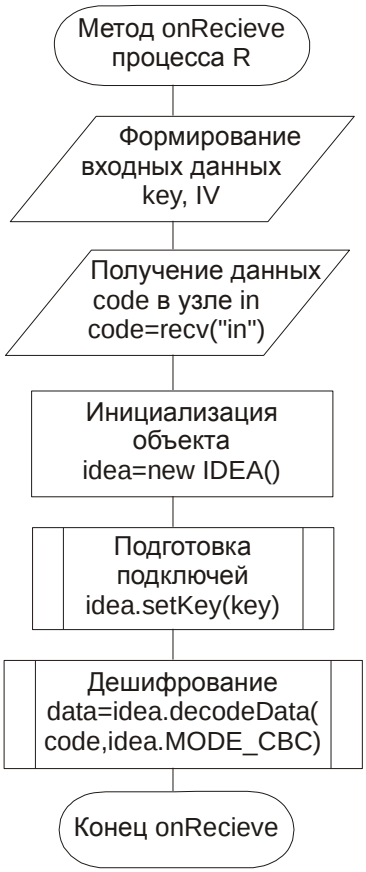


Рисунок 2.11 — Блок схема работы метода onRecieve процесса R

Для просмотра этих сообщений следует открыть окно сообщений моделирования двойным щелчком на элементе «Моделирование\Сообщения» в древовидном списке приложения.

Для работы с бинарными данными используется тип byte[] (массив байтовых целых чисел). Каждый элемент массива представляет собой бинарное число (0 или 1).

Для определения и обработки бинарных данных используются методы класса Binary:

- static public byte[] random(int len) — метод для генерации случайного бинарных данных длины len;
- static public byte[] setByBits(byte[] data) — метод для инициализации бинарных данных путем копирования вектора бинарных чисел data;
- static public byte[] setByInt(int num, int len) — метод для инициализации бинарных данных путем преобразования целого числа num в бинарные данные длиной len;

- static public byte[] setByLong(long num, int len) — метод для инициализации бинарных данных путем преобразования целого числа num в бинарные данные длиной len;
- static public byte[] setByLength(int len) — метод для инициализации бинарных данных длины len нулевыми значениями;
- static public byte[] setFromBinary(String binString) — метод для инициализации бинарных данных из строки содержащей число в бинарном формате;
- static public byte[] setFromHex(String hexString) — метод для инициализации бинарных данных из строки содержащей число в шестнадцатеричном формате;
- static public byte[] setFromString(String data) — метод для инициализации бинарных данных из строки путем перевода кодов символов в бинарные данные;
- static public String toBinString(byte[] data) — перевод бинарного значения data в строку, содержащую данные в бинарном формате;
- static public String toHexString(byte[] data) — перевод бинарного значения data в строку, содержащую данные в шестнадцатеричном формате;
- static public String toString(byte[] data) — перевод бинарного значения data в строку, содержащую символы с кодами из в массиве data;
- boolean equals(byte val1[],byte val2[]) — выполняет сравнение двух бинарных значений и возвращает истину если значения равны, иначе — ложь.

Для замера времени выполнения шифрования и дешифрования используется объекты класса TimeUtil. Для измерения времени выполнения определенных действий требуется до начала действий создать экземпляр класса TimeUtil, вызвать метод start(), после — вызвать метод finish(), после чего время в миллисекундах можно получить вызовом функции millisec().

Для передачи данных по каналу применяется метод void send(String node, Object data). В качестве параметров метода send указываются:

- node — узел процесса, выполняющего передачу данных;
- data — данные.

У процесса, который принимает данные вызывается обработчик события onRecieve. Для получения данных в обработчике onRecieve выполняется вызов метода Object recv(String node). Параметр node определяет в каком узле следует считывать данные. Если данные не обнаружены метод recv возвращает null, иначе будут возвращены полученные данные.

Метод do_send процесса S определяется следующим образом:

```
byte key[]=Binary.setFromHex("9B58 086D 9BF9 CD96 C6EA 3381
B1B4 F637");
byte IV[]=Binary.setFromHex("7836 ECD6 C5F0 37B6");
byte data[]=Binary.random(10000);
IDEA idea=new IDEA();
idea.setIV(IV);
idea.setKey(key);
logDataMessage("Открытый текст",data);
TimeUtil t=new TimeUtil();
t.start();
byte code[]=idea.encodeData(data,idea.MODE_CBC);
t.finish();
logDataMessage("Время шифрования ",t.millisec());
logDataMessage("Зашифрованный текст",code);
send("out",code);
```

Определение метода onRecieve процесса R представлено ниже.

```
byte key[]=Binary.setFromHex("9B58 086D 9BF9 CD96 C6EA 3381
B1B4 F637");
byte IV[]=Binary.setFromHex("7836 ECD6 C5F0 37B6");
IDEA idea=new IDEA();
idea.setIV(IV);
idea.setKey(key);
byte code[]=(byte[])recv("in");
TimeUtil t=new TimeUtil();
t.start();
byte data[]=idea.decodeData(code,idea.MODE_CBC);
t.finish();
logDataMessage("Время расшифрования",t.millisec());
logDataMessage("Расшифрованный текст",data);
```

2.3 Выполнение работы

1. Запустить RMI-реестр, выполнив runRegistry.bat
2. Запустить серверное приложение. Для этого требуется выполнить файл runServer.bat.

3. Для создания ресурса и субъекта открыть диалоговое окно «Элементы системы». Для этого следует выполнить команду меню «Модель\Статическая».
4. В окне редактирования статической модели на вкладке «Ресурсы» добавить ресурс, на вкладке «Субъекты» добавить субъект и указать доступный ресурс.
5. Открыть диаграмму взаимодействия процессов двойным щелчком на элементе «Динамическая модель\Диаграмма взаимодействия процессов» в древовидном списке.
6. Добавить у процесса, представляющего собой ресурс, выходной узел out. Для этого можно из контекстного меню группы «Узлы процесса» выполнить команду «Новый объект Узел», указать имя узла и его тип (in/out).
7. Добавить у процесса, представляющего собой субъект, входной узел in.
8. Для создания канала передачи данных на диаграмме взаимодействия процессов соединить узлы out и in процессов отправителя и получателя соответственно путем перетаскивания курсора мыши с нажатой левой кнопкой. Канал будет представлен на диаграмме в виде направленной стрелки от выходного узла к входному узлу.
9. Добавить у процесса, представляющего собой ресурс, метод do_send. Для этого можно из контекстного меню группы «Методы процесса» выполнить команду «Новый объект Метод».
10. Для определения метода do_send вызвать окно свойств метода двойным щелчком на методе в древовидном списке.
11. В соответствии с вариантом задания (см. таблицу 2.3.1) определить алгоритм работы метода do_send.
12. Определить алгоритм работы метода onRecieve процесса получателя сообщений.
13. Выполнить генерацию и компиляцию процессов, запустив команду «Генерировать» в контекстном меню процессов в древовидном списке. В случае, если в окне с сообщениями будут выведены ошибки, требуется их исправить и выполнить повторную генерацию.
14. Для выполнения моделирования запустить сервер моделирования, выполнив команду меню «Моделирование\Запуск сервера».
15. Запустить клиент моделирования для процесса отправителя на одной из рабочих станций лаборатории. Для этого следует выполнить исполняемый файл runClient.bat. В окне на рисунке 2.12 указать rmi путь к серверу моделирования.

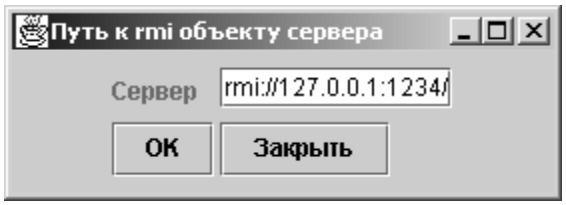


Рисунок 2.12 — Окно задания адреса удаленного сервера моделирования

16. В окне на рисунке 2.13. выбрать процесс S.

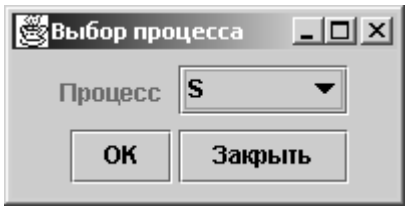


Рисунок 2.13 — Окно выбора процесса в клиенте моделирования

17. В окне клиента моделирования на рисунке 2.14 проконтролируйте правильность процесса регистрации клиента на сервере моделирования, создание процесса отправителя.

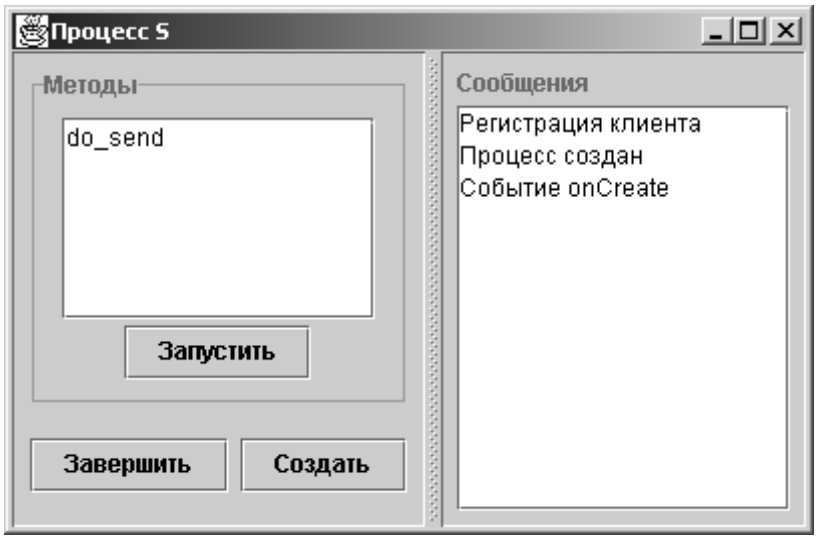


Рисунок 2.14 — Окно клиент моделирования

18. Запустить клиент моделирования для процесса получателя сообщения на одной из рабочих станций лаборатории.
19. Для моделирования передачи конфиденциальных данных запустите метод do_send из окна клиента моделирования процесса

отправителя. Для этого следует выбрать метод do_send в списке методов и выполнить команду «Запустить».

20. По сообщениям в протоколе работы клиентов моделирования и сообщениям, зарегистрированным сервером моделирования (см. «Моделирование\Сообщения» в древовидном списке приложения), проконтролируйте правильность конфиденциальной передачи данных, соответствие исходных данных расшифрованным данным и временные характеристики алгоритмов.
21. Сравнить быстродействие алгоритмов на основе нескольких замеров времени шифрования и дешифрования.
22. Проанализировать достоинства и недостатки применяемого режима сцепления.

Таблица 2.10 — Варианты заданий

Вариант	***Алгоритмы***	***Размер блока данных, байт***	***Режим сцепления***
1	*DES, IDEA*	*1000*	*Электронной шифровальной книги*
2		*1000*	*Сцепления шифровальных блоков*
3		*10000*	*Шифрованной обратной связи*
4		*10000*	*Обратной связи по выходу*

2.4 Содержание отчета

Отчет выполняется один на бригаду и должен включать:

1. Наименование и цель работы.
2. Краткие теоретические сведения.
3. Статическую модель согласно варианта.
4. Окна редактирования ресурсов, субъектов, угроз, уязвимостей, средств защиты.
5. Анализ защищенности
6. Выводы.

2.5 Контрольные вопросы к главе 2

1. Какие шаги алгоритма шифрования DES?
2. Режимы работы блочных шифров?

3 ЛАБОРАТОРНАЯ РАБОТА № 3

АЛГОРИТМЫ ШИФРОВАНИЯ С ОТКРЫТЫМ КЛЮЧОМ

3.1 Теоретические сведения

Применение алгоритмов шифрования с открытым ключом решает основную проблему симметричных алгоритмов шифрования распространения симметричного ключа между участниками системы.

В данном случае шифрования производится открытым ключом, а расшифрование — закрытым ключом.

Открытый и закрытый ключ связаны между собой, но не могут быть получены один из другого.

Схема шифрования сообщений с помощью алгоритма с открытым ключом представлена на рисунке 3.1.

Среди криптосистем с открытым ключом наиболее часто используются алгоритм шифрования RSA, алгоритм Эль-Гамаля.

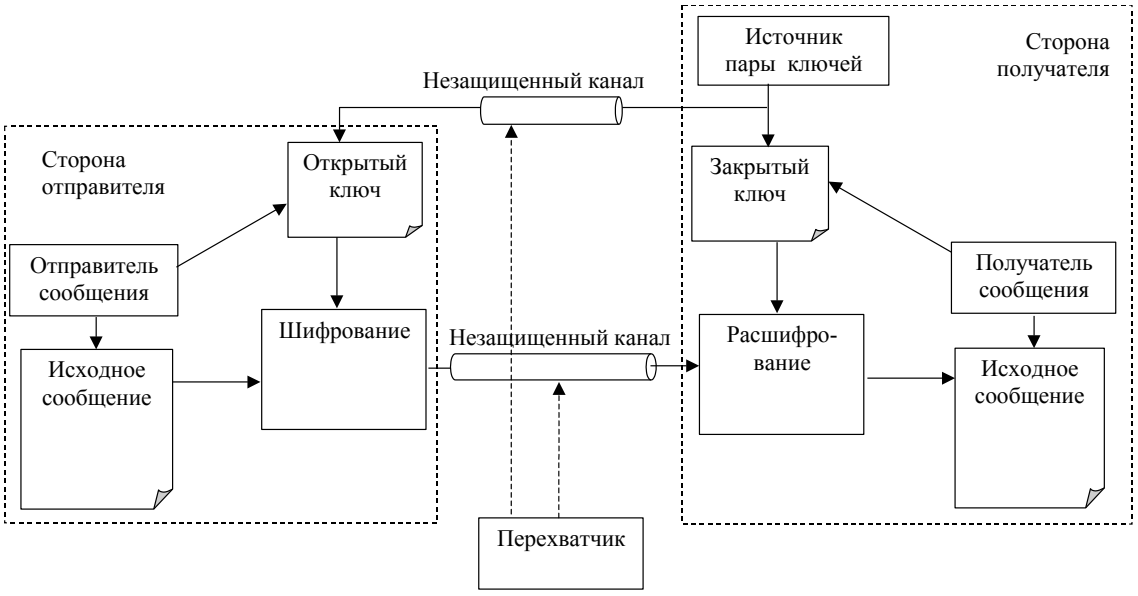


Рисунок 3.1 — Схема передачи сообщения с использованием алгоритма с открытым ключом

Основным достоинством криптосистемы с открытым ключом является то, что не требуется использовать защищенный канал для передачи

ключа. Недостатком криптосистемы считается большая потребность в вычислительных ресурсах и большой длине ключа.

3.1.1 Алгоритм шифрования RSA

Алгоритм RSA предложили в 1978 г. три автора: Р.Райвест (Rivest), А.Шамир (Shamir) и А.Адлеман (Aldeman). Алгоритм получил свое название по первым буквам фамилий его авторов. Алгоритм

RSA стал первым полноценным алгоритмом с открытым ключом, который может работать как в режиме шифрования данных, так и в режиме электронной цифровой подписи.

Надежность алгоритма основывается на трудности факторизации больших чисел и трудности вычисления дискретных логарифмов.

В криптосистеме RSA открытый ключ K_B секретный ключ k_B сообщение M и криптограмма C принадлежат множеству целых чисел

$Z_N = \{0, 1, 2, \dots , N-1\}$,

где N — модуль:

$N = P * Q$.

Здесь P и Q — случайные большие простые числа. Для обеспечения максимальной безопасности выбирают P и Q равной длины и хранят в секрете.

Множество Z_N с операциями сложения и умножения по модулю N образует арифметику по модулю N.

Открытый ключ K_B выбирают случайным образом так, чтобы выполнялись условия:

$1 < K_B \le \varphi (N)$, НОД $(K_B, \varphi (N)) = 1$,

$\varphi (N) = (P – 1) (Q – 1)$,

где $\varphi (N)$ — функция Эйлера.

Функция Эйлера $\varphi (N)$ указывает количество положительных целых чисел в интервале от 1 до N, которые взаимно просты с N.

Второе из указанных выше условий означает, что открытый ключ K_B и функция Эйлера $\varphi (N)$ должны быть взаимно простыми.

Далее, используя расширенный алгоритм Евклида, вычисляют секретный ключ k_B, такой, что

$k_B * K_B \equiv 1 (\mathrm{mod}\ \varphi (N))$

или

$k_B = K_B^{-1} (\text{mod } (P - 1)(Q - 1))$.

Это можно осуществить, если получатель B знает пару простых чисел (P, Q) и может легко найти $\varphi(N)$. Заметим, что k_B и N должны быть взаимно простыми.

Открытый ключ K_B используют для шифрования данных, а секретный ключ k_B — для расшифрования.

Преобразование шифрования определяет криптограмму C через пару (открытый ключ K_B, сообщение M) в соответствии со следующей формулой:

$$C = E_{K_B}(M) = E_B(M) = M^{K_B} (\text{mod} N)$$

В качестве алгоритма быстрого вычисления значения C используют ряд последовательных возведений в квадрат целого M с приведением по модулю N.

Предположим, что пользователь A хочет передать пользователю B сообщение в зашифрованном виде, используя криптосистему RSA.

В таком случае пользователь A выступает в роли отправителя сообщения, а пользователь B — в роли получателя. Криптосистему RSA должен сформировать получатель сообщения, т.е. пользователь B. Рассмотрим последовательность действий пользователя B и пользователя A.

Пользователь B выбирает два произвольно больших простых числа P и Q.

Пользователь B вычисляет значение модуля $N = P * Q$.

Пользователь B вычисляет функцию Эйлера

$\varphi(N) = (P - 1)(Q - 1)$

и выбирает случайным образом значение открытого ключа K_B с учетом выполненных условий:

$1 < K_B \le \varphi(N)$, НОД $(K_B, \varphi(N)) = 1$.

Пользователь B вычисляет значение секретного ключа k_B, используя расширенный алгоритм Эвклида при решении сравнения

$k_B = K_B^{-1} (\text{mod } \varphi(N))$.

Пользователь B пересылает пользователю A пару чисел (N, K_B) по незащищенному каналу.

Если пользователь А хочет передать пользователю В сообщение М, он выполняет следующие шаги.

Пользователь А разбивает исходный открытый текст М на блоки, каждый из которых может быть представлен в виде числа

$M_i = 0, 1, 2, \ldots, N - 1.$

Пользователь А шифрует текст, представленный в виде последовательности чисел M_i по формуле

$$C_i = M_i^{K_B} (\mathrm{mod} N)$$

и отправляет криптограмму

$C_1, C_2, C_3 \ldots C_i, \ldots$

пользователю В.

Пользователь В расшифровует принятую криптограмму

$C_1, C_2, C_3 \ldots C_i, \ldots$

используя секретный ключ k_B по формуле

$$M_i = C_i^{K_B} (\mathrm{mod} N)$$

В результате будет получена последовательность чисел M_i, которые представляют собой исходное сообщение М. Чтобы алгоритм RSA имел практическую ценность, необходимо иметь возможность без существенных затрат генерировать большие простые числа, уметь оперативно вычислять значения ключей k_B и K_B.

3.2 Применение компьютерной системы для изучения алгоритмов шифрования с открытым ключом

Для конфиденциальной передачи данных от отправителя к получателю с использованием алгоритма шифрования с открытым ключом требуется выполнить:

а) генерацию пары ключей (открытого и секретного ключей) на стороне получателя сообщения;

б) передачу открытого ключа отправителю сообщения;

в) шифрование сообщения открытым ключом;

г) отправку шифротекста получателю;

д) дешифрование с помощью секретного ключа получателем.

Для изучения используется передача данных от ресурса Источник субъекту Приёмник (см. рисунок 3.2).

Рисунок 3.2 — Статическая диаграмма, используемая при конфиденциальной передаче данных

При взаимодействии элементов Приёмник и Источник выполняется передача данных как от Приёмника к Источнику так и в обратную сторону. Приёмник моделируется в виде процесса R, Источник — в виде процесса S. Процесс R генерирует пару ключей: открытый ключ и секретный ключ, затем открытый ключ передается процессу S в незашифрованном виде. После получения открытого ключа процесс S выполняет шифрование исходного сообщения и его отправку процессу R. При получении шифротекста процессом R выполняется дешифрование сообщения с помощью секретного ключа.

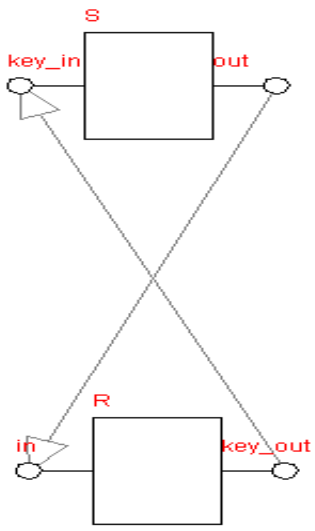


Рисунок 3.3 — Модель взаимодействия процессов при конфиденциальной передаче данных с использованием алгоритма шифрования с открытым ключом

Для генерации пары ключей будем использовать метод процесса R makeKeys. При получении открытого ключа вызывается обработчик onRecieve процесса S. Шифрование и отправка шифротекста выполняется в методе do_send процесса S.

В качестве примера рассмотрим шифрование целого числа алгоритмом RSA. В компьютерной системе разработан класс RSA.

Класс RSA предоставляет следующие методы:

- boolean setSecureKey(long p, long q) — установка секретного ключа, состоящего из целых чисел p и q (произведение чисел p и q должно быть меньше максимальной целого числа типа long);
- long getFi() — метод возвращает значение функции φ;
- boolean checkOpenKey(long e) — выполняет проверку значения открытого ключа на удовлетворение требований предьявляемых алгоритмом;
- void setOpenKey(long e,long n) — выполняет установку открытого ключа;
- long encode(long msg) — выполняет шифрование сообщения msg установленным открытым ключом с помощью функции setOpenKey;
- long decode(long msg) — выполняет дешифрование сообщения msg установленным секретным ключом с помощью функции setSecureKey.

Секретный и открытый ключи должны удовлетворять требованиям алгоритма RSA. Для проверки этих требований используется класс Numerical.

Методы класса Numerical:

- static boolean isPrime(long n) — проверяет является ли число n простым;
- static long gcd(long a, long b) — возвращает наибольший общий делитель чисел a и b;

- static long inverse(long a, long n) — возвращает мультипликативное обратное числу a по модулю n;
- static long power(long x, long n, long p) — возводит число x в степень n по модулю p.

В области данных процессов R и S определена переменная rsa класса RSA:

```
RSA rsa=new RSA();
```

Алгоритм работы метода makeKeys процесса R имеет следующий вид

```
long x       = 25000;
long y       = 30000;
long fi;
long p,q,e;
for ( p = x; !Numerical.isPrime( p ); p++ ) ;
for ( q = y + 2; !Numerical.isPrime( q ); q++ ) ;
rsa.setSecureKey(p,q);
logDataMessage("Секретный ключ","p=" + p +" q="+q);
fi=rsa.getFi();
for ( e = fi / 10; Numerical.gcd( e, fi ) != 1; e++ ) ;
rsa.checkOpenKey(e);
logDataMessage("Открытый ключ", "e=" + e+" n="+rsa.getN()
);
long open_key[]=new long[2];
open_key[0]=e;
open_key[1]=rsa.getN();
send("key_out",open_key);
```

Получение открытого ключа выполняется в методе-обработчике onRecieve процесса S:

```
long key[]=(long [])recv("key_in");
rsa.setOpenKey(key[0],key[1]);
logDataMessage("Получен открытый ключ","e="+key[0]+"
n="+key[1]);
```

Шифрование и отправка данных выполняются в методе do_send:

```
long message=12345435;
logDataMessage("Исходный текст",message);
long code = rsa.encode( message );
logDataMessage("Зашифрованный текст",code);
send("out",new Long(code));
```

Получение и дешифрование данных процессом R обрабатывается в методе onRecieve:

```
Long lcode=(Long)recv("in");
long code=lcode.longValue();
logDataMessage("Шифротекст",code);
long decode = rsa.decode( code );
logDataMessage("Расшифрованный текст",decode);
```

В результате моделирования получен протокол работы процессов.

Протокол работы процесса S имеет следующий вид:

```
Регистрация клиента
Процесс создан
Событие onCreate
Получены данные в узле S.key_in / in 75061013, 750665143
Событие onRecieve в узле key_in
Получен открытый ключ e=75061013 n=750665143
Запуск метода do_send
Исходный текст 12345435
Зашифрованный текст 583217134
Отправка данных с узла S.out / out 583217134
```

Протокол работы процесса R:

```
Регистрация клиента
Процесс создан
Событие onCreate
Запуск метода makeKeys
Секретный ключ p=25013 q=30011
Открытый ключ e=75061013 n=750665143
Отправка данных с узла R.key_out / out 75061013, 750665143
Получены данные в узле R.in / in 583217134
Событие onRecieve в узле in
Шифротекст 583217134
Расшифрованный текст 12345435
```

3.3 Выполнение работы

1. Запустить RMI-реестр, выполнив runRegistry.bat
2. Запустить серверное приложение. Для этого требуется выполнить файл runServer.bat.
3. Для создания ресурса и субъекта открыть диалоговое окно «Элементы системы». Для этого следует выполнить команду меню «Модель\Статическая».

4. В окне редактирования статической модели на вкладке «Ресурсы» добавить ресурс, на вкладке «Субъекты» добавить субъект и указать доступный ресурс.
5. Открыть диаграмму взаимодействия процессов двойным щелчком на элементе «Динамическая модель\Диаграмма взаимодействия процессов» в древовидном списке.
6. Добавить у процесса, представляющего собой ресурс, выходной узел out и входной узел key_in.
7. Добавить у процесса, представляющего собой субъект, входной узел in и выходной узел key_out.
8. Для создания канала передачи данных на диаграмме взаимодействия процессов соединить узлы out и in процессов отправителя и получателя соответственно путем перетаскивания курсора мыши с нажатой левой кнопкой.
9. Для создания канала для передачи открытого ключа на диаграмме взаимодействия процессов соединить узлы key_out и key_in процессов отправителя и получателя соответственно.
10. Добавить у процесса отправителя, представляющего собой ресурс, метод do_send, у процесса получателя — метод makeKeys.
11. Определить переменные в области данных процессов отправителя и получателя. Определение переменных выполняется в окне редактирования свойств процесса (см. рисунок 3.4).

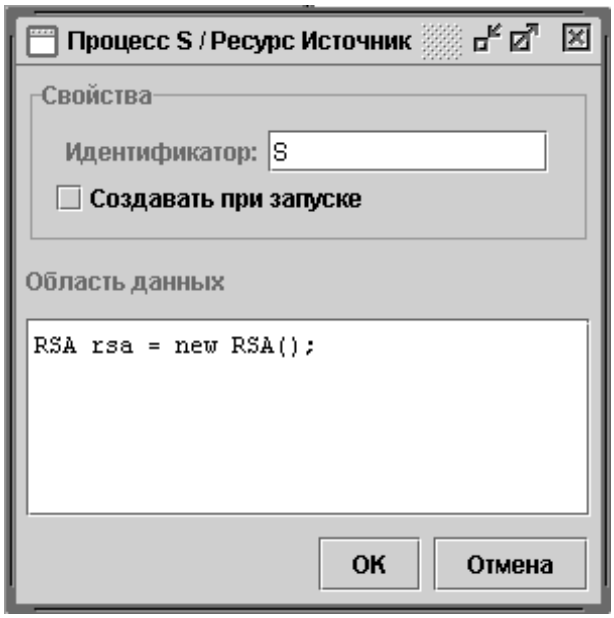


Рисунок 3.4 — Окно редактирования свойств процесса отправителя сообщения

12. Для определения метода makeKeys вызвать окно свойств метода двойным щелчком на методе в древовидном списке. В соответствии с вариантом задания (см. таблицу 3.1) определить алгоритм работы метода makeKeys. Данный метод выполняет генерацию пары ключей и отправку открытого ключа процессу отправителю.
13. Определить алгоритм работы метода onRecieve ресурса.
14. Определить алгоритм работы метода do_send процесса отправителя сообщения (ресурса), выполняющего шифрование и отправку данных получателю сообщения.
15. Определить алгоритм работы метода onRecieve процесса получателя, выполняющего получение и дешифрование данных.
16. Выполнить генерацию и компиляцию процессов, запустив команду «Генерировать» в контекстном меню процессов в древовидном списке. В том случае, если в окне с сообщениями будут выведены ошибки, требуется их исправить и выполнить повторную генерацию.
17. Для выполнения моделирования запустить сервер моделирования, выполнив команду меню «Моделирование\Запуск сервера».
18. Запустить клиент моделирования для процесса отправителя сообщения на одной из рабочих станций лаборатории.
19. Запустить клиент моделирования для процесса получателя сообщения на одной из рабочих станций лаборатории.
20. Для выполнения генерации и передачи открытого ключа отправителю требуется запустить работу метода makeKeys. По протоколам клиентов моделирования проконтролируйте правильность формирования и отправки ключей.
21. Для моделирования передачи конфиденциальных данных запустите метод do_send из окна клиента моделирования процесса отправителя.
22. По сообщениям в протоколе работы клиентов моделирования и сообщениям, зарегистрированным сервером моделирования (см. «Моделирование\Сообщения» в древовидном списке приложения), проконтролируйте правильность конфиденциальной передачи данных, соответствие исходных данных расшифрованным данным.
23. Сравнить быстродействие алгоритма на основе нескольких замеров времени шифрования и дешифрования при различной длине ключей.

Таблица 3.1 — Варианты заданий

Вариант	***Алгоритм***	***Ключ***	***Длина массива данных***
1	*RSA*	*От 1000 до 10000*	*100*
2		*От 1000 до 10000*	*1000*
3		*От 10000 до 20000*	*100*
4		*От 10000 до 20000*	*1000*

3.4 Содержание отчета

Отчет выполняется один на бригаду и должен включать:

1. Наименование и цель работы.
2. Краткие теоретические сведения.
3. Статическую модель согласно варианта.
4. Окна редактирования ресурсов, субъектов, угроз, уязвимостей, средств защиты.
5. Анализ защищенности
6. Выводы.

3.5 Контрольные вопросы к главе 3

1. Как работает алгоритм шифрования RSA?
2. Отличия и достоинства алгоритма Эль Гамаля?
3. На каких функциях основаны асимметричные алгоритмы?

4 ЛАБОРАТОРНАЯ РАБОТА № 4

СРЕДСТВА ИДЕНТИФИКАЦИИ И АУТЕНТИФИКАЦИИ

4.1 Теоретические сведения

Определим понятие идентификатора объекта: идентификатор — это некоторая информация, однозначно соответствующая этому объекту. Идентификация в таком случае является процессом определения идентификатора объекта.

Аутентификация — проверка подлинности, а именно определение является ли объект тем, за кого он себя представляет. Успешная аутентификация ведет к процессу предоставления полномочий — авторизации.

При применении пароля для подтверждения подлинности пароль преобразуется с помощью односторонней функции перед посылкой по незащищенному каналу, а затем отображение пароля сравнивается с отображением на стороне получателя. Если отображения совпадают, пароль считается подлинным, а объект законным.

Для взаимной проверки подлинности обычно используют процедуру «рукопожатия». В данном случае выполняется взаимная проверка ключей, используемых сторонами.

Рассмотрим процедуру «рукопожатия» двух пользователей *A* и *B*, основанную на симметричном алгоритме шифрования (рис 4.1):

- пользователь *A* инициирует процедуру рукопожатия, отправляя пользователю *B* свой идентификатор ID_A в открытой форме;
- пользователь *B*, получив идентификатор ID_A находит в базе данных секретный ключ K_{AB} и вводит его в свою криптосистему;
- пользователь *A* генерирует случайную последовательность *S* с помощью генератора и отправляет её пользователю B в виде криптограммы $E_{K_{AB}}(S)$;
- пользователь *B* расшифровывает эту криптограмму и раскрывает исходный вид последовательность *S*;
- оба пользователя применяют к S одностороннюю функцию $\alpha(S)$;
- пользователь *B* шифрует сообщение $\alpha(S)$ и отправляет эту криптограмму пользователю *A*.

- пользователь *A* расшифровывает эту криптограмму и сравнивает полученное сообщение с исходным, если эти значения равны пользователь А признает подлинность пользователя *B*.

Пользователь *B* аналогично выполняет проверку подлинности пользователя *A*.

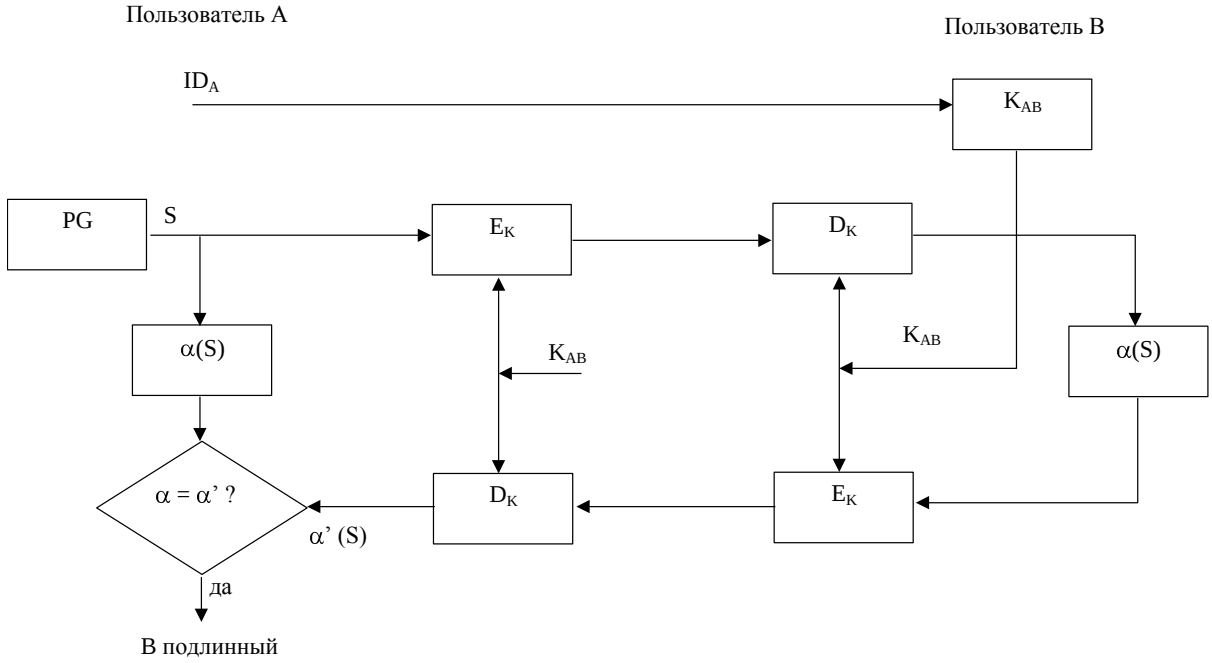


Рисунок 4.1 — Схема проверки подлинности с помощью процедуры «рукопожатия»

Хэш-функция предназначена для сжатия подписываемого документа M до нескольких десятков или сотен бит. Хэш-функция h() принимает в качестве аргумента сообщение (документ) M произвольной длины и возвращает хэш-значение h(M) = H фиксированной длины. Обычно хэшированная информация является сжатым двоичным предтавлением одного сообщения произвольной длины. Следует отметить, что значение хэш-функция h(M) сложным образом зависит от документа M и не позволяет восстановить сам документ M.

Хэш-функция должна удовлетворять целому ряду условий:

- хэш-функция должна быть чувствительна к всевозможным изменениям в тексте M, таким как вставки, выбросы, перестановки и т.п.;
- хэш-функция должна обладать свойством необратимости, то есть задача подбора документа M', который обладал бы требуемым

значением хэш-функции, должна быть вычислительно неразрешима;

- вероятность того, что значения хэш-функций двух различных документов (вне зависимости от их длин) совпадут, должна быть ничтожно мала.

Большинство хэш-функций строится на основе однонаправленной функции f(), которая образует выходное значение длиной n при задании двух входных значений длиной2 n. Этими входами являются блок исходного текста M_i и хэш-значение H_{i-1} предыдущего блока текста (см. рисунок 4.2):

$H_i = f(M_i, H_{i-1})$.

Хэш-значение, вычисляемое при вводе последнего блока текста, становится хэш-значением всего сообщения M.

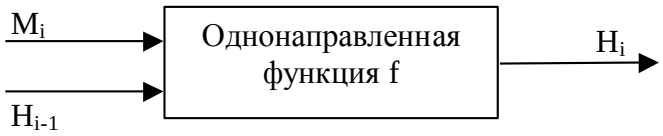


Рисунок 4.1 — Построение однонаправленной хэш-функции

В результате однонаправленная хэш-функция всегда формирует выход фиксированной длины n (независимо от длины входного текста).

4.2 Применение компьютерной системы для изучения протоколов идентификации и аутентификации

При исследовании схемы идентификации и аутентификации с помощью процедуры рукопожатия будем использовать следующие алгоритмы:

- для шифрования и дешифрования случайного значения S — алгоритм симметричного шифрования DES;
- для получения хэш значения от S — алгоритм вычисления дайджеста (профиля) сообщения MD5.

Рассмотрим криптографические блоки, реализующие функции получения дайджеста. Построение классов, реализующих хэш-функции, базируется на наборе стандартных функций определенных в классе BaseHash.

Класс BaseHash предоставляет пользователю следующие функции:

- void reset() — выполняет сброс внутренних данных для последующего получения хэш-значений;
- boolean compare(byte dig1[],byte dig2[]) — выполняет сравнение двух хэш-значений и возвращает истинное значение, если значения равны, иначе — ложное значение.

В данном классе также определены абстрактные функции, которые реализуются в классах-потомках.

Алгоритм хэширования MD5 реализован в виде класса MD5.

Основные функции класса MD5:

- MD5() — конструктор;
- void update(byte[] b, int offset, int len) — выполняет установку исходных данных для получения хэш-значения, offset — смещение в буфере данных b, len — длина данных;
- byte[] digest() — возвращает дайджест данных, установленных с помощью функции update, данные возвращаются в виде 16-байтового массива;
- byte[] convertBytes(byte d[]) — преобразует данные из 16-байтового массива в бинарный массив из 128 байт, где каждый байт определяет значение одного бита.

Для удобства получения хэш-значений разработаны статические методы:

- byte[] digest(String s) — возвращает дайджест строки s в виде бинарного массива из 128 байт;
- byte[] digest(byte b[]) — возвращает дайджест бинарных данных b в виде бинарного массива из 128 байт.

Пример получения дайджеста сообщения:

```
byte data[]= Binary.setFromHex("BBBB CCCC 4444 6666")
byte digest[]=MD5.digest(data);
```

Рассмотрим задачу идентификации и аутентификации субъекта B при обращении к ресурсу A.

Статическая модель представлена на рисунке 4.3.

Рисунок 4.3 — Статическая модель схемы для изучения схемы идентификации и аутентификации

Процессы, представляющие собой субъекты и ресурсы, выполняют взаимодействие по каналам передачи данных:

- канал передачи идентификатора от процесса A процессу B;
- канал передачи криптограммы случайной последовательности S от процесса A процессу B;
- канал передачи криптограммы дайджеста последовательности S от процесса B процессу A, полученной путем применения шифрования к значению односторонней функции от S.

Диаграмма взаимодействия процессов представлена на рисунке 4.4.

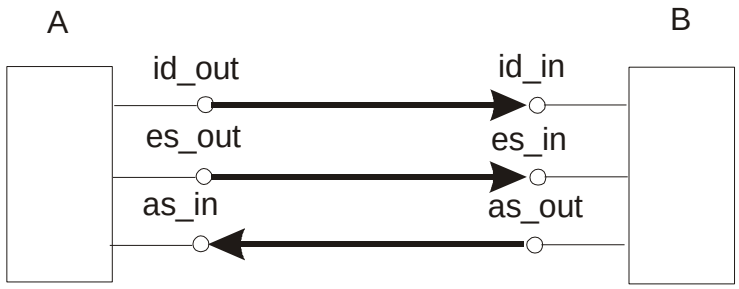


Рисунок 4.4 — Диаграмма взаимодействия процессов для изучения схемы идентификации и аутентификации

В области данных процесса A определим следующие переменные:

```
int id=1;
byte key[]=Binary.setFromHex("BBBB CCCC 4444 6666");
byte s[];
```

где s — переменная для хранения случайной последовательности S;
key — ключ процесса A;
id —идентификатор процесса A.

В области данных процесса B определены переменные:

```
BinaryVector ids=new BinaryVector();
byte key[];
byte s[];
```

где ids — список ключей для идентификаторов;
key — переменная для хранения ключа, соответствующего идентификатору процесса A;
s — переменная для хранения случайной последовательности.

Для управления процессами передачи сообщений у процесса A определены методы:

- sendId — выполняет отправку идентификатора процесса A;
- sendES — выполняет генерацию последовательности S, шифрование и отправку криптограммы процессу B.

У процесса B определен метод sendAS, выполняющий формирование дайджеста последовательности S, шифрование и отправку результата процессу A.

Определения методов процесса A приведено в таблице 4.1.

Таблица 4.1 — Методы процесса A

Метод	**Определение метода**
sendId	send("id_out",new Integer(id));

sendES	s=Binary.random(64);
	byte es[]=DES.encode(key,s);
	send("es_out",es);

onRecieve	if (hasMoreData("as_in"))
	{
	byte ds[]=MD5.digest(s);
	
	byte as_r[]=(byte[])recv("as_in");
	
	DES des = new DES();
	des.setKey(key);
	byte ds_r[] = des.decodeData(as_r,BlockCipher.MODE_ECB);
	
	if (BaseHash.compare(ds,ds_r))
	logMessage("B подлинный");
	else
	logMessage("B ложный");
	}

Определения методов процесса B представлены в таблице 4.2.

Таблица 4.2 — Методы процесса B

Метод	Определение метода
onCreate	ids.put(Binary.setFromHex("AAAA 1111 2222 3333"),0); ids.put(Binary.setFromHex("BBBB CCCC 4444 6666"),1);

onRecieve	if (hasMoreData("id_in"))
	{
	int id=((Integer)recv("id_in")).intValue();
	key=ids.get(id);
	} else if (hasMoreData("es_in"))
	{
	byte es[]=(byte[])recv("es_in");
	s=DES.decode(key,es);
	}

SendAS	byte ds[]=MD5.digest(s);
	DES des = new DES();
	des.setKey(key);
	byte as[] = des.encodeData(ds,BlockCipher.MODE_ECB);
	send("as_out",as);

Для моделирования описанного протокола требуется выполнить действия:

а) запустить сервер моделирования;

б) запустить клиент моделирования для процесса A;

в) запустить клиент моделирования для процесса B;

г) запустить метод процесса A sendId;

д) запустить метод процесса A sendES;

е) запустить метод процесса B sendAS.

В результате будет получено сообщение о проверке подлинности процесса B. Для проверки подлинности процесса A требуется дополнить модель аналогичной схемой.

4.3 Выполнение работы

1. Определить модель схемы идентификации и аутентификации, указанную в разделе 4.2.
2. Выполнить моделирование схемы идентификации и аутентификации. Зафиксировать результат аутентификации и промежуточные данные.
3. Изменить идентификатор процесса А на 0 в области данные процесса А. Провести повторное моделирование и зафиксировать результат и промежуточные данные.
4. Расширить модель аутентификацией процесса А и зафиксировать результат.

4.4 Содержание отчета

Отчет выполняется один на бригаду и должен включать:

1. Наименование и цель работы.
2. Краткие теоретические сведения.
3. Статическую модель согласно варианта.
4. Окна редактирования ресурсов, субъектов, угроз, уязвимостей, средств защиты.
5. Анализ защищенности
6. Выводы.

4.5 Контрольные вопросы к главе 4

1. В чем заключается процедура «рукопожатия» во взаимной проверке подлинности?
2. Что такое хэш-функция? Ее назначение.
3. Какие способы построения однонаправленной хэш-функции?

5 ЛАБОРАТОРНАЯ РАБОТА № 5

ФОРМАЛЬНЫЕ ПОЛИТИКИ БЕЗОПАСНОСТИ

Теоретические сведения

Потребители путем составления формальных моделей безопасности получают возможности довести до сведения производителей свои требования в четко определенной и непротиворечивой форме, а также оценить соответствие защищенных систем своим потребностям.

Эксперты по квалификации в ходе анализа адекватности реализации политики безопасности в защищенных системах используют модели безопасности в качестве эталонов.

Рассматриваемые далее модели безопасности основаны на следующих базовых представлениях:

а) система является совокупностью взаимодействующих сущностей — субъектов и объектов;

б) все взаимодействия в системе моделируются установлением отношений определенного типа между субъектами и объектами (множества типов отношений определяется в виде набора операций, которые субъекты могут производить над объектами);

в) все операции контролируются монитором взаимодействий и либо запрещаются, либо разрешаются в соответствии с правилами политики безопасности;

г) политика безопасности задается в виде правил, в соответствии с которыми должны осуществляться все взаимодействия между субъектами и объектами;

д) совокупность множеств субъектов, объектов и отношений между ними (установившихся взаимодействий) определяет состояние системы, которое является либо безопасным, либо небезопасным в соответствии с предложенным в модели критерием безопасности;

е) основной элемент модели безопасности — это доказательство утверждения (теоремы) о том, что система, находящаяся в безопасном состоянии, не может перейти в небезопасное состояние при соблюдении всех установленных правил и ограничений.

Объекты можно интуитивно представлять в виде контейнеров, содержащих информацию, а субъектами считать выполняющиеся

программы, которые воздействуют на объекты различными способами. При таком представлении системы безопасность обработки информации обеспечивается путем решения задачи управления доступом субъектов к объектам в соответствии с заданным набором правил и ограничений, которые образуют политику безопасности. Считается, что система безопасна, если субъекты не имеют возможности нарушить правила политики безопасности.

Среди моделей политик безопасности можно выделить два основных класса: дискреционные (произвольные) и мандатные (нормативные). Наиболее распространены политики произвольного управления доступом, в основе которых лежат модель Харрисона-Руззо-Ульмана и модель типизованной матрицы доступа, фундаментальную нормативную модель безопасности Белла-ЛаПадулы, а также модель ролевой политики.

5.1.1 Дискреционная модель безопасности Харрисона-Руззо-Ульмана

Система обработки информации представляется в виде совокупности активных субъектов *S*, пассивных объектов O и конечного множества прав доступа $R=\{r_1, \ldots, r_n\}$ (рисунок 5.1.1). Субъекты осуществляют доступ к информации, объекты содержат информацию, права доступа означают полномочия на выполнение соответствующих действий (чтение, запись, выполнение). Еще одним моментом является то, что все субъекты являются одновременно и объектами, это позволяет представлять права доступа между субъектами $S \subset O$.

Текущее состояние системы в пространстве состояний $O \times S \times R$ определяется тройкой $Q = (S, O, M)$, где M — матрица прав доступа субъектов к объектам. Строки матрицы соответствуют субъектам, а столбцы — объектам. Элемент матрицы $M[s, o]$ содержит набор прав доступа субъекта s к объекту o, принадлежащих множеству прав доступа *R*.

Переход между различными состояниями осуществляется путем внесения изменений в матрицу M с помощью команд.

Будем описывать команды в следующем виде:

$command \quad \alpha(x_1, \ldots, x_k)$

$if \quad r_1 \; in \; M[x_{s_1}, x_{o_1}] \; and$

$r_2 \; in \; M[x_{s_2}, x_{o_2}] \; and$

.

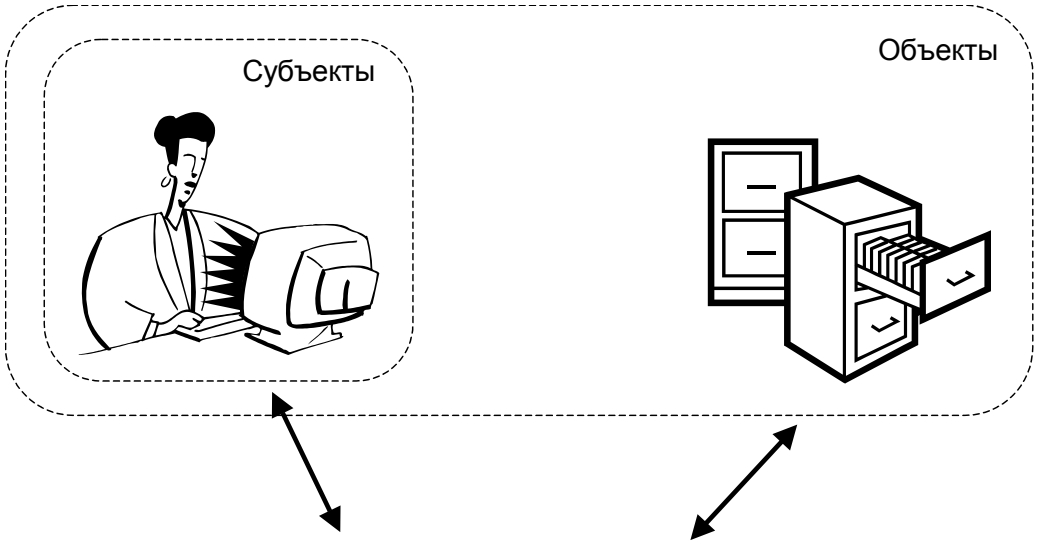
.

.

$r_m \; in \; M[x_{s_m}, x_{o_m}]$

then

$op_1, op_2, ..., op_n$



S \ O	O_1	O_2	S_1	S_2
S_1	{r_1 r_2}	∅	—	{r_2 r_3}
S_2	∅	{r_1}	{r_2}	—

Матрица доступа

Рисунок 5.1 — Дискреционная модель безопасности Харрисона-Руззо-Ульмана

Здесь α — имя команды; $x_1, ..., x_k$ — параметры команды, являющиеся идентификаторами субъектов и объектов; s_i и o_i — индексы субъектов и объектов; op_i — элементарные операции, описанные ниже.

Элементарные операции составляющие команду, выполняются только в том случае, если все условия, означающие присутствие указанных прав доступа в ячейках матрицы *M*, являются истинными.

Классическая модель допускает только следующие элементарные операции:

- *enter r into M[s,o]* — добавление субъекту s права r для объекта o;
- *delete r from M[s,o]* — удаление у субъекта a права r для объекта o;
- *create subject s* — создание нового субъекта s;
- *create object o* — создание нового объекта o;
- *destroy subject s* — удаление существующего субъекта s;
- *destroy object o* — удаление существующего объекта o.

Применение любой элементарной операции *op* в системе, находящейся в состоянии $Q=(S,O,M)$ влечет за собой переход в другое состояние $Q'=(S',O',M')$, которое отличается от предыдущего состояния Q по крайней мере одним компонентом. Выполнение базовых операций приводит к следующим изменениям в состоянии системы:

enter r into M[s, o] (где $s \in S, o \in O$ *)*

$O' = O$

$S' = S$

$M'[x_s,x_o] = M[x_s,x_o]$, *если* $(x_s,x_o) \neq (s,o)$

$M'[s,o] = M[s,o] \cup \{r\}$

delete r from M[s, o] (где $s \in S, o \in O$ *)*

$O' = O$

$S' = S$

$M'[x_s,x_o] = M[x_s,x_o]$, *если* $(x_s,x_o) \neq (s,o)$

$M'[s,o] = M[s,o] \setminus \{r\}$

create subject s (где $s \notin S$*)*

$O' = O \cup \{s\}$

$S' = S \cup \{s\}$

$M'[x_s,x_o] = M[x_s,x_o]$, *для всех* $(x_s,x_o) \in S \times O$

$M'[s,x_o] = \varnothing$ *для всех* $x_o \in O'$

$M'[s,x_s] = \varnothing$ *для всех* $x_s \in S'$

destroy subject s (где $s \in S$*)*

$O' = O \setminus \{s\}$

$S' = S \setminus \{s\}$

$M'[x_s,x_o] = M[x_s,x_o]$, *для всех* $(x_s,x_o) \in S' \times O'$

create object o (где $o \notin O$*)*

$O' = O \cup \{o\}$

$S' = S$

$M'[x_s,x_o] = M[x_s,x_o]$, *для всех* $(x_s,x_o) \in S \times O$

$M'[x_s,x_o] = \varnothing$ *для всех* $x_s \in S'$

destroy object o (где $o \in O$*)*

$O' = O \setminus \{o\}$

$S' = S$

$M'[x_s,x_o] = M[x_s,x_o]$, *для всех* $(x_s,x_o) \in S' \times O'$

Формальное описание системы состоит из следующих элементов:

- конечный набор прав доступа $R=\{r_1, \ldots, r_n\}$;
- конечные наборы исходных субъектов и объектов $S_0=\{s_1, \ldots, s_l\}$ и объектов $O_0=\{o_1, \ldots, o_m\}$, где $S_0 \subseteq O_0$;
- исходная матрица доступа, содержащая права доступа субъектов к объектам — M_0;
- конечный набор команд $C=\{\alpha_i(x_1,..,x_k)\}$, каждая из которых состоит из условий выполнения и интерпретации в терминах вышеперечисленных элементарных операций.

Поведение системы во времени моделируется с помощью последовательности состояний $\{Q_i\}$, в которой каждое последующее состояние является результатом применения некоторой команды из множества C к предыдущему $Q_{n+1}= C_n(Q_n)$. Таким образом, для заданного начального состояния только от условий команд из C и составляющих их операций зависит, сможет ли система попасть в то или иное состояние, или нет. Каждое состояние определяет отношения доступа, которые существуют между сущностями системы в виде множества субъектов, объектов и матрицы прав. Поскольку для обеспечения безопасности необходимо наложить запрет на некоторые отношения доступа, для заданного начального состояния системы должна существовать возможность определить множество состояний, в которые она сможет из него попасть. Это позволит задавать такие начальные условия (интерпретацию команд C, множество объектов O_0, субъектов S_0 и матрицу доступа M_0), при которых система никогда не сможет попасть в состояния, нежелательные с точки зрения безопасности.

Все дискреционные модели уязвимы по отношению к атаке с помощью «троянского коня», поскольку в них контролируются только

операции доступа субъектов к объектам, а не потоки информации между ними. Возможна ситуация, когда нарушитель подставит пользователю программу, переносящую информацию из доступного этому пользователю объекта в объект, доступный нарушителю. В данном случае формальное правило дискреционной политики безопасности не нарушается, но утечка информации происходит.

5.1.1 Мандатная модель Белла-ЛаПадулы

Мандатная модель управления доступом основана на правилах секретного документооборота, принятых в государственных и правительственных учреждениях многих стран. Основным положением политики Белла-ЛаПадулы, взятым им из реальной жизни, является назначение всем участникам процесса обработки защищаемой информации, и документам, в которых она содержится, специальной метки, например, секретности, совершенно секретно и т. д, получившей название уровня безопасности. Все уровни безопасности упорядочиваются с помощью установленного отношения доминирования, например, уровень совершенно секретно считается более высоким, чем уровень секретно, или доминирует над ним. Контроль доступа осуществляется в зависимости от уровней безопасности взаимодействующих сторон на основании двух простых правил:

- уполномоченное лицо (субъект) имеет право читать только те документы, уровень безопасности которых не превышает его собственный уровень безопасности;
- уполномоченное лицо (субъект) имеет право заносить информацию только в те документы, уровень безопасности которых не ниже его собственного уровня безопасности.

Первое правило обеспечивает защиту информации, обрабатываемой более доверенными (высокоуровневыми) лицами, от доступа со стороны менее доверенных (низкоуровневых). Второе правило предотвращает утечку информации (сознательную или несознательную) со стороны высокоуровневых участников процесса обработки информации к низкоуровневым.

Таким образом, если в дискреционных моделях управление доступом происходит путем наделения пользователей полномочиями осуществлять определенные операции над определенными объектами, то мандатные модели управляют доступом неявным образом — с помощь назначения всем

сущностям системы уровней безопасности, которые определяют все допустимые взаимодействия между ними. Следовательно, мандатное управление доступом не различает сущностей, которым присвоен одинаковый уровень безопасности, и на их взаимодействия ограничения отсутствуют. Поэтому в тех ситуациях, когда управление доступом требует более гибкого подхода, мандатная модель применяется совместно с какой-либо дискреционной, которая используется для контроля за взаимодействиями между сущностями одного уровня и для установки дополнительных ограничений, усиливающих мандатную модель.

Система в модели безопасности Белла-ЛаПадулы, как и в модели Харрисона-Руззо-Ульмана, представляется в виде множеств субъектов *S*, объектов *O* (множество объектов включает множество субъектов, $S \subset O$) и прав доступа read (чтение) и write (запись). В мандатной модели рассматриваются только эти два вида доступа, и, хотя она может быть расширена введением дополнительных прав (например, правом на добавление информации, выполнение программ и т.д.), все они будут отображаться в базовые (чтение и запись). Использование столь жесткого подхода, не позволяющего осуществлять гибкое управление доступом, объясняется тем, что в мандатной модели контролируются не операции, осуществляемые субъектом над объектом, а потоки информации, которые могут быть только двух видов: либо от субъекта к объекту (запись), либо от объекта к субъекту (чтение).

Уровни безопасности субъектов и объектов задаются с помощью функции уровня безопасности $F:S \cup O \rightarrow L$, которая ставит в соответствие каждому объекту и субъекту уровень безопасности, принадлежащий множеству уровней безопасности L, на котором определена решетка уровней безопасности.

Решетка уровней безопасности — это формальная алгебра $(L, \leq, \bullet, \otimes)$, где $\leq$ — частичное нестрогое отношение порядка для элементов множества уровней безопасности *L*, $\bullet$ — наименьшая верхняя граница и $\otimes$ — наибольшая нижняя граница. Для каждой пары элементов *L* можно указать единственный элемент, ограничивающий ее сверху или снизу таким образом, что между ними и этим элементом не будет других элементов. Использование решетки для описания отношений между уровнями безопасности позволяет в качестве атрибутов безопасности использовать сложные составные элементы, для которых определено отношение «меньше или равно».

Функция уровня безопасности вместе с решеткой уровней безопасности определяют состояние системы. Состояния системы делятся на безопасные и небезопасные, в которых нарушаются правила.

Состояние называется безопасным по чтению в том случае, когда для каждого субъекта, осуществляющего в этом состоянии доступ чтения к объекту, уровень безопасности этого субъекта доминирует над уровнем безопасности объекта.

Состояние называется безопасным по записи в том случае, когда для каждого субъекта, осуществляющего в этом состоянии доступ записи к объекту, уровень безопасности этого объекта доминирует над уровнем безопасности субъекта

Состояние безопасно, когда оно является безопасным по чтению и записи.

Исходя из предыдущих определений, критерий безопасности выглядит следующим образом:

> ***Система безопасна тогда и только тогда, когда её начальное состояние безопасно и все состояния, достижимые путем применения конечной последовательности запросов безопасны.***

5.1.2 Ролевая политика безопасности

Ролевая политика безопасности представляет собой существенно усовершенствованную модель Харрисона-Руззо-Ульмана, однако ее нельзя отнести ни к дискреционным, ни к мандатным, потому что управление доступом в ней осуществляется как на основе матрицы прав доступа для ролей, так и с помощью правил, регламентирующих назначение ролей пользователям и их активацию во время сеансов. Поэтому ролевая модель представляет собой совершенно особый тип политики, основанной на компромиссе между гибкостью управления доступом, характерной для дискреционных моделей, и жесткостью правил контроля доступа, присущей мандатным моделям.

В ролевой модели классическое понятие субъект и замещается понятиями пользователь и роль. Пользователь — это человек, работающий с системой и выполняющий определенные служебные обязанности. роль — это активно действующая в системе абстрактная сущность, с которой связан ограниченный, логически связанный набор полномочий, необходимых для осуществления определенной деятельности. Самым распространенным примером роли является присутствующий почти в каждой системе

административный бюджет (например root для UNIX и Administrator для Windows NT), который обладает специальными полномочиями и может использоваться несколькими пользователями.

Ролевая политика распространена очень широко, потому что она, в отличие от других более строгих и формальных политик, очень близка к реальной жизни. Ведь на самом деле работающие в системе пользователи действуют не от своего личного имени — они всегда осуществляют определенные служебные обязанности, т. е. выполняют некоторые роли, которые никак не связаны с их личностью.

Поэтому вполне логично осуществлять управление доступом и назначать полномочия не реальным пользователям, а абстрактным (не персонифицированным) ролям, представляющим участников определенного процесса обработки информации. Такой подход к политике безопасности позволяет учесть разделение обязанностей и полномочий между участниками прикладного информационного процесса, т. к. с точки зрения ролевой политики имеет значение не личность пользователя, осуществляющего доступ к информации, а то, какие полномочия ему необходимы для выполнения его служебных обязанностей. Например, в реальной системе обработки информации могут работать системный администратор, менеджер баз данных и простые пользователи.

В такой ситуации ролевая политика позволяет распределить полномочия между этими ролями в соответствии с их служебными обязанностями: роли администратора назначаются специальные полномочия, позволяющие ему контролировать работу системы и управлять ее конфигурацией, роль менеджера баз данных позволяет осуществлять управление сервером баз данных, а права простых пользователей ограничиваются минимумом, необходимым для запуска прикладных программ. Кроме того, количество ролей в системе может не соответствовать количеству реальных пользователей — один пользователь, если на нем лежат различные обязанности, требующие различных полномочий, может выполнять (одновременно или последовательно) несколько ролей, а несколько пользователей могут пользоваться одной и той же ролью, если они выполняют одинаковую работу.

При использовании ролевой политики управление доступом осуществляется в две стадии: во-первых, для каждой роли указывается набор полномочий, представляющий набор прав доступа к объектам, и, во-вторых, каждому пользователю назначается список доступных ему ролей. Полномочия назначаются ролям в соответствии с принципом наименьших привилегий, из которого следует, что каждый пользователь должен обладать

только минимально необходимым для выполнения своей работы набором полномочий.

Ролевая модель описывает систему в виде следующих множеств:

U — множество пользователей;

R — множество ролей;

P — множество полномочий на доступ к объектам, представленное, например, в виде матрицы прав доступа;

S — множество сеансов работы пользователей с системой.

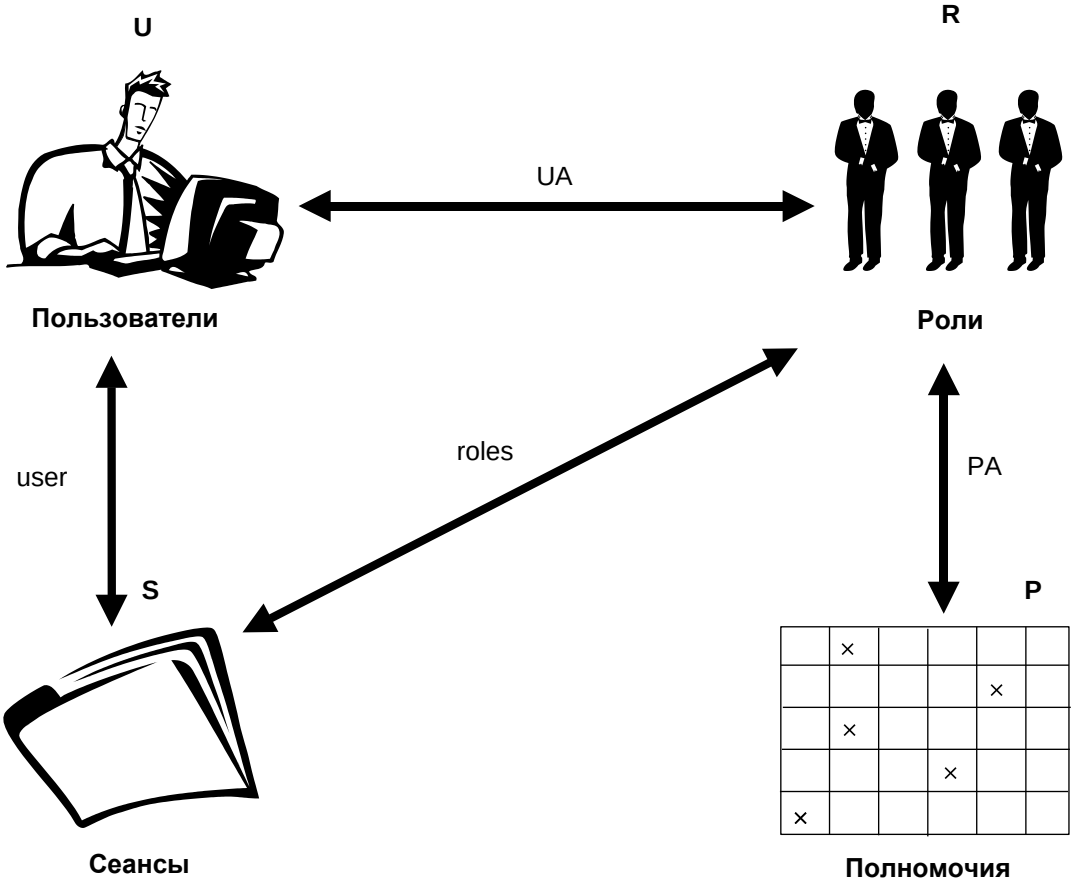


Рисунок 5.1 — Ролевая модель управления доступом

Для перечисленных множеств определяются следующие отношения (рис. 5.2):

$PA \subseteq P \times R$ — отображает множество полномочий на множество ролей, устанавливая для каждой роли набор присвоенных ей полномочий;
$UA \subseteq U \times R$ — отображает множество пользователей на множество ролей, определяя для каждого пользователя набор доступных ему ролей.

Правила управления доступом ролевой политики безопасности определяются следующими функциями:

- *user: S → U* - для каждого сеанса *s* эта функция определяет пользователя, который осуществляет этот сеанс работы с системой: *user(s)=u*;
- roles: S → P(R) *для каждого сеанса* s *эта функция определяет набор ролей из множества* R *которые могут быть одновременно доступны пользователю в этом сеансе:* roles(s)={r_i | (user(s),r_i) ∈ UA}*;*
- permissions: S → P — *для каждого сеанса* s *эта функция задает набор доступных в нем полномочий, который определяется как совокупность полномочий всех ролей, задействованных в этом сеансе:* permissions(s) = $U_{r \in roles(s)}$ {p_i | (p_i, r) ∈ PA}

В качестве критерия безопасности ролевой модели используется следующее правило: система считается безопасной, если любой пользователь, работающий в сеансе *s*, может осуществлять действия, требующие полномочия p только в том случае, если *p ∈ permissions (s)*.

5.2 Применение компьютерной системы для изучения формальных политик безопасности

В компьютерной системе реализована дискреционная модель безопасности Харрисона-Руззо-Ульмана.

Для определения и изучения политики безопасности необходимо активизировать модуль, реализующий политику безопасности. Активизация выполняется путем включения опции в диалоговом окне «Модели политики безопасности», в результате чего в древовидном списке появляется элемент «Дискреционная модель Харрисона-Руззо-Ульмана» (см. рисунок 5.3).

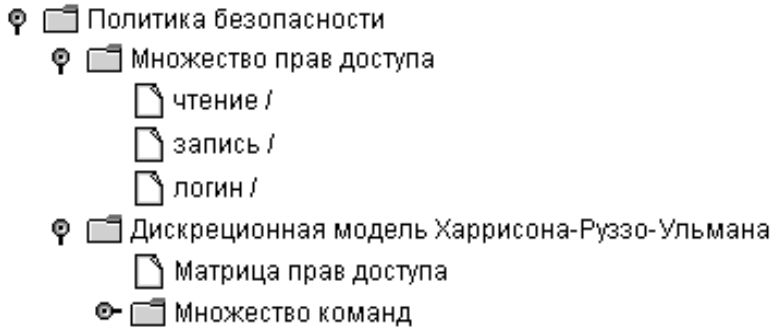


Рисунок 5.3 — Древовидный список политики безопасности

Множество прав доступа состоит из всех прав, которые могут делегироваться субъектам и объектам. Для добавления нового права доступа следует выбрать команду «Новый объект Право доступа» в контекстном меню множества прав доступа. Удаление выполняется путем выполнения команды «Удалить» в контекстном меню права доступа, которое требуется исключить.

Для задания матрицы прав доступа используется окно, изображенное на рисунке 5.4.

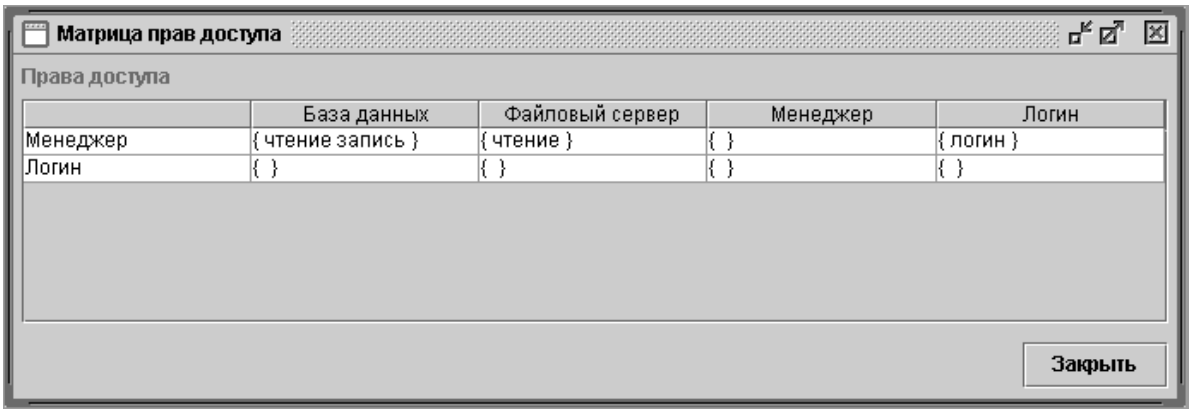


Рисунок 5.4 — Окно задания матрицы прав доступа

В ячейках таблицы указывается перечень прав доступа субъекта, указанного в первой колонке, к объекту, указанному впервой строке таблицы. К объектам относятся все ресурсы, а также все субъекты, определенные в статической модели.

Редактирование перечня прав доступа выполняется в окне, представленном на рисунке 5.5, путем двойного нажатия мышью в соответствующей ячейке таблицы.

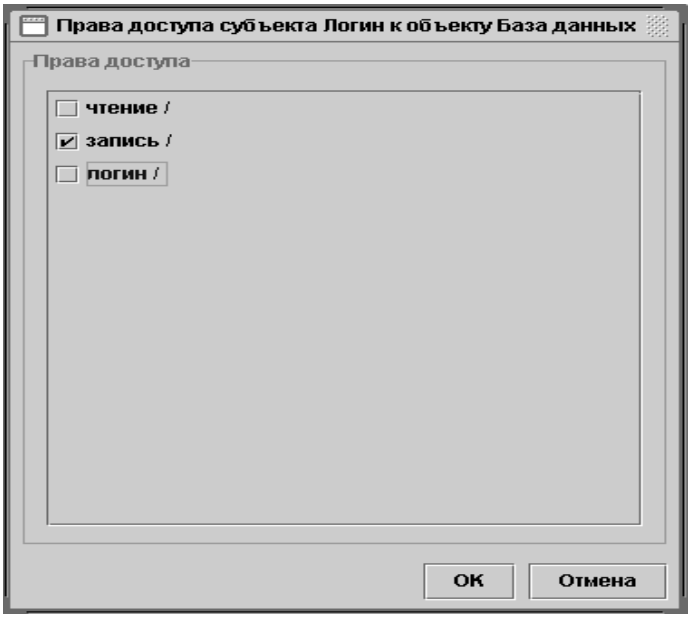


Рисунок 5.5 — Окно редактирования перечня прав доступа

Задание команд, необходимых для изменения состояния системы, выполняется из контекстного меню элемента «Множество команд». Для добавления новой команды нажмите на пункт контекстного меню «Новый объект Команда». Удаление выполняется путем нажатия на пункт «Удалить» в контекстном меню соответствующей команды. Редактирование команды производится в окне, представленном на рисунке 5.6.

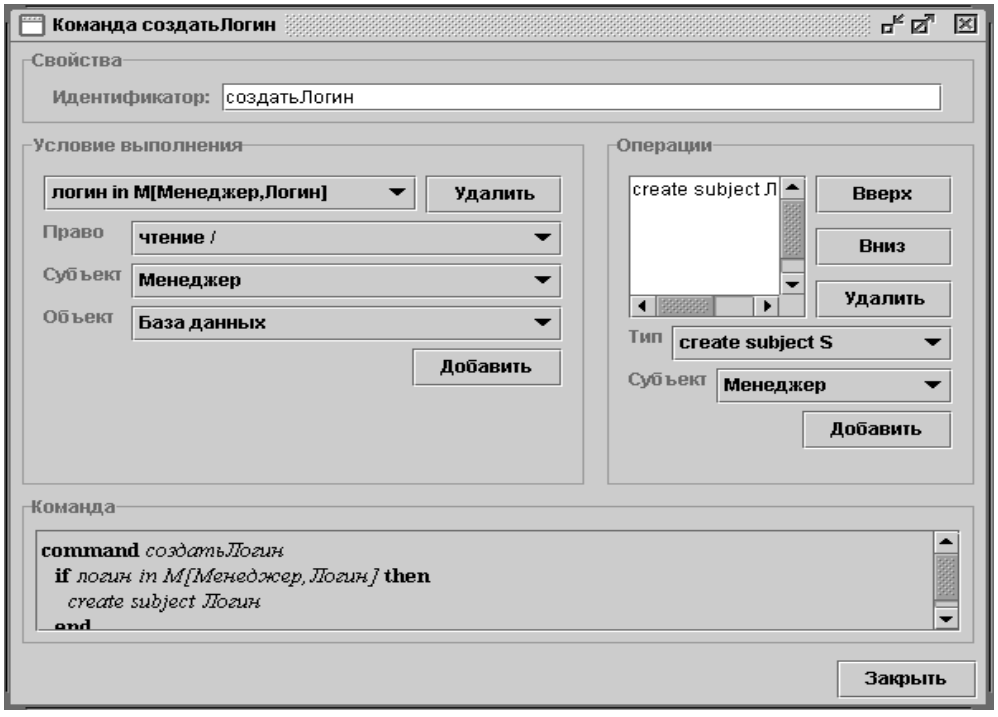


Рисунок 5.6 — Окно редактирования команды

Окно редактирования команды содержит 4 группы управляющих элементов:

- свойства;
- условие выполнения;
- операции;
- команда.

В группе свойства определяется идентификатор команды.

Группа «Условие выполнения» предназначена для задания перечня условий, которые должны быть истинными для выполнения перечня операций, указанных в группе «Операции».

В группе «Команда» отображается формулировка редактируемой команды.

Исследование политики безопасности выполняется в процессе моделирования. Пользователю предоставляется возможность выполнять команды политики безопасности двумя способами:

- вызов команды из серверного приложения;
- вызов команды из метода процесса.

Для вызова команды из серверного приложения требуется выполнить команду «Выполнить» в контекстном меню соответствующей команды. Вызов команды из метода процесса выполняется следующей функцией:

```
void execCommand(String command)
```

Например, в методе createLogin определен вызов команды «создатьЛогин» (см. рисунок 5.7).

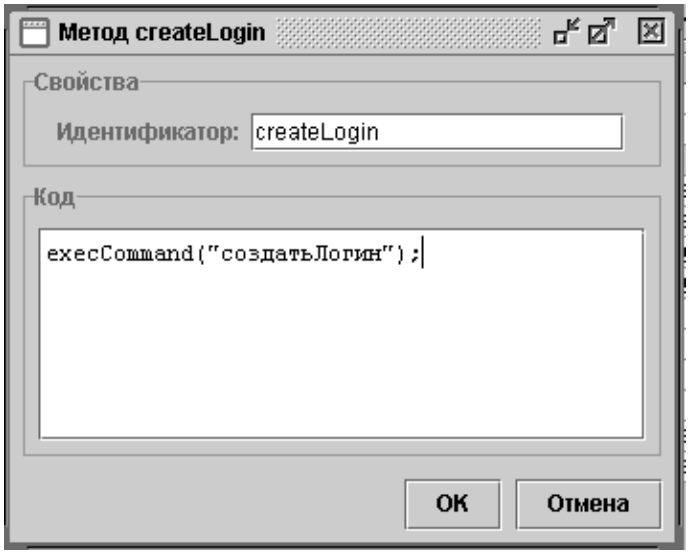


Рисунок 5.7 — Окно редактирования метода createLogin

Контроль соблюдения политики безопасности осуществляется модулем в сервере безопасности. Сообщения о попытках недопустимых операций заносятся в журнал сообщений моделирования (см. рисунок 5.8).

При включении политики безопасности в обмен сообщениями между процессами вносится дополнительная характеристика: запрашиваемое право доступа. Функция отправки сообщения через выходной узел при использовании политики безопасности имеет следующий вид:

```
void send(String node, Object data,String right)
```

Пример отправки сообщения в узел out:

```
send("out", "сообщение", "чтение");
```

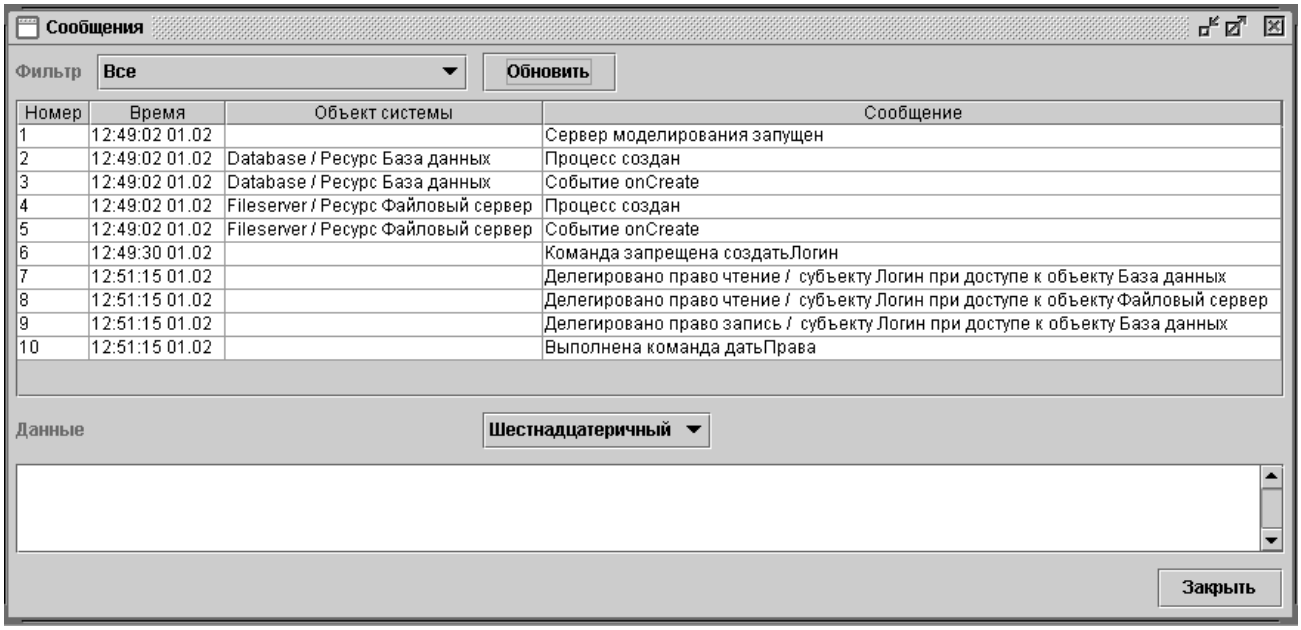


Рисунок 5.8 — Сообщения сервера, связанные с контролем политики безопасности

Получение сообщения получается с помощью функции recv, а для получения запрашиваемого права доступа перед вызовом функции recv требуется вызвать функцию recvRight:

```
String recvRight(String node)
```

Функция recvRight возвращает идентификатор права доступа. Пример получения сообщения из узла in:

```
String right=recvRight("in");
```

```
String data=(String)recv("in");
```

Взаимодействия процессов путем обмена сообщениями контролируется сервером и в случае запроса недопустимых прав доступа событие регистрируется в журнале событий моделирования.

5.3 Выполнение работы

1. Создать статическую модель, изображенную на рисунке 5.9. Для изучения используется система, состоящая из двух ресурсов: База данных, Файловый сервер и двух субъектов: Менеджер, Сеанс. Субъект сеанс представляет собой подключение менеджера к файловому серверу.

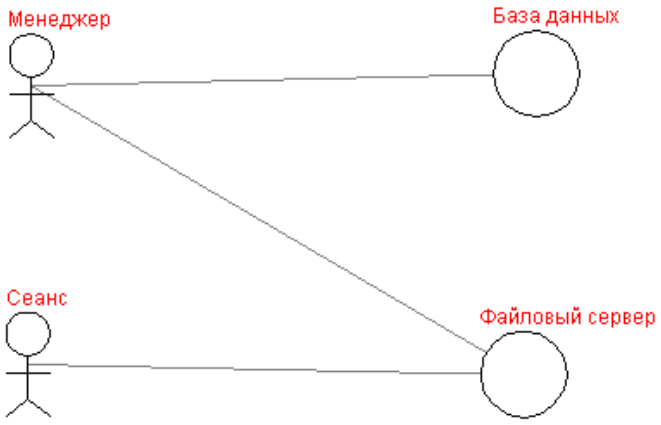


Рисунок 5.9 — Статическая модель для изучения политики безопасности

2. В диалоговом окне активизации политик безопасности включите дискреционную модель Харрисона-Руззо-Ульмана.
3. Задать множество прав доступа в соответствии с таблицей 5.1.

Таблица 5.1 — Права доступа

Право доступа	Описание
Чтение	Право на чтение данных из ресурсов
Запись	Право на запись данных в ресурсы
Создать	Право на создание сеанса

4. Задать матрицу прав доступа в соответствии с вариантом задания (см. таблицу 5.2).

Таблица 5.2 — Варианты задания по определению матрицы прав доступа

Вариант	**Отношение доступа**	**Права доступа**
1	Менеджер – База данных	чтение
	Менеджер – Файловый сервер	чтение
	Менеджер – Сеанс	создать
	Сеанс – Файловый сервер	чтение
2	Менеджер – База данных	чтение, запись
	Менеджер – Файловый сервер	—
	Менеджер – Сеанс	создать
	Сеанс – Файловый сервер	чтение, запись
3	Менеджер – База данных	чтение, запись
	Менеджер – Файловый сервер	запись
	Менеджер – Сеанс	—
	Сеанс – Файловый сервер	—
4	Менеджер – База данных	запись
	Менеджер – Файловый сервер	чтение
	Менеджер – Сеанс	—
	Сеанс – Файловый сервер	запись

5. Определить модель взаимодействия процессов идентичную модели на рисунке 5.10.

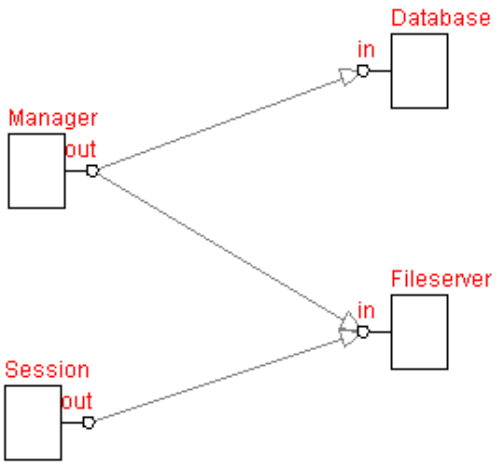


Рисунок 5.10— Модель взаимодействия процессов

Характеристики процессов определены в таблице 5.3.

Таблица 5.3 — Характеристики процессов динамической модели

Процесс	**Элемент статической модели**	**Создавать при запуске**
Manager	Менеджер	
Database	База данных	Да
Fileserver	Файловый сервер	Да
Session	Сеанс	

6. У процессов, представляющих субъекты (Manager, Session) добавить методы checkRead и checkWrite, которые выполняют отправку данных ресурсам с правами доступа чтение и запись.
7. Запустить сервер моделирования и клиенты моделирования для процессов Менеджер и Сеанс.
8. Проконтролировать и зафиксировать в отчете разрешенные и запрещаемые права доступа путем запуска методов checkRead и checkWrite при доступе от субъектов к ресурсам.
9. В множестве команд политики безопасности добавить команды в соответствии с таблицей вариантов 5.4.

Таблица 5.3 — Варианты задания по определению команд

Вариант	**Идентификатор команды**	**Определение команды**
1,2,3,4	создатьСеанс	command создатьСеанс if создать in M[Менеджер,Сеанс] then create subject Сеанс end
	удалитьСеанс	command удалитьСеанс if создать in M[Менеджер,Сеанс] then destroy subject Сеанс end
1	измПрава	command измПрава enter запись into M[Менеджер,Файловый сервер], delete создать from M[Менеджер,Сеанс], enter запись into M[Сеанс,Файловый сервер] end

Продолжение таблицы 5.4

2	измПрава	command измПрава enter запись into M[Менеджер, База данных], delete создать from M[Менеджер,Сеанс], delete чтение from M[Менеджер,База данных] end
4	измПрава	command измПрава delete создать from M[Менеджер,Сеанс], enter чтение into M[Сеанс,Файловый сервер], delete запись from M[Менеджер,База данных] end

10. Выполнить последовательно команды создатьСеанс и удалитьСеанс, проконтролировать результат их выполнения по сообщения моделирования.
11. Изменить состояние системы, выполнив команду измПрава. Проконтролировать изменение состояния по матрице прав доступа.
12. Повторно выполнить команду создатьСеанс и проконтролировать результат её выполнения.
13. Из клиентов моделирования процессов Менеджер и Сеанс запустить методы checkRead и checkWrite и зафиксировать в отчете разрешенные и запрещенные права доступа.

5.4 Содержание отчета

Отчет выполняется один на бригаду и должен включать:

1. Наименование и цель работы.
2. Краткие теоретические сведения.
3. Статическую модель согласно варианта.
4. Окна редактирования ресурсов, субъектов, угроз, уязвимостей, средств защиты.
5. Анализ защищенности
6. Выводы.

5.5 Контрольные вопросы кглаве 5

1. Какие существуют модели политик безопасности?

2. В чем различие модели Белла-ЛаПадулы и модели Харрисона-Руззо-Ульмана?

6 ЛАБОРАТОРНАЯ РАБОТА № 6

КОДИРОВАНИЕ ИНФОРМАЦИИ

6.1 Цель работы

Закрепить знания в построении кодов ОНК и помехоустойчивых кодов, в обнаружении и исправлении ошибок.

6.2 Темы лабораторной работы № 6

1. Построить код ОНК по методике Хаффмена.
2. Построить код ОНК по методике Шеннона-Фано.
3. Построить помехоустойчивый линейный систематический код.
4. Построить помехоустойчивый код Хэмминга.
5. Построить циклический код.
6. Исправление ошибок в линейном систематическом коде.
7. Исправление ошибок в коде Хэмминга.
8. Исправление ошибок в циклическом коде.

6.3 Основные определения

Оптимальным кодированием называется процедура преобразования символов первичного алфавита m_1 в кодовые слова во вторичном алфавите m_2, при котором средняя длина сообщений во вторичном алфавите имеет минимально возможную для данного m_2 длину.

Оптимальными называются коды, представляющие кодируемые понятия кодовыми словами минимальной средней длины.

Коды, представляющие первичные алфавиты с неравномерным распределением символов, имеющие среднюю минимальную длину кодового слова во вторичном алфавите, называются *оптимальными неравномерными кодами* (ОНК).

Эффективность ОНК оценивают при помощи *коэффициента статического сжатия*

$$K_{c.c} = \frac{H_{\max}}{l_{cp}} = \frac{\log_2 N}{\log_2 m \sum_{i=1}^{N} p_i l_i},$$

который характеризует уменьшение количества двоичных знаков на символ сообщения при применении ОНК по сравнению с применением

методов нестатистического кодирования, и *коэффициента относительной эффективности*

$$K_{о.э} = \frac{H}{l_{ср}} = \frac{-\sum_{i=1}^{N} p_i \log_2 p_i}{\log_2 m \sum_{i=1}^{N} l_i p_i},$$

который показывает, насколько используется статическая избыточность передаваемого сообщения.

Из свойств оптимальных кодов вытекают принципы их построения:

- *Первый принцип* оптимального кодирования: выбор каждого кодового слова необходимо производить так, чтобы содержащееся в нем количество информации было максимальным.
- *Второй принцип* оптимального кодирования заключается в том, что буквам первичного алфавита, имеющим большую вероятность, присваиваются более короткие кодовые слова во вторичном алфавите.

Принципы оптимального кодирования определяют методики построения оптимальных кодов.

Для защиты полезной информации от помех необходимо в том или ином виде вводить избыточность: увеличивать число символов и время их передачи, повторять целые сообщения, повышать мощность сигнала - все это ведет к усложнению и удорожанию аппаратуры.

Коды без избыточности обнаружить, а тем более исправлять ошибки не могут. Количество символов, в которых любые две комбинации кода отличаются друг от друга, называется *кодовым расстоянием*. Минимальное количество символов, в которых все комбинации кода отличаются друг от друга, называется *минимальным кодовым расстоянием.* Минимальное кодовое расстояние - параметр, определяющий помехоустойчивость кода и заложенную в коде избыточность. Минимальным кодовым расстоянием определяются корректирующие свойства кода.

В общем случае для обнаружения t ошибок минимальное кодовое расстояние

$$d_0 = t + 1.$$

Минимальное кодовое расстояние, необходимое для одновременного обнаружения и исправления ошибок,

$$d_0 = t + \sigma + 1 ,$$

где σ - число исправляемых ошибок.

Для кодов, только исправляющих ошибки,

$$d_0 = 2\sigma + 1.$$

Для того чтобы определить кодовое расстояние между двумя комбинациями двоичного кода, достаточно просуммировать эти комбинации по модулю 2 и подсчитать число единиц полученной комбинации.

Для обнаружения и исправления одиночной ошибки соотношение между числом информационных разрядов k и числом корректирующих разрядов ρ должно удовлетворять следующим условиям:

$$2^{\rho} \geq n+1, \qquad 2^{k} \leq \frac{2^{n}}{n+1},$$

при этом подразумевается, что общая длина кодовой комбинации

$$n = k + t$$

Для практических расчетов при определении числа контрольных разрядов кодов с минимальным кодовым расстоянием $d_0 = 3$ удобно пользоваться выражениями

$$t_{1(2)} = [\log_2 (n + 1)],$$

если известна длина полной кодовой комбинации n, и

$$t_{1(2)} = [\rho \log_2 \{ (k + 1) + [log_2 (k + 1)] \}],$$

если при расчетах удобнее исходить из заданного числа информационных символов k .

Для кодов, обнаруживающих все трехкратные ошибки ($d_0=4$),

$$t_{1(3)} \geq 1 + \log_2 (n + 1),$$

или

$$t_{1(3)} \geq 1 + \log_2 [(n + 1) + \log_2 (k + 1)]$$

Для кодов длиной в n символов, исправляющих одну или две ошибки ($d_0=5$),

$$t_2 \geq log_2 (C_n^2 + C_n^1 + 1).$$

Для практических расчетов можно пользоваться выражением:

$$t_2 = [\log_2 \frac{n^2 + n + 1}{2}].$$

Для кодов, исправляющих 3 ошибки ($d_0=7$),

$$t_3 = [\log_2 \frac{n^3 + n^2 + n + 1}{6}].$$

6.4 Теоретические сведения

6.4.1 Построение ОНК по методике Шеннона-Фано

Построение оптимального кода по методу Шеннона-Фано для ансамбля из *M* сообщений сводится к следующей процедуре:

1) множество из *M* сообщений располагают в порядке убывания вероятностей;
2) первоначальный ансамбль кодируемых сигналов разбивают на две группы таким образом, чтобы суммарные вероятности сообщений обеих групп были по возможности равны;
3) первой группе присваивают символ 0, второй группе символ 1;
4) каждую из подгрупп делят на две группы так, чтобы их суммарные вероятности были по возможности равны;
5) первым подгруппам каждой из групп вновь присваивают 0, а вторым - 1, в результате чего получают вторые цифры кода. Затем каждую из четырех подгрупп вновь делят на равные (с точки зрения суммарной вероятности) части и т. д. До тех пор, пока в каждой из подгрупп останется по одной букве.

Рассмотрим несколько конкретных примеров построения оптимальных кодов.

Пример: Построим оптимальный код сообщения, состоющего из восьми равновероятных букв.

Решение. Так вероятности данного ансамбля сообщений равны p1 = p2 = ... = = p8 = 2-3 и порядок их расположения не играет роли, то расположим их так, как показано в табл. 1. Затем разбиваем данное

множество на две равновероятные группы. Первой группе в качестве первого символа кодовых слов присваиваем 0, а второй - 1. Во второй колонке табл. 1 записываем четыре нуля и четыре единицы. После чего разбиваем каждую из групп еще на две равновероятные подгруппы. Затем каждой первой подгруппе присваиваем 0, а второй - 1 и записываем в третью колонку табл. 1. Далее каждую из четырех подгрупп разбиваем на две равновероятные части и первой из них присваиваем 0, а второй - 1. Таким образом в четвертой колонке табл. 1 появится значение третьего символа кодовых слов.

Таблица 6.1 – Оптимальный код равновероятных букв.

Буква	**Кодовое слово полученное после разбиения**		
	первого	Второго	третьего
A_1	0	0	0
A_2	0	0	1
A_3	0	1	0
A_4	0	1	1
A_5	1	0	0
A_6	1	0	1
A_7	1	1	0
A_8	1	1	1

Проверка оптимальности кода осуществляется путем сравнения энтропии кодируемого (первичного) алфавита со средней длиной кодового слова во вторичном алфавите.

Для рассматриваемого примера энтропия источника сообщений

$$H = log_2 N = log_2 8 = 3 \text{ бит/символ}$$

а среднее число двоичных знаков на букву кода

$$L = \sum_{i}^{N} l_i p_i = 0.125 \cdot 3 \cdot 8 = 3$$

где l_i - длина i-ой кодовой комбинации; p_i - вероятность появления i-го символа комбинации длиной в l_i.

Таким образом, *H=L,* т. е. код является оптимальным для данного ансамбля сообщений.

Вывод: *Для ансамблей равновероятных сообщений оптимальным является равновероятный код. Если число исходных элементов ансамбля равно целой степени двух, то всегда H=L.*

Пример: Первичный алфавит состоит из 8 символов со следующими вероятностями появления: $a_1=0,5;\ a_2=0,25;\ a_3=0,098;\ a_4=0,052;\ a_5=0,04;\ a_6=0,03;\ a_7=0,019;\ a_8=0,011$. Построить ОНК по методу Шеннона - Фано и подсчитать коэффициенты.

Решение.

Таблица 6.2 – Оптимальный код неравновероятных букв.

a_i	p_i	**Кодовое слово после разбиения**						**Знаков**	$l_i p_i$
		1	2	3	4	5	6		
1	0,5	0	-	-	-	-	-	1	0,5
2	0,25	1	0	-	-	-	-	2	0,5
3	0,098	1	1	0	0	-	-	4	0,392
4	0,052	1	1	0	1	-	-	4	0,208
5	0,04	1	1	1	0	-	-	4	0,16
6	0,03	1	1	1	1	0	-	5	0,15
7	0,019	1	1	1	1	1	0	6	0,114
8	0,011	1	1	1	1	1	1	6	0,066

$$H_{\max} = \log_2 m_1 = \log_2 8 = 3 \ \ бит/символ$$

$$H = -\sum_{i=1}^{8} p_i \log_2 p_i = 2,07 \ \ бит/символ$$

$$l_{ср} = \sum_{i=1}^{8} l_i p_i = 2,09 \ \ бит/символ$$

$$K_{с.с} = \frac{H_{\max}}{l_{ср}} = \frac{3}{2,09} = 1,43 \qquad K_{о.э} = \frac{H}{l_{ср}} = \frac{2,07}{2,09} = 0,98$$

6.4.2 Построение ОНК по методике Хаффмена

Хаффмен предложил следующий метод построения ОНК:

1) Символы первичного алфавита выписываются в порядке убывания вероятностей.
2) Последние n_0 символов, где $2 \le n_0 \le m$ и $N - n_0/m - 1$ - целое число, объединяют в некоторый новый символ с вероятностью, равной сумме вероятностей объединяемых символов.
3) Последние символы с учетом образованного символа вновь объединяют и получают новый, вспомогательный, символ.
4) Опять выписывают символы в порядке убывания вероятностей с учетом вспомогательного символа и т. д. До тех пор, пока вероятности m оставшихся символов после $N - n_0/m - 1$-го выписывания не дадут в сумме 1.

На практике обычно не производят многократного выписывания вероятностей символов, а обходятся элементарными геометрическими построениями, суть которых для кодов с числом качественных признаков $m = 2$ сводится к тому, что символы кодируемого алфавита попарно объединяются в новые символы, начиная с символов, имеющих наименьшую вероятность, а затем, с учетом вероятностей вновь образованных символов, опять производят попарное объединение символов с наименьшими вероятностями и таким образом строят двоичное кодовое дерево, в вершине которого стоит символ с вероятностью 1.

Пример: Построить методом Хаффмена оптимальный код для алфавита со следующим распределением вероятностей появления букв в тексте: *A = 0,5; B = 0,15; C = 0,12; D = 0,1; E = 0,04; F = 0,04; G = 0,03; H = 0,02.*

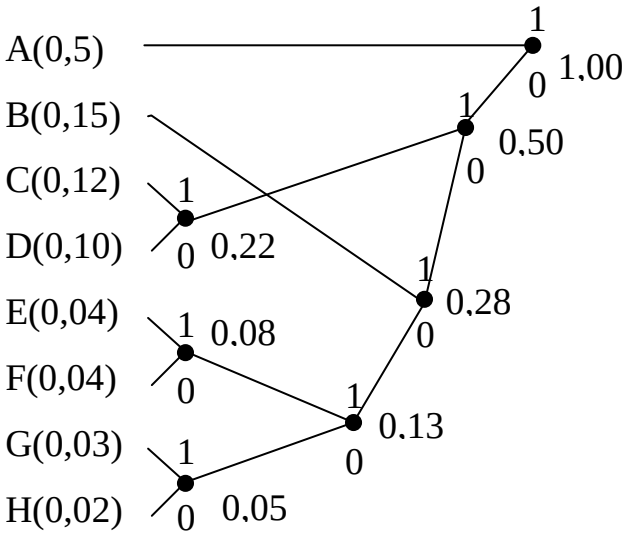


Рисунок 6.1 – Построение кода Хаффмена.

Решение. Сначала находят буквы с наименьшими вероятностями 0,02 (*H)* и 0,03 (*G),* затем проводят от них линию к точке, в которой вероятность появления буквы *G* равна 0,05. Затем берут две наименьшие вероятности 0,04 (*F*) и 0,04 (*E*) и получают новую точку с вероятностью 0,08. Теперь наименьшими вероятностями обладают точки, соответствующие вспомогательным символам с вероятностями 0,05 и 0,08. Соединяем их линией с новой точкой, соответствующей вспомогательному символу с

вероятностью 0,13. Продолжаем в том же духе до тех пор, пока линия от основных и вспомогательных символов не сольются в точке, дающую суммарную вероятность, равную 1. Обозначим каждую верхнюю линию (см. рисунок) цифрой 1, а нижнюю - цифрой 0. Получим ОНК, который для каждого слова представляет собой последовательность нулей и единиц, которые встречаются по пути к данному слову от конечной точки (1,00). Полученные коды представлены в таблице 6.3

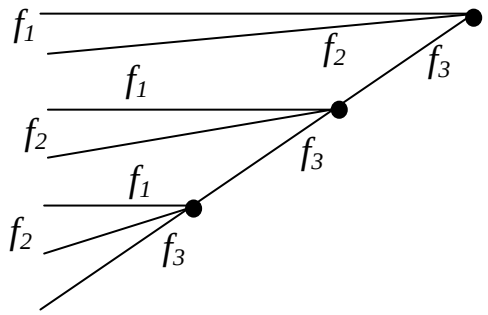
Таблица 6.3 – Двоичный код Хаффмена для 8-и сообщений

Буква	Вероятность	ОНК	Число знаков в кодовом слове	$p_i l_i$
A	0,50	1	1	0,50
B	0,15	0 0 1	3	0,45
C	0,12	0 1 1	3	0,36
D	0,10	0 1 0	3	0,30
E	0,04	0 0 0 1 1	5	0,20
F	0,04	0 0 0 1 0	5	0,20
G	0,03	0 0 0 0 1	5	0,15
H	0,02	0 0 0 0 0	5	0,10

Пример: Построить код Хаффмена для передачи сообщений при помощи трех частот f_1, f_2, f_3, если символы первичного алфавита встречаются в сообщениях со следующими вероятностями: *А = 0,24; Б = 0,18; В = 0,38; Г = 0,1; Д = 0,06; Е = 0,02; Ж = 0,02.*

Решение

Таблица 6.4 – Троичный код Хаффмена для 7-и сообщений

Буква	Вероятность	Дерево	Код
В	0,38		f_1
А	0,24		f_2
Б	0,18		f_3f_1
Г	0,1		f_3f_2
Д	0,06		$f_3f_3f_1$
Е	0,02		$f_3f_3f_2$
Ж	0,02		$f_3f_3f_3$

Полученный код легко декодируется, так как ни один код не начинается с f_1 и f_2, кроме одного одноразрядного кода

6.4.3 Построение линейного систематического кода

Для этого используется образующая матрица $P_{n,k}$, состящая из двух матриц U_k – информационной, квадратной k×k, и H_ρ – проверочной, k×ρ:

$$P_{n,k} = \left\| \begin{matrix} a_{11} & a_{12} & \dots & a_{1k} \\ a_{21} & a_{22} & \dots & a_{2k} \\ \dots & \dots & \dots & \dots \\ a_{k1} & a_{k2} & \dots & a_{kk} \end{matrix} \right\| \left\| \begin{matrix} b_{11} & b_{12} & \dots & b_{1p} \\ b_{21} & b_{22} & \dots & b_{2p} \\ \dots & \dots & \dots & \dots \\ b_{k1} & b_{k2} & \dots & b_{kp} \end{matrix} \right\|$$

$$U_k \qquad\qquad H_\rho$$

Рисунок 6.2 – Производящая матрица

Для построения производящей матрицы удобно U_k брать в виде квадратной единичной:

$$U_k = \left\| \begin{matrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 \end{matrix} \right\|$$

Рисунок 6.3 – Информационная матрица

Тогда матрица $P_{n,k}$ в канонической форме:

$$P_{n,k} = \left\| \begin{matrix} 1 & 0 & \dots & 0 & b_{11} & b_{12} & \dots & b_{1p} \\ 0 & 1 & \dots & 0 & b_{21} & b_{22} & \dots & b_{2p} \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & 1 & b_{k1} & b_{k2} & \dots & b_{kp} \end{matrix} \right\|$$

Рисунок 6.4 – Проверочная матрица

При этом проверочная матрица H_ρ строится так:

- количество единиц в строке должно быть $\geq (d_{min} - 1)$;
- $\oplus$ двух любых строк должна содержать $\geq (d_{min} - 2)$ единиц.

Проверочные символы образуются линейными операциями над информационными символами.

Для каждой кодовой комбинации нужно составить ρ независимых сумм по mod 2.

Для этого удобно использовать проверочную матрицу H. Сначала строят подматрицу H’ – транспонированную матрицу H_ρ:

$$H' = \begin{Vmatrix} b_{11} & b_{21} & \dots & b_{k1} \\ b_{12} & b_{22} & \dots & b_{k2} \\ \dots & \dots & \dots & \dots \\ b_{1p} & b_{2p} & \dots & b_{kp} \end{Vmatrix}$$

Рисунок 6.5 – Транспонированная матрица H’

Затем справа приписывается единичная матрица:

$$H = \begin{Vmatrix} b_{11} & b_{21} & \dots & b_{k1} & 1 & 0 & \dots & 0 \\ b_{12} & b_{22} & \dots & b_{k2} & 0 & 1 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ b_{1p} & b_{2p} & \dots & b_{kp} & 0 & 0 & \dots & 1 \end{Vmatrix}$$

Рисунок 6.6 – Матрица H

Проверочные символы по матрице H находятся так:

- позиции, занимаемые единицами в ***первой*** строке подматрицы H’, определяют информационные разряды, участвующие в формировании ***первого*** проверочного разряда кодовой комбинации;

- позиции, занимаемые единицами во ***второй*** строке подматрицы H’, определяют информационные разряды, участвующие в формировании ***первого*** проверочного разряда кодовой комбинации и т.д.….

Пример: построить линейный систематический код для информационной комбинации 0011 (k = 4) по производящей матрице $P_{7,4}$:

$$P_{7,4} = \begin{Vmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{Vmatrix}.$$

Рисунок 6.7 – Образующая матрица $P_{7,4}$

Т.е. проверочная матрица:

$$H_3 = \begin{Vmatrix} 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{Vmatrix}.$$

Рисунок 6.8 – Проверочная матрица H_3

Тогда транспонированная матрица:

$$H' = \begin{Vmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \end{Vmatrix}$$

Рисунок 6.9 – Транспонированная матрица H’

Приписываем справа единичную матрицу:

$$H = \begin{array}{c} a_1\ a_2\ a_3\ a_4\ \ b_1\ b_2\ b_3 \\ \begin{Vmatrix} 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{Vmatrix} \end{array}.$$

Рисунок 6.10 – Матрица H

Кодовая комбинация должна содержать ρ проверочных символов:

$\rho = n - k = 7 - 4 = 3.$

Нужно найти три проверочных символа: b_1, b_2, b_3.

Из подматрицы H’ определяем:

$b_1 = a_2 \oplus a_3 \oplus a_4$

$b_2 = a_1 \oplus a_3 \oplus a_4$
$b_3 = a_1 \oplus a_2 \oplus a_4$

Тогда для нашего сообщения 0011 проверочные символы будут:
$b_1 = 0 \oplus 1 \oplus 1 = 0$
$b_2 = 0 \oplus 1 \oplus 1 = 0$
$b_3 = 0 \oplus 0 \oplus 1 = 1$

$a_1\ a_2\ a_3\ a_4\ \ b_1\ b_2\ b_3$

Полная кодовая комбинация будет: 0 0 1 1 0 0 1 .

6.4.4 Исправление ошибок в линейном систематическом коде

Производится по проверочной матрице H (см.рисунок 4.10): производится суммирование по mod 2 проверочных символов кодовой комбинации и проверочных символов вычисленных по принятым информационным:

$S_1 = b_1 \oplus a_2 \oplus a_3 \oplus a_4$
$S_2 = b_2 \oplus a_1 \oplus a_3 \oplus a_4$
$S_3 = b_3 \oplus a_1 \oplus a_2 \oplus a_4$

В результате ρ таких проверок получаем ρ – разрядное двоичное число – ***синдром***, которое будет равно 0, при отсутствии ошибок.

Если ошибка в первом разряде a_1 (он используется при образовании S_2 и S_3), то получим $S_1 = 0$, $S_2 = 1$, $S_3 = 1$, т.е. синдром 011. При ошибке в a_2 синдром 101, и т.д.

6.4.5 Построение кода Хэмминга

Код строится таким образом, чтобы в результате $\rho = n - k$ проверок получить ρ – разрядное двоичное число, указывающее номер ошибочног разряда в кодовой комбинации.

Для этого проверочные символы должны находиться на позициях в кодовой комбинации, номера которых равны целой степени двойки, т.е. 2^0, 2^1, 2^2,…, $2^{\rho-1}$, т.к. каждый из них входит только в одно из проверочных уравнений. Т.о. проверочные символы должны находиться на 1 – й, 2 – й, 4 – й … позициях (нумерация слева-направо).

Результат первой проверки дает цифру младшего разряда синдрома в двоичной записи. Если он равен 1, то один из проверенных символов искажен.

Таким образом, в первой проверке должны участвовать символы, номера которых содержат единицы в двоичной записи в первом разряде: $001_2=1_{10}$, $011_2=3_{10}$, $101_2=5_{10}$, $111_2=7_{10}$, …

Вторая проверка дает цифру второго разряда синдрома т.е, во второй проверке должны участвовать символы, номера которых содержат единицу во втором разряде: $010_2=2_{10}$, $011_2=3_{10}$, $110_2=6_{10}$, $111_2=7_{10}$,… и т.д.

Т.о., синдром должен быть:

$S_1 = b_1 \oplus a_3 \oplus a_5 \oplus a_7$

$S_2 = b_2 \oplus a_3 \oplus a_6 \oplus a_7$

$S_3 = b_3 \oplus a_5 \oplus a_6 \oplus a_7$

Пример: Построить код Хэмминга 9,5 для информационной комбинации 10011. Т.к. информационные символы должны быть на третьей, пятой, шестой, седьмой, девятой позициях, то:

a_3	a_5	a_6	a_7	a_9
1	0	0	1	1 .

Находим проверочные символы:

$a_1 = b_1 = a_3 \oplus a_5 \oplus a_7 \oplus a_9 = 1 \oplus 0 \oplus 1 \oplus 1 = 1$

$a_2 = b_2 = a_3 \oplus a_6 \oplus a_7 = 1 \oplus 0 \oplus 1 = 0$

$a_4 = b_3 = a_5 \oplus a_6 \oplus a_7 = 0 \oplus 0 \oplus 1 = 1$

$a_8 = b_4 = a_9 = 1$

Т.о. код Хэмминга будет:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
1	0	1	1	0	0	1	1	1
b_1	b_2		b_3				b_4	

6.4.6 Исправление ошибок в коде Хэмминга

Пусть получена кодовая комбинация в коде Хэмминга: 101110111. Определить есть ли ошибка, и при наличии ее исправить.

Считаем синдром для принятой комбинации:

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_9
1	0	1	1	1	0	1	1	1

$S_1 = a_1 \oplus a_3 \oplus a_5 \oplus a_7 \oplus a_9 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$

$S_2 = a_2 \oplus a_3 \oplus a_6 \oplus a_7 = 0 \oplus 1 \oplus 0 \oplus 1 = 0$

$S_3 = a_4 \oplus a_5 \oplus a_6 \oplus a_7 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$

$S_4 = a_8 \oplus a_9 = 1 \oplus 1 = 0$

Получаем: $0101_2 = 5_{10}$. Т.е.ошибка в пятом разряде.

Исправляем ее: $a_1\ a_2\ a_3\ a_4\ a_5\ a_6\ a_7\ a_8\ a_9$

1 0 1 1 ± 0 1 1 1

0

Получаем правильную комбинацию: 101100111

6.4.7 Построение циклического кода

Информационная кодовая комбинация G(x) степени (k-1) умножается на одночлен x^{n-k} и прибавляется R(x) - остаток от деления данного произведения на образующий многочлен P(x) степени ρ:

$$F(x) = x^{n-k} \times G(x) \oplus R(x)$$

Т.о. первые k символов полученной кодовой комбинации будут совпадать с информационными символами.

Пример:

Пусть k = 12. Построить циклический код с помощью образующего многочлена пятой степени $P(x) = x^5 \oplus x^4 \oplus x^2 \oplus 1$.

Пусть $G(x) = x^{10} \oplus x^6 \oplus x^4 \oplus x^3 \oplus 1$, т.е. G(x) = 010001011001. Найти F(x).

Решение

Находим произведение $x^{n-k} \times G(x) = x^5 \times G(x) = x^5 \times (x^{10} \oplus x^6 \oplus x^4 \oplus \oplus x^3 \oplus 1) = x^{15} \oplus x^{11} \oplus x^9 \oplus x^8 \oplus x^5$

Получаем частное $\dfrac{G(x) \times x^{n-k}}{P(x)} = \dfrac{x^{15} \oplus x^{11} \oplus x^9 \oplus x^8 \oplus x^5}{x^5 \oplus x^4 \oplus x^2 \oplus 1}$

Остаток будет: $R(x) = x^4 \oplus x^2 \oplus 1$.

Т.о. кодовая комбинация будет: $x^{15} \oplus x^{11} \oplus x^9 \oplus x^8 \oplus x^5 \oplus x^4 \oplus x^2 \oplus 1 =$

$=$ 010001011001 10101

---------- -------

k ρ

6.4.8 Исправление ошибок в циклическом коде

Производится по остаткам от деления принятой комбинации на образующий многочлен P(x): если принятая комбинация делится на P(x) без остатка, то код принят без ошибок. При наличии остатка определяют его вес ω.

Если ω ≤ σ (число исправляемых ошибок), то F(x) ⊕ остаток = правильная кодовая комбинация.

Если ω > σ, то циклически сдвигаем полученную кодовую комбинацию влево на один разряд. Полученную комбинацию делим на образующий многочлен P(x). Определяем вес остатка ω.

Если ω ≤ σ, то сдвинутая комбинация ⊕ остаток. Сдвигаем вправо циклически на один разряд и получаем правильную кодовую комбинацию.

Если ω > σ, то циклически сдвигаем сдвинутую кодовую комбинацию влево на один разряд. Полученную комбинацию делим на образующий многочлен P(x). Определяем вес остатка ω.

И так до тех пор пока не подучим ω ≤ σ.

Пример: F(x) = 1000110. $P(x) = x^3 \oplus x \oplus 1$.

Исправить ошибку при ее наличии. σ = 1.

Решение

1.

1000110	1011
1011	1011
1111	
1011	
1000	
1011	
11	

ω =2. ω > σ.

Сдвигаем F(x) влево на один разряд: 0001101

2.

0001101	1011
1011	1
0110	

ω =2. ω > σ.

Сдвигаем F(x) влево на один разряд: 0011010

3.

0011010	1011
1011	1011
1100	
1011	
111	

$\omega = 3$. $\omega > \sigma$.

Сдвигаем F(x) влево на один разряд: 0011010

4.

0110100	1011
1011	1011
1100	
1011	
1110	
1011	
101	

$\omega = 2$. $\omega > \sigma$.
Сдвигаем F(x) влево на один разряд: 1101000

5.

1101000	1011
1011	1011
1100	
1011	
1110	
1011	
1010	
1011	
1	

$\omega = 1$. $\omega = \sigma$.
Суммируем по модулю два: $1101000 \oplus 1 = 1101001$.

Циклически сдвигаем вправо на 4 разряда и получаем исправленную комбинацию: 1001110.

6.5 Выполнение работы

Каждая бригада получает номер задания для написания программы, реализующей конкретную задачу:

1 – я бригада выполняет задания по темам 1,5;

2 – я бригада выполняет задания по темам 2,6;

3 – я бригада выполняет задания по темам 3,7;

4 – я бригада выполняет задания по темам 4,8;

Инструментарий не ограничивается.

Программа должна предусматривать ввод информационных символов, ввод образующей матрицы или многочлена, выдачу результатов.

6.6 Содержание отчета

1. Наименование и цель работы
2. Краткие теоретические сведения
3. Описание программы
4. Схема алгоритма
5. Текст программы
6. Контрольный пример
7. Выводы

6.7 Контрольные вопросы к главе 6

1. Что такое коды ОНК?
2. В чем различие методик Шеннона-Фано и Хаффмена?
3. Основные шаги построения помехоустойчивых кодов.
4. Какие этапы нахождения ошибок в линейном систематическом коде, коде Хэмминга, циклическом коде.

РЕКОМЕНДОВАННАЯ ЛИТЕРАТУРА

1. Домашев А. В. и др. Программирование алгоритмов защиты информации. – М.: Нолидж, 2000.

2. Зегжда Д. П. и Ивашко А. М.. Основы безопасности информационных систем. – М.: Горячая линия – Телеком, 2000.

3. Мельников В. В. Защита информации в компьютерных системах. – М.: Финансы и статистика, 1997.

4. Романец Ю.В. и др. Защита информации в компьютерных системах и сетях. – М.: Радио и связь, 1999.

5. Столингс В. Криптография и защита сетей. – М., СПб, К.: Вильямс, 2001.

6. Цымбал В.П. Теория информации и кодирование. – К.: Выща школа, 1997.

7. Ященко В. В. Введение в криптографию. – СПб.: Питер, 2001.