bhv

Михаил Масленников

Практическая криптография

+ CD-ROM

- Некоторые идеи мэтра криптографии Клода Шеннона
- Основные криптологические процедуры
- Построение качественного генератора гаммы
- Хэш-функция и электронная подпись

МАСТЕР РЕШЕНИЙ

Михаил Масленников

Практическая криптография

Санкт-Петербург

«БХВ-Петербург»

2003

УДК 681.3.06
ББК 32.81
М31

Масленников М. Е.

М31 Практическая криптография. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.

ISBN 5-94157-201-8

Книга российского криптографа посвящена прикладным проблемам современной криптографии. Наряду с основными теоретическими положениями рассматривается: создание криптографического ядра, встраивание криптографических алгоритмов в Microsoft Outlook и Lotus Notes, создание автоматизированной системы документооборота, технология отпечатков пальцев. Все программное обеспечение, описываемое в книге, создано в Borland C++ Builder. На прилагаемом к книге компакт-диске находятся демонстрационные версии некоторых программ и документация.

Для широкого круга IT-специалистов и специалистов, отвечающих за безопасность систем

УДК 681.3.06
ББК 32.81

Группа подготовки издания:

Главный редактор	*Екатерина Кондукова*
Зам. главного редактора	*Анатолий Адаменко*
Зав. редакцией	*Анна Кузьмина*
Редактор	*Петр Науменко*
Компьютерная верстка	*Натальи Караваевой*
Корректор	*Виктория Пиотровская*
Оформление серии	*Via Design*
Дизайн обложки	*Игоря Цырульникова*
Зав. производством	*Николай Тверских*

Лицензия ИД № 02429 от 24.07.00. Подписано в печать 25.11.02.
Формат 70×100 1/16. Печать офсетная. Усл. печ. л. 37,41.
Тираж 3000 экз. Заказ №
"БХВ-Петербург", 198005, Санкт-Петербург, Измайловский пр., 29.

Гигиеническое заключение на продукцию, товар № 77.99.02.953.Д.001537.03.02 от 13.03.2002 г. выдано Департаментом ГСЭН Минздрава России.

Отпечатано с готовых диапозитивов
в Академической типографии "Наука" РАН
199034, Санкт-Петербург, 9 линия, 12.

ISBN 5-94157-201-8

© Масленников М. Е., 2002
© Оформление, издательство "БХВ-Петербург", 2002

Содержание

Предисловие **1**

Введение **7**

Часть I. Основы криптографии **13**

Глава 1. История криптографии **15**

Глава 2. Стойкость шифра **23**

Глава 3. Генератор гаммы **31**

Глава 4. Ключевые системы **42**

Схемы организации шифрованной связи с традиционной ключевой системой 45

Системы связи с "открытым" ключом 48

Глава 5. Блочные шифры **54**

Глава 6. Электронная подпись **67**

Часть II. Криптографическое ядро и интерфейсная оболочка **77**

Глава 7. Что понимать под криптографическим ядром и интерфейсной оболочкой? **79**

Глава 8. Выработка секретного ключа **91**

Почему секретный ключ представлен десятичными цифрами? 96

Что такое пароль? 97

Что такое метка? 101

Глава 9. База данных открытых ключей **105**

Глава 10. Процедура шифрования **115**

Глава 11. Процедура подписи 123
Дата и время осуществления подписи 129

Часть III. Встраивание гарантированных алгоритмов в Microsoft Outlook 133

Глава 12. Основы MAPI 135
Компонент доступа к хранилищу сообщений 135
Команды, связанные с секретными ключами 141
Команды, связанные с шифрованием без использования системы с открытым распределением ключей 144
Команды, связанные с пересылкой сообщений в защищенном режиме 146
Команды, связанные с подписью 147

Глава 13. Шифрование и подпись объектов в хранилищах MAPI 151

Глава 14. Организация закрытой почты 166

Глава 15. Организация хранилища открытых ключей 182

Часть IV. Встраивание гарантированных алгоритмов в Lotus Notes 197

Глава 16. Notes API-приложения 199

Глава 17. Ключевая книга 209

Глава 18. База данных с открытыми ключами 217

Глава 19. Шифрование с использованием ключевой книги 227

Глава 20. Шифрование по списку 250
Выработка секретного ключа для шифрования по списку 251

Глава 21. Организация электронной подписи 263

Часть V. Автоматизированная система электронного документооборота 281

Глава 22. Кто принимает участие в электронном документообороте и кто его обслуживает 283

Глава 23. Электронный документ и протокол к нему 291
База данных документов DDB 297
База данных подписей SDB 298
База данных реестров RDB 300

Глава 24. Специализированные программные модули и конфигурационные файлы 302

Глава 25. Подготовка электронных документов 327

Глава 26. Пересылка электронных документов 336

Прямая доставка по Internet с использованием протоколов SMTP и POP3 337
Pegasus Mail 338
Sprint Mail 338
Альтернативная почта 338
Организация дозвона и минимизация времени доступа к хосту 347
Пароли для доступа к хосту и для приема почты 348
Особенности использования Pegasus Mail 348
Особенности использования Sprint Mail 351
Особенности использования альтернативной почты 355

Глава 27. Контроль за доставкой электронных документов 357

Глава 28. Главный менеджер системы 365

Глава 29. Менеджер системы безопасности 377

Глава 30. Взаимодействие участников документооборота с менеджерами 386

ЧАСТЬ VI. ТЕХНОЛОГИЯ ОТПЕЧАТКОВ ПАЛЬЦЕВ 393

Глава 31. Проблема носителя секретного ключа 395

Глава 32. Уникальный рисунок отпечатка пальца человека 398

Глава 33. Формирование секретного ключа по отпечатку пальца 406

Глава 34. Криптосервер 427

Глава 35. Генератор случайных паролей по отпечатку пальца 437

ЗАКЛЮЧЕНИЕ 443

ПРИЛОЖЕНИЕ. СОДЕРЖАНИЕ КОМПАКТ-ДИСКА 453

СПИСОК ЛИТЕРАТУРЫ 455

Internet-ресурсы 455

ПРЕДМЕТНЫЙ УКАЗАТЕЛЬ 457

Предисловие

Все взято в трубы, перекрыты краны,
Ночами только воют и скулят.
Что надо? Надо сыпать соль на раны.
Чтоб лучше помнить, пусть они болят.

Владимир Высоцкий. "Побег на рывок"

Россия, 1992 год. Переход к рынку. Динамика роста курса доллара:

01.07.92 1$ = 125 руб.

01.08.92 1$ = 161 руб. (рост за месяц почти на 29%)

01.09.92 1$ = 205 руб. (+27%)

01.10.92 1$ = 254 руб. (+24%)

01.11.92 1$ = 398 руб. (+57%)

01.12.92 1$ = 447 руб. (+12%)

Наш родной рубль в стремительном падении. И вдруг

02.12.92 1$ = 417 руб. (−7% за день!)

31.12.92 1$ = 415 руб.

Весь декабрь рубль оставался стабильным, несмотря на проходившие в то время бурные политические события: Съезд народных депутатов, на котором было отправлено в отставку правительство Гайдара. "Рубль аплодирует правительству Гайдара! " — заголовки газет того времени.

Конечно же, на курс рубля влияет огромное множество факторов. И все же, наверное, ЦБ что-то такое предпринял. Тем более, что экс-председатель Центрального банка России Виктор Владимирович Геращенко делал доклад на Съезде, отчитывался о мерах по стабилизации финансового рынка. Откроем этот доклад и прочитаем внимательно. И в одном абзаце найдем фразу о том, что с начала декабря 1992 года во всех расчетно-кассовых центрах ЦБ РФ стали применяться **криптографические** устройства для защиты от подделок почтовых и телеграфных авизо.

Любому серьезному проекту (а оснащение ЦБ РФ криптографическими методами защиты — это более чем серьезный проект!) всегда должна предшествовать длительная и кропотливая работа: НИР, ОКР, опытная эксплуатация,

приемка, доработка, обучение персонала, составление документации и так далее. Этот процесс может растянуться на годы. А если к этому добавить традиционно огромное число больших и прочих начальников, стремящихся увековечить свое участие в проекте, бюрократизм и стремление к минимуму ответственности, то естественно встает вопрос: а как же ЦБ удалось в 1992 году, в "смутный" период, оперативно (а фактически за два месяца!) решить проблему внедрения экзотических в то время криптографических устройств и спасти народные деньги от разграбления?

Многое познается только в кризисные времена. Например, с незапамятных времен в Советской Армии были переговорные таблицы в виде толстой книги. Об удобстве их использования особо никто не задумывался: солдату прикажут — он и "Капитал" Карла Маркса в качестве переговорной таблицы будет использовать. Но вот началась война в Афганистане. Под обстрелом — не до "Капитала", тут бы побыстрее с командиром связаться и поддержки попросить, а то накроют. И пошла связь открытым текстом, пошли из-за этого человеческие потери, и тогда наконец решили придумать что-то на замену этим таблицам. Придумали.

В городе Зеленограде, нашей Силиконовой долине, есть завод "Ангстрем", выпускающий советскую электронику, в том числе и бытовые программируемые калькуляторы. А не переделать ли такой калькулятор под специализированное устройство, предназначенное заменить в СА переговорные таблицы времен Царя Берендея? Легко сказать — переделать. Вернее, сравнительно легко оказалось придумать необходимые для такого устройства алгоритмы, т. е. всю "математику". А дальше — советская промышленность того (середина 80-х годов XX века) времени. Ей подавай дефицитные в то время материалы для изготовления специализированной микросхемы, включай в план, обхаживай и ублажай, в том числе, конечно, и деньгами. А деньги откуда?

И скрепя сердце решили начальники: изготовим таких калькуляторов побольше, часть из них пустим в открытую продажу (нарождавшимся тогда коммерсантам) и заломим за них умопомрачительную цену: в те времена практически никаких криптографических устройств на рынке не было, возьмут и по такой цене. А на вырученный от продажи доход оснастим Советскую Армию. Так в 1991 году на рынке появился портативный шифратор "Электроника — МК 85 С" (рис. П1).

Тут и смутные времена подоспели. Сменилась власть в нашем Государстве, армия перестала быть Советской, да и вообще то, что было плохо вчера, стало престижным сегодня. "Учитесь жить в условиях рынка, зарабатывать деньги!" — так стали наставлять своих подчиненных начальники. Стали учиться. Показывать свою продукцию, представлять ее на выставках, рекламировать. Кто пошустрее — те и частные фирмы создавать.

Переход к рынку, как мы теперь знаем, весьма труден, а ко всеобщему — тут и говорить нечего. И пошли в нашем Государстве фальшивые, или, как

их еще называли, "чеченские" авизо. Украдено по ним было, по приведенным в газете "Аргументы и Факты" оценкам МВД, чуть ли не треть национального бюджета. Как от них защититься? Увидев на одной из выставок "Электронику МК-85С", ЦБ РФ принял решение: это то, что надо.

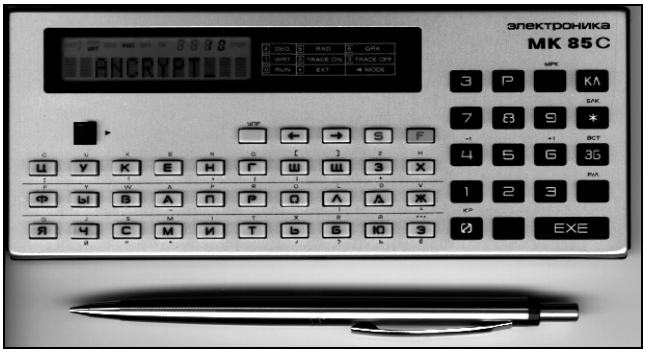

Рис. П1. Портативный шифратор "Электроника МК-85 С"

Истина — конкретна. Все в принципе может быть прекрасно, но дьявол всегда скрывается в деталях. Ведь это устройство предназначалось для шифрования информации, а ЦБ требовалась, фактически, цифровая проверочная комбинация, некое подобие цифровой подписи. А это совсем разные задачи. Кроме того, любое криптографическое устройство требует ключей, надо создать систему выработки и смены ключей, подготовить нормативные документы, обучить операционисток ЦБ, женщин, как правило ни разу в жизни ничего не слышавших ни о какой криптографии, работать с устройством и ключами к нему, подготовить программный аналог устройства на персональном компьютере. И все за два месяца. Ежу, как говорят математики, понятно, что в нашем Государстве такое может произойти только в одном случае: если число начальников, принимающих в этом участие, будет минимальным, а в идеале — стремиться к нулю. Попросту говоря, ЦБ напрямую обратился за помощью к специалистам.

Много воды утекло с той поры. В 1995 году был принят печально известный Указ № 334 Президента России, который я бы назвал криптографическим "Указом о трех колосках", откровенно ставящий целью отбить у любого нормального человека всякое желание связываться с криптографией. Хорошо это или плохо? Смотря с какой точки зрения подходить к этому вопросу. Будучи в 1992 году в самой гуще событий с ЦБ, невольно задаешься вопросом: а если некоторая аналогичная кризисная ситуация повторится, не будут ли интересы бюрократического аппарата, уютно чувствующего себя под этим Указом, ставиться превыше всего? Найдутся ли специалисты, к которым можно будет обратиться напрямую? И даже в некризисные времена в состоянии ли одно ведомство охватить все многообразие задач, в которых требуются криптографические приложения? Ведь как заметил один корреспондент

в беседе с вице-президентом Академии криптографии РФ Н. Н. Андреевым (журнал "Конфидент", N 23), "несколько смущает тот факт, что все выданные ФАПСИ сертификаты, по сути, выданы самим себе".

А между тем в мире идут как раз обратные процессы. Вот выдержка из Международного отчета "Криптография и свобода" о ситуации с криптографией, опубликованного Electronic Privacy Information Center, Вашингтон. Отчет по странам:

Соединенные Штаты

2000 ЖЕЛТЫЙ СЕКТОР/ЗЕЛЕНЫЙ СЕКТОР
1999 ЖЕЛТЫЙ СЕКТОР
1998 ЖЕЛТЫЙ/КРАСНЫЙ СЕКТОР

В течение долгого времени правительство США прилагало все усилия, чтобы ограничить разработку и распространение шифровальных технологий. За последние двадцать лет США неоднократно предпринимали такие попытки, манипулируя стандартами, рекомендуя принятие соответствующих законов и вводя экспортный контроль. Лишь недавно правительство признало, что электронный бизнес стал важным фактором американской экономики, и стало постепенно отказываться от прежней безрезультатной политики, вызывавшей негативные реакции в обществе. Наконец, в январе 2000 года американская администрация объявила о новых экспортных правилах и об ослаблении контроля над экспортом.

Франция

2000 ЖЕЛТЫЙ СЕКТОР/ЗЕЛЕНЫЙ СЕКТОР
1999 ЖЕЛТЫЙ СЕКТОР/ЗЕЛЕНЫЙ СЕКТОР
1998 КРАСНЫЙ СЕКТОР/ЖЕЛТЫЙ СЕКТОР

В 1999 г. политика в области крипто во Франции серьезно переменилась. 19 января 1999 года премьер-министр Лионель Жоспин (Lionel Jospin) объявил о коренном повороте в политике о криптографии. Эта политика отменяла существовавшую ранее сложную систему лицензирования импорта и внутреннего использования крипто, обязательную регистрацию ключей для частных пользователей и существование "доверенных" лиц государства.

Великобритания (Соединенное Королевство Великобритании и Сев.Ирландии)

2000 ЖЕЛТЫЙ СЕКТОР
1999 ЖЕЛТЫЙ СЕКТОР/ЗЕЛЕНЫЙ СЕКТОР
1998 ЗЕЛЕНЫЙ/ЖЕЛТЫЙ СЕКТОР

Сегодня в этой стране нет контроля ни за внутренним использованием крипто, ни за импортом криптографических средств. Есть лишь требование к радиолюбителям передавать свои данные в эфир в незашифрованном виде.

Великобритания всегда была самым верным соратником США в деле продвижения ограничений на шифрование. С 1996 года британское правительство (сначала консервативное, потом лейбористское) предлагало различные меры по развитию систем депонирования ключей. До сих пор эти попытки не имели успеха из-за противодействия со стороны общества и бизнеса.

Германия

2000 ЗЕЛЕНЫЙ СЕКТОР
1999 ЗЕЛЕНЫЙ СЕКТОР
1998 ЗЕЛЕНЫЙ СЕКТОР

Германия всегда была на переднем фронте в том, что касалось борьбы с ограничениями на криптографию. Она создавала противовес американским усилиям по внедрению систем депонирования ключей и международных ограничений на крипто. Германия сыграла заметную роль в создании в 1997 г. документа ЕС по криптографии и цифровым подписям. В 1999 г. благодаря усилиям Германии депонирование ключей не было включено в Вассенаарское соглашение.

Корея (Южная)

2000 ЗЕЛЕНЫЙ/ЖЕЛТЫЙ СЕКТОР
1999 ЗЕЛЕНЫЙ/ЖЕЛТЫЙ СЕКТОР
1998 ЖЕЛТЫЙ СЕКТОР

В своем отчете Организации по экономическому сотрудничеству и развитию Республика Корея указала, что в стране нет ограничений ни на импорт, ни на использование криптографии в частном секторе. Правительство рассматривает возможность введения общих правил для обращения с открытыми ключами.

Российская Федерация

2000 КРАСНЫЙ СЕКТОР
1999 КРАСНЫЙ СЕКТОР
1998 КРАСНЫЙ СЕКТОР

Импорт крипто и его использование регулируются в России следующими актами:

- Пунктом 5 Указа Президента N 334 от 3 апреля 1995 года, в котором запрещается импорт шифровальных средств без наличия лицензии.
- Пунктом 4 того же Указа, в котором запрещена любая деятельность по разработке, продаже и использованию шифровальных средств без лицензии, выдаваемой Федеральным агентством правительственной связи и информации (ФАПСИ), российским аналогом АНБ.
- Общим решением N 60 Гостехкомиссии и ФАПСИ от 24 июня 1997 года, определяющим процедуру получения лицензий на импорт и использование шифровальных средств.

По имеющимся данным, средства шифрования, не лицензированные ФАПСИ, широко доступны в России. Тем не менее, в феврале 2000 года Microsoft объявила,

что по соглашению с ФАПСИ выпускает русскую версию Windows 2000 без стойкого крипто. В декабре 1999 г. группа известных деятелей Интернет-сообщества подписала открытое письмо руководству страны, в котором призывала не устанавливать контроль над Internet, в частности, над крипто. Авторы письма заявили, что такой контроль снизит экспортные возможности российского бизнеса и уменьшит привлекательность электронного бизнеса в России для инвесторов.

Задав в поисковой системе Yandex слово "Криптография", я получил в ответ свыше 45000 ссылок. Де-факто, Россия уже пошла по пути других развитых стран и криптография перестала быть монополией спецслужб. Надеюсь, что вскоре это произойдет и де-юре. Число людей, желающих разобраться в криптографических хитросплетениях, стремительно растет. И неизбежно, вместо слепого поклонения бумаге с высочайшими подписями и печатями, криптографическим ГОСТам, отступление от которых выжигается каленым железом, у людей будут возникать вопросы:

- А какова оценка стойкости шифра в двоичных операциях?
- Надежность наилучшего метода определения ключа?
- Группа реализуемых преобразований?
- Является ли он наиболее оптимальным по скорости работы?
- Каков период вырабатываемой гаммы?
- Какова вероятность перекрытий гаммы?

И множество других подобных вопросов, ответы на которые, вместо общих фраз "стойкое нестойкое крипто", давали бы объективные характеристики того или иного криптографического решения.

Но этого мало. Сама по себе криптосхема, образующая криптографическое ядро, не гарантирует безопасности. Для нее нужна удобная система выработки, хранения и распределения ключей, система учета и контроля, система передачи ответственности, система управления и еще много других вспомогательных систем. Все это можно объединить одним словом — интерфейсная оболочка. И как показывает опыт, объем интерфейсной оболочки составляет 95—99% от общего объема законченной прикладной системы защиты компьютерной информации. Правильное построение интерфейсной оболочки, бережное и чрезвычайно деликатное обращение с криптографическими ключами (как с секретными, так и открытыми), удобство пользователя, автоматизация процедур обработки информации — все эти факторы напрямую влияют на информационную безопасность.

Я пытаюсь обобщить в этой книге весь свой 25-летний опыт работы в области криптографии. Постарался изложить его доступным языком. Надеюсь, что книга окажется полезной современному читателю.

Введение

Актуальна ли массовая популяризация теоретических основ криптографии? Этот вопрос был задан в 1998 году корреспондентом журнала "Компьютерра" заместителю заведующего лабораторией МГУ по математическим проблемам криптографии, кандидату физико-математических наук Валерию Владимировичу Ященко. "Да, вне всякого сомнения", — таков был ответ. Действительно, слово "криптография" достаточно часто можно встретить в Internet, различных газетных публикациях, статьях, специализированных журналах. Вышло несколько книг, целиком посвященных криптографии. Но все ли подобные публикации написаны квалифицированными специалистами в этой области? Вот опять же мнение В. В. Ященко: "В Internet масса сайтов, где встречается слово "crypto". Я на некоторые из них заглядывал, и у меня осталось впечатление, что это огромная мусорная свалка. И чтобы отсеять весь этот мусор от чего-то добротного, нужно иметь соответствующую квалификацию. Там масса неправильного, вредного, масса обманов. Так что, если говорить о публикации текстов, то нужно уметь четко отделить рекламный момент от содержательного. Это очень трудно. Ведь нет же никаких стандартов, нет способов проверки качества! Вот тебе дают продукт. Как ты можешь пытаться его взломать? Либо надо самому иметь высочайшую квалификацию, либо приходится довериться некоему коллективу специалистов. Потому что единственной гарантией стойкости шифра является то, что много квалифицированных криптоаналитиков пытались его вскрыть, но не смогли".

Практически всю свою сознательную жизнь — около 25 лет — я был так или иначе связан с криптографией. Сразу после школы — Технический факультет Высшей школы КГБ СССР, где готовили специалистов-криптографов. Сейчас он называется Институтом криптографии, связи и информатики, имеет свой сайт в Internet — **http://www.fssr.ru**. После его окончания — Теоретический отдел в 8 Главном управлении КГБ СССР, который занимался вопросами анализа и синтеза различных шифрсистем,

аспирантура, диссертация — все по тематике, связанной с криптографией. Мне приходилось работать в коллективе математиков-криптографов, квалификация которых без всяких сомнений может быть оценена по самым высоким мировым стандартам. Постоянные дискуссии и споры, вера только в ставшие очевидными доказательства, поиск нетривиальных решений, но вместе с тем достаточно демократичный и свободолюбивый дух общения, отсутствие слепого преклонения перед авторитетами, дружеские шутки и юмор — вот стиль работы Теоретического отдела в начале 80-х годов прошлого века. Советская криптографическая школа того периода на равных конкурировала с американцами.

Общий развал начала 90-х годов XX века не мог обойти и криптографию. Очень многие криптографы в этот период ушли в банки, коммерческие структуры, частный бизнес, стали заниматься вопросами, не относящимися к криптографии. Традиции оказались забытыми, в официальной криптографической политике воцарился чиновничий подход, далекий от криптографии как одного из разделов математики. А потребности в криптографии, наоборот, выросли. Появился Internet, электронная почта, разнообразные коммуникации, виртуальные банковские расчеты, покупки, электронный бизнес. И везде оказалась необходимой защита информации, осуществляемая с помощью криптографических методов. И не просто защита, а гарантированная защита, т. е. такая, которой не страшно было бы доверять финансовые тайны, являющиеся иногда в прямом смысле вопросами жизни и смерти.

Я попытался в этой книге сжато изложить основные идеи, необходимые для построения гарантированных криптографических систем. Книга разбита на 6 частей, в каждой части — несколько глав.

В первой части вкратце изложены некоторые сведения из истории криптографии и те основные понятия, которые наиболее часто встречаются в современной криптографической литературе.

В первой главе приведен ряд сведений из истории криптографии, в основном — Российской, за период с Первой мировой войны. Этот период, на мой взгляд, очень показателен тем, насколько важен профессиональный подход к криптографии, к каким негативным последствиям может привести пренебрежение вопросами криптографии или недостаточная квалификация криптографов.

Вторая глава посвящена результатам американского математика Клода Шеннона, который в конце 40-х годов XX века одним из первых осознал необходимость строгого математического подхода к криптографии и сумел обосновать понятие абсолютно стойкого шифра.

В третьей главе идет речь о том, осуществимы ли идеи Шеннона на практике. Здесь же даются и некоторые подробности того, как в 1992 году появилась система защиты телеграфных авизо для Центрального банка России.

Четвертая глава посвящена ключам. Что такое система с открытым распределением ключей, ее достоинства и недостатки по сравнению с традиционной ключевой системой.

В пятой главе объясняется, что такое блочный шифр, а также дается описание двух наиболее известных блочных шифров — алгоритмов DES и ГОСТ 28147-89. Я не могу скрыть своего негативного отношения к этим алгоритмам — они слишком медленны и не предназначены для программной реализации. Предлагаемый оригинальный алгоритм имеет предысторию не меньшую, чем, например, ГОСТ 28147-89, ориентирован на операции с байтами и по своей производительности значительно превосходит ГОСТ 28147-89. Вопрос о том, почему он не стал российским стандартом, с математикой и криптографией, на мой взгляд, связан очень слабо.

Шестая глава целиком посвящена одной из наиболее красивых криптографических идей — электронной подписи. Что такое хэш-функция, какие с ней связаны проблемы — обо всем этом также можно прочитать в этой главе.

Любая криптографическая идея плодотворна только тогда, когда удалось осуществить ее удобную реализацию. Проблемам реализации основных криптографических идей, изложенных в первой части книги, посвящены ее остальные части. Логика их построения — от простого к более сложному, от простых операций, связанных с шифрованием и подписью файлов к построению программных систем по защите информации, реализованных на базе Microsoft Outlook, Lotus Notes или на базе оригинальной системы электронного документооборота.

Здесь возникает много специфических и, я бы сказал, эксплуатационных проблем: как вырабатывать секретные ключи, как хранить открытые ключи, как проводить смену ключей, как распознавать зашифрованный текст, как встроить криптографические процедуры в основное меню, как автоматизировать процессы криптографической подготовки и пересылки электронных документов и т. п. Все эти эксплуатационные проблемы напрямую связаны, с одной стороны, с созданием необходимых удобств пользователям, а с другой — с информационной безопасностью. И следует отметить, что ради достижения гарантированной информационной безопасности иногда приходится идти на ограничение удобств пользователей. Если, к примеру, программная система, отвечающая за информационную безопасность, может "помочь" пользователю в случае, если он забыл пароль или потерял дискету с ключом, то точно так же она сможет оказать помощь и потенциальному злоумышленнику, взявшемуся за ее взлом. Гарантированная система доставляет и некоторые другие неудобства пользователям, например, необходимость периодической полной смены секретных и соответствующих им открытых ключей. Есть предел в компромиссах между удобством и информационной безопасностью. Задача криптографа — ясно и четко рассказать обо всем этом заказчику, а он сам принимает решение, что важнее: удобства или гарантированная безопасность. А еще одна задача криптографа —

не создавать при этом лишних, надуманных неудобств, как это иногда бывает, если к криптографии прикладывают руку чиновники от криптографии.

Нужна ли широкая публикация исходных текстов криптографических программ? Ответ отрицательный, и даже не только потому, что при этом могут быть затронуты права интеллектуальной собственности их разработчика. Дело в том, что современные профессионально подготовленные криптографические системы, как правило, практически не поддаются взлому с помощью чисто криптографических, математических методов. И потенциальный злоумышленник начинает искать обходные пути — пытается похитить секретный ключ или открытый текст. Создает так называемые криптовирусы, задача которых — обычное воровство ключа, только с помощью изощренных программных методов. И наличие исходного текста сильно помогает такому взломщику при написании криптовируса.

Но, с другой стороны, потребитель криптографической системы должен иметь некоторое углубленное представление об этой системе и особенностях ее программной реализации с тем, чтобы принять решение о том, какие секреты можно ей доверить. Разработчик вынужден немного приоткрыть тайны своих решений, но не настежь. Примерно такими соображениями я руководствовался при написании этой книги. В них достаточно много различных листингов, но они, как правило, не являются законченными криптографическими процедурами, в некоторые их параметры я сознательно внес коррективы. В основном они предназначены для разработчиков интерфейса к криптографическим системам. По собственному опыту знаю, насколько более ценным при программировании оказывается листинг с примерами, чем длинное и витиеватое описание.

Последняя, шестая часть книги затрагивает вопросы, имеющие к криптографии косвенное отношение. Это идентификация пользователя по отпечатку его пальца. Но криптографические идеи могут быть плодотворно использованы и здесь. Отпечаток пальца — это некоторая случайная черно-белая матрица, источник, по которому можно построить индивидуальный секретный ключ пользователя. А этот ключ можно использовать как в криптографии, так и для иных целей: для доступа в помещение или к каким-то ресурсам, в криминалистике — для опознания человека, для создания специализированных баз данных и т. п. Такой ключ нельзя оставить без присмотра, потерять, изменить. Потенциальный злоумышленник, помимо чисто технических трудностей, имеет в этом случае дело с задачей перебора, для которой можно дать более-менее точные математические оценки ее сложности.

В книге используется общепринятая криптографическая терминология: секретный и открытый ключ, система с открытым распределением ключей, открытый и шифрованный текст, гамма наложения, блочный шифр и т. п. Эти термины по ходу изложения поясняются, так что читатель, взявший в руки эту книгу, вполне может быть человеком, весьма далеким от проблем криптографии. Примеры листингов программ подготовлены с помощью

языка программирования Borland C++ Builder 3, Client/Server Suite. Я с ним сильно подружился за последние годы. Его русскоязычное описание можно найти, например, в книге Рейсдорф К., Хендерсон К. Освой самостоятельно Borland C++ Builder. – М.:Издательство БИНОМ, 1998, или в книге Шамис В. А. Borland C++ Builder 3. Техника визуального программирования. – М.: Издательство "Нолидж", 1998.

Я постарался изложить криптографические идеи доступным и понятным языком, не перегружая текст излишними формулами, непонятными терминами. Иногда с анекдотами и юмором. Но нас так учили в Высшей школе КГБ: "Через 15—20 минут рассказа — отвлеките внимание аудитории, расскажите ей какую-нибудь смешную историю или анекдот". После таких слов лектора вся аудитория сразу же начала отсчет времени, ибо эта была последняя лекция перед обедом. Через 20 минут с полсотни голодных глоток дружно заскандировали: "А-нек-дот! А-нек-дот!" Лектор вынужден был отпустить нас на обед.

Время пошло!

Часть I

Основы криптографии

Глава 1. История криптографии

Глава 2. Стойкость шифра

Глава 3. Генератор гаммы

Глава 4. Ключевые системы

Глава 5. Блочные шифры

Глава 6. Электронная подпись

Как я представляю себе своего читателя? Это молодой человек, лет 20—30, может побольше, интеллигентный, эрудированный, с чувством юмора. Работает программистом в банке. А может и не программистом и не в банке. Но с компьютерами тесно связан, даже, бывает, иногда пишет свои программы. Или, по крайней мере, разбирается в программировании.

И вот вызывает его однажды к себе начальник и говорит:

— Давай-ка ты, парень, напиши какую-нибудь защиту для нашей очень ценной базы данных.

А может и не для базы данных, может для системы "Банк-клиент", "Банковский операционный день" или еще для какой-то другой системы. Мало ли в банке всяких баз данных и программ, с деньгами работающих. А то, что с деньгами работает, лучше все-таки защитить от посторонних глаз.

Так ведь есть уже много готовых систем защиты! Но начальника все они почему-то не устраивают. То ли очень сложные, громоздкие, то ли слишком дорогие, то ли операционная система не та, то ли вообще начальник не с той ноги сегодня встал или по пути на работу ему черная кошка дорогу перебежала. Сделай свою, оригинальную, и все тут! Отвечает мой читатель:

— Так ведь я ничего и не слышал про криптографию, никогда раньше никаких дел с ней не имел.

— Вот иди и разберись со всеми этими шифрами, ключами и электронными подписями!

И грозно сверкнул глазами.

Не бойся, читатель! Разберемся. И начальник твой тоже разберется.

Глава 1

История криптографии

Всем читателям, интересующимся историей криптографии, я настоятельно рекомендую прочитать давно ставшую классикой книгу Дэвида Кана "Криптографы". Об этой книге я впервые узнал, будучи слушателем Технического факультета Высшей школы КГБ в середине 70-х годов. Опытные и заставшие еще сталинские времена преподаватели, рассказывая нам, молодым 20-летним парням, об истории криптографии, упоминали о том, что в спецбиблиотеке есть таинственная книга Дэвида Кана про историю криптографии, но круг допущенных к ней читателей строго ограничен "по идеологическим соображениям". Запретный плод сладок. Будучи уже офицером КГБ, получил все-таки доступ к этой таинственной книге и сразу же стал искать в ней раздел про Россию. О послереволюционной России Дэвид Кан писал: "Одной из задач тайной полиции являлась защита диктатуры пролетариата от самих пролетариев, которые не обрели обещанного счастья при новоявленных диктаторах". Да, в 70-х годах это было страшной тайной.

Давайте немного проследим историю криптографии по книге Дэвида Кана, которую в 2000 году я свободно купил на книжном рынке у Кузнецкого моста (Кан Д. Взломщики кодов. — М.: Центрополиграф, 2000), да и по многим другим источникам, появившимся в последнее время в Internet.

Криптография зародилась в глубокой древности, однако ее назначение подчас было в "усилении тайны и, следовательно, колдовской силы поминальных текстов". Как пишет Дэвид Кан, "секретность проистекала из понятного желания древних египтян заставить прохожего прочитать их эпитафии и тем самым выразить умершим благословения, которые содержались в надгробных надписях. В Древнем Египте, с характерной для него верой в загробную жизнь, количество надписей быстро выросло до такой степени, что внимание к ним прохожих пошатнулось. Чтобы возродить их интерес, писцы нарочно делали надписи несколько туманными. Они ввели криптографические знаки, дабы привлечь внимание читателя, заставить его задуматься и вызвать у него желание разгадать их смысл".

Начало криптографии, в современном понимании этого слова, связывают с Юлием Цезарем. Вот что пишет об этом в своем цикле статей "Криптография. Ее истоки и место в современном обществе" Андрей Винокуров: "Первые действительно достоверные сведения с описанием метода шифрования относятся к периоду смены старой и новой эры и описывают шифр Цезаря — способ, которым Юлий Цезарь прятал свои записи от излишне любопытных глаз. С высоты достижений современной криптографии шифр Цезаря предельно примитивен: в нем каждая буква сообщения заменялась на следующую за ней по алфавиту. Однако для того времени, когда умение читать и писать было редким исключением, его криптостойкости вполне хватало. Использование шифра решало проблему секретности передаваемого сообщения, а проблема его подлинности решалась практически сама собой".

Известен также пример другого древнего шифра — шифра Сциталла. Он положил начало так называемым шифрам перестановки. Вот цитата из курса "История Криптографии", Московский государственный университет.

"Одним из первых приборов, реализующих шифр перестановки, является так называемый прибор СЦИТАЛЛА. Он был изобретен в древней "варварской" Спарте во времена Ликурга; Рим быстро воспользовался этим прибором. Для зашифровывания текста использовался цилиндр определенного диаметра. На цилиндр наматывался тонкий ремень из пергамента, и текст выписывался построчно по образующей цилиндра (вдоль его оси). Затем ремень сматывался и отправлялся — получателю сообщения. Последний наматывал его на цилиндр того же диаметра и читал текст по оси цилиндра. В этом примере ключом такого шифра являлся диаметр цилиндра и его длина, которые, по существу, порождают двухстрочную запись, указанную выше.

Интересно, что изобретение дешифровального устройства "АНТИСЦИТАЛЛА" приписывается великому Аристотелю. Он предложил для этого использовать конусообразное "копье", на которое наматывался перехваченный ремень, который передвигался по оси до того положения, пока не появлялся осмысленный текст".

В России криптография впервые появилась при Петре I, когда он начал "прорубать окно в Европу". Как пишет Дэвид Кан, "в XVIII веке Россия переняла у Запада одно из его полезных нововведений — "черные кабинеты". Так же, как в Англии и Австрии, у русских они размещались в почтовых отделениях. В число сотрудников "черных кабинетов" входили специалисты по вскрытию конвертов и подделке печатей, переводчики и дешифровальщики".

Но наибольшую значимость криптография всегда приобретала во время войн. Так, во время первой мировой войны разгром русских армий Самсонова и Ренненкампфа в Восточной Пруссии Дэвид Кан напрямую связывает со слабостью шифровальных средств, использовавшихся в тот период в русской армии. Вот несколько цитат из его книги.

"Хотя в начале войны Россия испытывала большие трудности в обеспечении своих войск всем необходимым, в том числе и средствами связи, уже в первой половине сентября 1914 года ей удалось полностью снабдить их шифровальными средствами. 14 сентября российская ставка верховного главнокомандования отдала распоряжение о том, что все военные приказы подлежат зашифровыванию.

Принятая шифрсистема основывалась на многоалфавитном шифре цифровой замены, в котором допускалось зашифровывание нескольких букв подряд по одному алфавиту. Этот шифр представлял собой таблицу, в верхней части которой в строку были выписаны буквы русского алфавита. Сама таблица состояла из восьми строк двузначных цифровых групп, выписанных в произвольном порядке. Строки отличались друг от друга порядком расположения в них этих групп. Слева они были бессистемно пронумерованы. При зашифровывании эти строки использовались поочередно: сначала под номером один, потом два и так далее. Каждая из строк применялась для зашифровывания нескольких знаков открытого текста. Количество знаков, подлежащих шифрованию данной строкой, определялось самим шифровальщиком. Для того чтобы адресат мог расшифровать полученное сообщение, в его заголовке пять раз проставлялась цифра, соответствующая количеству знаков, которые были зашифрованы каждой из строк. Когда в процессе шифрования оператор хотел изменить это число, он вставлял в текст шифровки пятизначную группу, элементами которой была одна и та же цифра, соответствующая новому числу знаков, шифруемых одной и той же строкой. Таким образом, шифртелеграммы русской армии состояли из групп букв, зашифрованных одним и тем же алфавитом. Длина каждой группы букв определялась однозначно по пятизначной цифровой группе, состоявшей из одной и той же цифры".

Обратите внимание на описание этого шифра. Мне пришлось перечитать этот абзац несколько раз, чтобы хотя бы приблизительно представить себе шифровальную систему Русской армии 1914 года. Подчас с помощью запутанного описания разработчик шифровальной системы пытается самоуспокоиться, скрыть грубые криптографические просчеты. Так было и в этом случае.

"Уже к 19 сентября молодой одаренный начальник русского отделения дешифровальной службы Австро-Венгрии капитан Герман Покорный вскрыл эту систему и полностью восстановил все ее строки. Дело в том, что такие шифрсистемы не представляли непреодолимых преград для криптоаналитиков, поскольку в шифртексте зачастую сохранялась структура наиболее часто встречавшихся в открытом тексте слов, таких как "атака" и "дивизия", которые полностью шифровались одной строкой таблицы".

Здесь мне хотелось бы немного прокомментировать этот абзац. В каждом открытом тексте — будь то разговорный текст, телефонное сообщение, компьютерный файл — обязательно присутствует преобладание встречаемости

одних символов перед другими. Например, в русском разговорном языке наиболее часто встречающиеся буквы — это СЕНОВАЛИТР. Если усидчивый читатель не поленится и возьмется подсчитать статистику встречаемости различных букв, например, на этой странице, то наибольшее преобладание в статистике будут иметь как раз эти буквы. Предположим теперь, что мы взяли и заменили во всем тексте, например, букву С — на К, Е — на Р, Н — на И, О — на П, В — на Т, А — на О, Л — на Г, И — на Р, Т — на А, Р — на Ф. Текст получится визуально нечитаемым, но наиболее часто встречающиеся буквы будут КРИПТОГРАФ. Статистика встречаемости букв от этого не изменится. Можно даже заменить буквы разными значками, палочками и крючочками, "пляшущими человечками", как это описано у Конан Дойля в одноименном рассказе, — это не меняет статистики текста. Преобладание встречаемости одних знаков перед другими сохраняется. Почему возникает подобный эффект — об этом поподробнее мы поговорим в следующей главе.

Такой шифр в криптографии принято называть шифром простой замены. Он вскрывается достаточно просто: зная наиболее часто встречающиеся буквы языка, собираем статистику шифрованного текста и наиболее часто встречающиеся буквы шифртекста заменяем на наиболее часто встречающиеся буквы открытого текста. Для помощи используем, например, "протяжку вероятного слова", как это делал Шерлок Холмс в рассказе про пляшущих человечков, принятые в предполагаемом открытом тексте стандарты (деловая переписка, как правило, начинается со слов Уважаемый г-н, в EXE-файле компьютерной программы в начале присутствует заголовок, начинающийся с MZP, и т. п.). Единственное, что нужно для вскрытия простой замены, — знать статистику языка, на котором подготовлено открытое сообщение, и сравнительно немного шифрованного текста для набора его статистики. Так типовое сообщение на русском языке при шифровании его с помощью шифра простой замены вскрывается уже при 24 символах.

Итак, получается, что в 1914 году гибель русских армий Самсонова и Ренненкампфа поизошла во многом по той причине, что действующие в то время шифры не смогли скрыть статистику открытого текста.

С этими азами криптографии, к сожалению, нередко приходится сталкиваться и сейчас. Типичное искушение, которое испытывает программист, создающий парольную систему защиты: взять пароль, сложить его со знаками открытого текста. Пароль короткий, а знаков много. Ничего, пароль можно повторить несколько раз, пока не кончится текст. Такая, или подобная ей, система "защиты" фактически является простой заменой, только не всех знаков подряд, а находящихся друг от друга на расстоянии длины пароля, и без труда вскрывается мало-мальски грамотным хакером. Протянув вероятное слово (например, серию пробелов в начале строки), легко вычислить и сам пароль.

Продолжим читать Дэвида Кана дальше. "Примерно в это же время (25 сентября) русские впервые сменили шифр. Сами строки остались без изменений, переменился порядок выбора строк для шифрования. Новый шифр был вскрыт Покорным в течение нескольких минут: все трудности отпали, когда одна из русских радиостанций передала зашифрованную новым шифром телеграмму, переданную еще до смены шифра".

Смена ключей. До боли знакомая проблема, ведь это весьма хлопотная и нервная процедура. Представьте: вы наладили закрытый документооборот, все неурядицы улеглись, система работает без сбоев, а тут вдруг надо взять и своими собственными руками ее разломать. Отменить старые ключи. Выработать и разослать новые. Получить подтверждения их получения. Проконтролировать, чтобы все одновременно в "час X" перешли со старых на новые ключи. И наверняка в большой сети найдется разгильдяй, который что-нибудь напутает. Криптографический дух Русской армии 1914 года прямо витает над современными банками!

И все-таки смена ключей необходима. Конечно же, правильно построенная современная криптографическая система не содержит таких грубых "дыр", какие были в 1914 году, уж что-что, а статистика шифртекста в ней не имеет преобладаний. И вообще, вскрыть современный шифр чисто умозрительными методами, с карандашом и бумагой, подобно Герману Покорному, невозможно. Даже для анализа шифра современному хакеру нужен мощный компьютер и, как правило, INTERNET-поддержка. Но в жизни всякое может случиться. Операционистка в банке, работающая с шифрсистемой и недовольная своей зарплатой, может уволиться. Дискета с секретным ключом, оставленная на столе без присмотра, может попасть в чужие руки. В общем, возможна компрометация ключей. Тогда смена ключей позволяет локализовать негативные последствия, вызванные такой компрометацией.

И еще одна старая песня. Нельзя ли при смене ключей новые ключи шифровать с помощью старых? Это голос управления безопасности, стремящегося таким образом облегчить себе жизнь. А зачем тогда вообще менять ключи? Если старый ключ скомпрометирован, то новый, зашифрованный на старом, также автоматически оказывается скомпрометированным. Читайте Дэвида Кана, господа!

Вторая мировая война. "Советская шифровальная служба в основном учла плачевный опыт своей российской предшественницы времен первой мировой войны. Об этом свидетельствует следующий полный драматизма обмен радиограммами между советскими воинскими частями 22 июня 1941 г. Сразу же после внезапного нападения Германии на Советский Союз один из передовых постов Красной Армии передал по радио открытым текстом: "Нас обстреливают. Что нам делать?" На что последовал следующий ответ: "Вы с ума сошли! Почему Ваше сообщение не зашифровано?"

Во время второй мировой войны шифровальная служба Красной Армии использовала в основном коды с перешифровкой".

Прервемся в этом месте. Коды с перешифровкой — что это такое? В армии принято отдавать короткие четкие команды: "Батарея — огонь!", "Прицел 120", "Взвод — в атаку" и т. п. Для таких коротких команд изготавливают специальную кодовую книгу, где каждому слову в команде ставят в соответствие некоторую цифровую комбинацию, наиболее частым словам — покороче, редким — подлиннее. Такой код называется разнозначным. Вместо слов "атака", "дивизия" по рации можно передать короткий код, например, 032, 1458. Но если кодовую книгу использовать напрямую, без дополнительной защиты, то мы попадемся в ту же ловушку, что и Русская армия в 1914 году. Кодовая книга не скрывает статистики текста, часто повторяющиеся команды выделяются и вскрываются. Поэтому их еще и перешифровывают, т. е. накладывают на них некоторую гамму наложения. И тут опять тянется ниточка в наше время. Помните разрабатывавшееся для Советской Армии портативное шифровальное устройство "Электроника МК 85 С"? Так вот, одним из его назначений как раз и была перешифровка кодовых книг. Что это значит? А то, что в "Электронике МК 85 С" был предусмотрен режим, когда как открытый, так и шифрованный текст являются только цифрами. И кто бы мог предполагать при разработке "математики" для "Электроники МК 85 С", что именно этот режим и спасет в 1992 году Центральный банк России от фальшивых авизо! Неисповедимы пути Господни!

Вернемся к Кану. "Немецкая радиоразведка против Советского Союза была малоэффективной. В стратегическом отношении она вообще не имела ни одного сколько-нибудь заметного успеха. Немцы оказались не в состоянии вскрыть шифрсистемы, применявшиеся для засекречивания переписки высшего советского военного командования". "Явная неспособность немецких криптографов вскрыть советские стратегические шифрсистемы, с помощью которых засекречивалась самая важная информация, вынудила одного немецкого криптографа признать, что, хотя Россия и проиграла первую мировую войну в эфире, во время второй мировой войны она сумела взять реванш за свое поражение". "Шифрпереписка советских разведчиков не поддавалась дешифрованию. Большинство из них использовало стандартную для советской агентуры того времени шифрсистему, которая была триумфом шифровальной техники. Она представляла собой доведенную до совершенства старую систему, применявшуюся русскими революционерами, и объединяла в себе шифр равнозначной замены с одноразовой гаммой. В Москве обоснованно считали ее абсолютно стойкой".

Какой ценой строилась абсолютно стойкая советская криптография — об этом можно узнать из романа Александра Солженицына "В круге первом". А вот выдержка из воспоминаний Леонида Александровича Кузьмина о развитии криптографии в первый послевоенный период, опубликованных в журнале "Конфидент", N 22.

"По окончании Великой Отечественной войны наступило время, потребовавшее не меньших усилий от советских криптографов, — период "холодной войны". Между тем многие научные работники военной криптографической службы были уже демобилизованы...

В этих условиях немногочисленные теперь квалифицированные криптографы военного призыва нашли в себе мужество обратиться через голову своего могущественного и страшного шефа — Лаврентия Берия — к самому "отцу народов". Вопреки ожиданиям, обращение было услышано и советская криптография была принята под крыло самого мощного органа Советского Союза — ЦК ВКП(б). Осенью 1949 года Политбюро ЦК приняло ряд важнейших для советской криптографии решений, суть которых сводилась к следующему:

- на базе разрозненных подразделений создавалось Главное управление специальной службы (ГУСС) при ЦК ВКП(б) с одновременным выделением больших средств для его становления и развития;
- были приняты меры к привлечению наиболее сильных ученых как для выполнения оперативных задач криптографической службы, так и в роли преподавателей для подготовки новых высококвалифицированных кадров;
- для выполнения последней цели создавались Высшая школа криптографов и закрытое отделение механико-математического факультета МГУ.

Реализация этих решений за каких-то три с небольшим года позволила коренным образом изменить лицо советской криптографии.

В начале 1949 года я закончил Московский авиационный институт и начал работу в КБ знаменитого С. В. Ильюшина. В декабре того же года в ленинградском райкоме ВКП(б) я в числе других комсомольцев завода проходил один из этапов традиционной процедуры вступления в партию — собеседование со старыми большевиками. По окончании мероприятия меня неожиданно пригласили в другую комнату, где состоялся уже нетрадиционный разговор с неизвестным мне человеком средних лет. Эта беседа, резко изменившая мою судьбу, началась с обычных вопросов о семье, биографических данных, а затем перешла в другое русло:

— Нравится ли вам нынешняя работа?

— Да, нравится!

— Были ли во время учебы в МАИ трудности с математическими дисциплинами?

— Не было...

— Это очень хорошо! Вот анкета — заполните её сейчас же, а вот направление на медкомиссию, пройдете её завтра!

Продолжение этой истории последовало только через год, когда мне передали приказ от самого С. В. Ильюшина: явиться завтра в райком партии за новым назначением! Там последовало новое распоряжение: "Явитесь с паспортом в дом 21 по улице Дзержинского и позвоните по внутреннему телефону, там вас встретят".

Встретил меня тот же человек, который давал мне анкету в райкоме партии. Им оказался один из руководителей отдела кадров Главного управления

специальной службы при ЦК ВКП(б) майор М. Н. Хитров. Разговор снова короткий:

— Мы отобрали вас для учебы — в течение двух лет будете слушателем нашего вуза со стипендией 1800 рублей в месяц. Вскоре вам будет присвоено офицерское звание.

— Чему я должен учиться?

— Об этом узнаете позже. Вы, кажется, конструктор? До начала обучения остается полтора месяца, так что пока мы вас направим в наше конструкторское бюро. Там явитесь к генерал-майору Ивану Павловичу Волоску, который и введет вас в курс наших задач.

Через час я стою в небольшом кабинете перед тучным генералом, на груди которого сияет значок лауреата Сталинской премии. Генерал сразу перепоручает меня тут же вызванному заместителю — подполковнику Евстигнею Дмитриевичу Щукину, с которым мы, пройдя по коридору, заходим в одну из комнат, где стоят 4—5 кульманов и десяток столов. Он сажает меня на стул рядом с собой и тихим голосом, чтобы не мешать работающим людям, начинает разговор:

— Так вы авиаконструктор? Эта работа ценится нашим народом. Товарищ Сталин вам поставил задачу: "Летать дальше всех, выше всех, быстрее всех!" Наша же работа не афишируется, мы — криптографы, работаем в области секретной связи, с шифрами. Но и нам товарищ Сталин поставил задачу: "Читать всех, но наши переговоры и переписку читать никто не должен!" Я и мои товарищи создаём для этого специальную технику засекречивания телеграфной связи. Но об этом не говорят! Ни ваша мать, ни жена, ни один из друзей не должны знать, что теперь и вы причастны к этой работе!"

В заключение этой главы не могу не привести еще одной цитаты из Дэвида Кана. "Представляет интерес поразмышлять об успехах русской криптографии. Россия сама по себе остается загадкой, овеянной тайной из тайн. То же самое касается и ее средств связи. Одноразовые шифрблокноты обеспечивают надежную защиту для сообщений российских разведчиков, военных, дипломатов и работников тайной политической полиции. Грамотно сконструированные шифры навечно сохраняют в секрете от врагов России ее наиболее важную дипломатическую, агентурную и военную переписку. В период "холодной войны" русские сумели вскрыть шифры американского посольства в Москве. Такие подвиги свидетельствуют об их осведомленности, базирующейся на глубоком понимании шифровального дела и криптоанализа. Исходят ли эти знания из врожденной способности русских к естественным наукам, что позволило им первыми запустить искусственные спутники Земли, или же из большого опыта в области криптологии, которая исправно служила коммунистическим диктаторам в России в их борьбе за власть, или же из привычки, которая впиталась в кровь всякому жителю тоталитарного общества, на каждом шагу видеть и разгадывать секреты, или из врожденной любви славян ко всему таинственному в природе, но так или иначе русские вознесли достижения своей страны в криптологии до высоты полета ее космических спутников".

Глава 2

Стойкость шифра

Интуитивно ясно, что стойкость шифра — это его способность противостоять попыткам взлома. Что такое попытка взлома? Это попытка потенциального злоумышленника получить некоторую дополнительную информацию об открытом тексте, зашифрованном с помощью этого шифра. А что такое "информация об открытом тексте"? Как дать этому, в общем-то довольно туманному, понятию строгое математическое определение? Ответ на эти вопросы был получен американским математиком Клодом Шенноном в конце 1940-х годов. К точным математическим формулам мы перейдем чуть позже, а пока, для неискушенного в вопросах математики читателя, я позволю себе привести цитату, посвященную Клоду Шеннону из книги Дэвида Кана.

"Клод Шеннон родился в городе Петоски в штате Мичиган 30 апреля 1916 года. Поступив в Мичиганский университет, Шеннон занялся серьезным изучением электротехники и математики. Именно там у него впервые проявился интерес к теории связи и криптографии.

В Массачусетском технологическом институте Шеннон написал диссертацию, в которой содержалось множество новаторских идей, связанных с разработкой телефонных систем. Получив степень доктора математических наук, Шеннон поступил на службу в лабораторию компании "Белл", которая была заинтересована в реализации этих идей на практике.

"Во время второй мировой войны, — рассказывал Шеннон, — компания "Белл" работала над засекречиванием информации. Я тогда занимался системами связи и был назначен в несколько комиссий, изучавших криптоаналитические методы. Начиная примерно с 1941 г. исследования в области математической теории связи и теории шифров велись мной одновременно. Я трудился в обеих областях сразу, и кое-какие идеи в одной из них возникали у меня, когда я работал в другой. Я не хочу сказать, что одна из этих областей доминирует над другой. Просто они настолько тесно связаны, что их невозможно разделить". Хотя разработка обеих теорий была в основном завершена примерно к 1944 г., Шеннон продолжал уточнять полученные

результаты до 1948—1949 гг., когда они были опубликованы в виде двух отдельных статей в солидном теоретическом журнале "Bell system technical journal".

В обеих статьях Шеннона — "Математическая теория связи" и "Теория связи в секретных системах" — идеи излагаются в краткой, математической форме. Точный и выразительный стиль изложения Шеннона вдохнул в них жизнь. В результате его первая статья породила теорию информации, а вторая — теорию шифров.

Главной в работах Шеннона является концепция избыточной информации. Избыточность, по Шеннону, означает, что в сообщении содержится больше символов, чем в действительности требуется для передачи информации. Избыточность связана с излишком правил, обременяющих все языки. Второй источник языковой избыточности происходит из человеческой лености, которая заставляет людей выбирать легко выговариваемые и узнаваемые звуки. Процесс корректорской правки текста сродни криптоанализу, ибо при вскрытии шифров криптоаналитики также используют свое знание правил фонетики, грамматики, идиом, слов-функций и фонетических склонностей, которые в совокупности и придают языку избыточность.

Тут я позволю себе немного дополнить признанного мэтра криптографии. Появление компьютеров и специфической компьютерной информации (текста, набранного в текстовом редакторе, файла, содержащего рисунок, exe-файла и т. п.) еще больше обострило проблему избыточности. К упоминаемым Шенноном правилам фонетики и грамматики добавились стандартные служебные символы, добавляемые компьютерной программой, пробелы, возврат каретки и перевод строки, различные заголовки, информация о версии, Copyright, информация о пользователе и многое, многое другое. Возьмите текстовой файл, набранный в Microsoft Word, и посмотрите, сколько места в нем занимает сам текст, а сколько — служебные символы. В совокупности избыточность компьютерной информации оказалась во много раз выше избыточности простой разговорной речи.

Продолжим рассказ Дэвида Кана о Шенноне. "С чего начинается криптоанализ? При исправлении ошибки все избыточные элементы, используемые для правки, лежат в готовом виде на поверхности. В криптограмме все наоборот — они незаметны. Криптоаналитик производит подсчет частот букв криптограммы и соотносит полученные результаты с известными частотами букв предполагаемого языка, на котором записан открытый текст. Откуда у криптоаналитика уверенность в том, что частоты букв открытого текста данной криптограммы примерно совпадают с частотами эталонного открытого текста? Разве не может это соответствие нарушиться из-за различий в словарном запасе корреспондентов и в темах их переписки? Нет, не может, ибо избыточные элементы языка превалируют над остальными. Сила ума Шеннона, его огромный вклад в теорию шифровального дела выразились в открытии избыточности как основы криптоанализа. Шеннон первым сумел объяснить постоянство частот встречаемости букв, а тем самым и такое зависящее от него явление, как криптоанализ, дав возможность глубоко понять процесс аналитического вскрытия шифров".

Применение идей Шеннона на практике мы уже видели в прошлой главе, на примере Русской армии 1914 года.

А теперь — немного математики, взятой из учебника "Лекции по теории информации", МГУ, раздел "Количество информации в дискретном сообщении. Энтропия".

"Предположим, что источник сообщений может в каждый момент времени случайным образом принять одно из конечного множества возможных состояний. Такой источник называют дискретным источником сообщений. При этом принято говорить, что различные состояния реализуются вследствие выбора их источника. Каждому состоянию источника U ставится в соответствие условное обозначение в виде знака. Совокупность знаков u_1, u_2, ..., u_i, ..., u_N, соответствующих всем N возможным состояниям источника, называют его алфавитом, а количество состояний N — объемом алфавита. Формирование таким источником сообщений сводится к выбору им некоторого состояния u_i и выдачи соответствующего знака. Таким образом, под элементарным дискретным сообщением будем понимать символ u_i, выдаваемый источником, при этом в течение некоторого времени T источник может выдать дискретное сообщение в виде последовательности элементарных дискретных сообщений, представляющей собой набор символов u_i (например, u_5, u_1, u_3), каждый из которых имеет длительность t_i секунд. В общем случае необязательно одинаковую для различных i. Такая модель источника сообщений соответствует реальной ситуации, имеющей место в телеграфии ($t_i \neq \text{const}$) и передаче данных ($t_i = \text{const}$). Отдельные состояния источника могут выбираться им чаще, другие реже. Поэтому в общем случае он хранится дискретным ансамблем U, т. е. полной совокупностью состояний с вероятностями их появления, составляющими в сумме 1.

$$U = \begin{pmatrix} u_1 & u_2 & \dots & u_i & \dots & u_N \\ P(u_1) & P(u_2) & \dots & P(u_i) & \dots & P(u_N) \end{pmatrix}, \qquad \sum_{i=1}^{N} P(u_i) = 1, \tag{1.1}$$

где $P(u_i)$ — это вероятность выбора состояния u_i источником сообщений. При выдаче источником сообщений в виде последовательности элементарных дискретных сообщений, полным вероятностным описанием является вероятность совместного появления набора различных символов u_i в момент t_1, t_2, ..., t_n, где n — длина последовательности

$$P(u_i^{t_1}, u_j^{t_2}, \dots, u_k^{t_i}, \dots, u_l^{t_n}).$$[1]

[1] В нашем случае эта модель описывает множество всевозможных открытых текстов, которые могут быть затем подвергнуты зашифровыванию.

Располагая такими сведениями об источнике, можно вычислить вероятность любого отрезка сообщения длиной меньше n.[1]

В каждом элементарном сообщении содержится для его получателя определенная информация: совокупность сведений о состоянии дискретного источника сообщения. Определяя количественную меру этой информации, мы совершенно не будем учитывать ее смыслового содержания, также ее значения для конкретного получателя. Очевидно, что при отсутствии сведений о состоянии источника имеется неопределенность относительно того, какое сообщение u_i из числа возможных им выбрано, а при наличии этих сведений данная неопределенность полностью исчезает. Естественно количество информации, содержащейся в дискретном сообщении, измерять величиной исчезнувшей неопределенности. Введем меру этой неопределенности, которую можно рассматривать и как меру количественной информации. Мера должна удовлетворять ряду естественных условий, одним из них является необходимость ее монотонного возрастания с увеличением возможности выбора, т. е. объема алфавита источника N. Кроме того, желательно, чтобы вводимая мера обладала *свойством аддитивности*, заключающемся в следующем: если 2 независимых источника с объемами алфавита N и M рассматривать как один источник, одновременно реализующий пары состояний n_i и m_i, то в соответствии с принципом аддитивности полагают, что неопределенность объединенного источника равна сумме неопределенностей исходных источников. Поскольку объем алфавита объединенного источника равен NM, то искомая функция при равной вероятности состояний источников должна удовлетворять условию $f(NM) = f(N) + f(M)$. Можно математически строго показать, что единственной функцией, при перемножении аргументов которой значение функций складываются, является логарифмическая функция. Поэтому перечисленные требования выполняются, если в качестве меры неопределенности источника с равновероятными состояниями и характеризующего его ансамбля U принять логарифм объема алфавита источника

$$H(u) = \log N.$$

Легко видеть, что:

- с ростом N величина $H(U)$ монотонно возрастает;
- в случае если объем алфавита источника N равен 1, т. е. когда неопределенность отсутствует,

$$H(u) = \log 1 = 0;$$

- величина $H(U)$ обладает свойством аддитивности, поскольку

$$\log(NM) = \log(N) + \log(M).$$

[1] То есть вероятность любого открытого текста длины, не превосходящей n.

Впервые данная мера была предложена Хартли в 1928 г. Основание логарифма не имеет принципиального значения и определяет только масштаб или единицу количества информации. Чаще всего в качестве основания используют число 2, при этом единица количества информации называется двоичной единицей или битом и представляет собой информацию, содержащуюся в одном дискретном сообщении источника равновероятных сообщений с объемом алфавита равным двум. При выборе основания логарифма равным 10 получаем десятичную единицу, называемую дитом. Иногда используют натуральную единицу количества информации, называемую натом, при этом основание логарифма равно $e \approx 2{,}7$. Рассматриваемая мера количества информации может иметь лишь ограниченное применение, поскольку предполагает равную вероятность выбора источником любого из возможных его состояний.[1]

В более общем случае, когда вероятности различных состояний источника не одинаковы, степень неопределенности конкретного состояния зависит не только от объема алфавита источника, но и от вероятности этого состояния. В такой ситуации количество информации, содержащееся в одном дискретном сообщении u_k, целесообразно определить как функцию вероятности появления этого сообщения $P(u_k)$ и характеризовать величиной

$$i(u_k) = -\log P(u_k) = \log \frac{1}{P(u_k)} \qquad (1.2)$$

Теперь количество информации, содержащееся в дискретном сообщении, зависит от степени неожиданности этого сообщения, характеризуемой вероятностью его появления. Количество информации в сообщении тем больше, чем оно более неожиданно. Если источник выдает последовательность зависимых между собой элементарных сообщений, то наличие предшествующих сообщений может изменить вероятность последующего а, следовательно, и количество информации в нем.[2]

Оно должно определяться по условной вероятности $P(u_k/u_{k-1}, u_{k-2}, ...)$ выдачи сообщений u_k при известных предшествующих сообщениях $u_{k-1}, u_{k-2}, ...$, тогда количество информации

$$i(u_k/u_{k-1}, u_{k-2}, ...) = -\log P(u_k/u_{k-1}, u_{k-2}, ...) \qquad (1.3)$$

Определения (1.2) и (1.3) количества информации являются случайной величиной, поскольку сами сообщения являются случайными. Его распределение вероятностей, определяемое распределением вероятностей сообщений

---

[1] В нашем случае открытые тексты выбираются, как правило, неравновероятно.

[2] Именно так и обстоит дело, к примеру, в русском, да и в любом другом разговорном языке. Например, за буквой 'ы' не может следовать 'ь'. Наличие таких запретов помогает криптоаналитику вскрыть шифр.

в данном ансамбле для цифровой характеристики всего ансамбля или источника сообщения, используется для математического ожидания количества информации в отдельных сообщениях, называемого *энтропией*.

$$H(U) = M\left\{\log\frac{1}{P(u_i)}\right\} = \sum_{i=1}^{N} P(u_i)\cdot\log\left(\frac{1}{P(u_i)}\right) \qquad (1.4)$$

Чем больше энтропия источника, тем больше степень неожиданности выдаваемых им сообщений в среднем, т. е. тем более неопределенным является ожидание сообщений. Впервые мера (1.4) была преложена Клодом Шенноном в его фундаментальной работе "Математические основы теории связи", опубликованной в 1948 г., в которой были заложены основы современной теории информации".

Итак, Шеннон впервые вывел точную формулу для меры неопределенности открытого текста. И с точки зрения криптографии сделал фундаментальный вывод: обосновал абсолютно стойкий шифр, т. е. такой шифр, который никто и никогда не сможет вскрыть. А именно:

Определение 2.1

Шифр является абсолютно стойким, если энтропия открытого текста при условии известного шифртекста равна безусловной энтропии открытого текста.

Попросту говоря, если наличие шифртекста не дает криптоаналитику никакой новой информации об открытом тексте. Конечно же, этому условию не удовлетворяет простая замена: в ней статистика шифртекста — это просто переставленная статистика открытого текста. По шифртексту криптоаналитик в этом случае сразу же определяет места появления наиболее часто встречающихся символов в открытом тексте, т. е. сразу же получает огромную дополнительную информацию об открытом тексте.

Теперь осталось сделать последний шаг — построить пример абсолютно стойкого шифра. И это, имея формулы Шеннона, описывающие энтропию, оказалось возможно.

Определение 2.2

Если шифр получается путем наложения на открытый текст случайной и равновероятной гаммы, то такой шифр является абсолютно стойким.

Из теории вероятностей известно, что при сложении двух случайных величин, если одна из них является случайной и равновероятной, то сумма, независимо от распределения другой величины, также будет случайной и равновероятной. Если при гаммировании произвольный открытый текст (неравновероятная случайная величина) складывается со случайной и равновероятной гаммой, то шифртекст будет случайным и равновероятным.

Подставляя в формулу (1.4) значение $P(u_i) = 1/N$ для всех i, получаем максимальное значение $H(U) = \log(N)$.

Помните слова Дэвида Кана из прошлой главы о том, что "одноразовые шифрблокноты обеспечивают надежную защиту для сообщений российских разведчиков, военных, дипломатов и работников тайной политической полиции"? Это и есть пример абсолютно стойких шифров. В них пишется случайная и равновероятная гамма, и вскрыть их, не имея второго экземпляра блокнота, теоретически невозможно.

Однако в большинстве практических случаев одноразовые шифрблокноты, т. е. таблички со случайной гаммой, сразу после наложения которой они уничтожаются, неудобны. Запас гаммы ограничен, для предотвращения компрометации нужно соблюдать очень строгие правила при хранении неиспользованных блокнотов. При защите компьютерной информации применение шифрблокнотов возможно только в уж очень экзотических ситуациях, например, при смене секретных ключей для рассылки новых ключей по электронной почте. В повседневной, реальной жизни используются программные генераторы гаммы. А их уже необходимо анализировать и не только на предмет того, насколько случайную и равновероятную гамму они вырабатывают, но и с точки зрения возможности определения начальных параметров генератора по какому-то отрезку гаммы или, крайний случай, по всей гамме. Такие начальные параметры и являются секретным ключом системы шифрования.

Насколько правомерна такая задача? В каких случаях потенциальному злоумышленнику может быть известен отрезок гаммы? Давайте не забывать, что в открытом тексте могут быть достаточно длинные стандарты. Криптоаналитиков часто выручают выражения вроде: "В ответ на Вашу телеграмму от", "На Ваш исходящий № от ", "Уважаемый господин ", "С уважением" и т. п. Они позволяют определить некоторые отрезки гаммы, и если по этой информации оказывается возможным вычислить начальные параметры генератора гаммы, то сразу же вскрывается и весь остальной шифрованный текст. Например, если в качестве генератора гаммы используется линейная рекуррентная последовательность вида

$$x_{i+n} = x_i + x_{i+1} + \ldots + x_{i+n-1},$$

а секретным ключом являются первые n значений, то если хотя бы в одном месте криптоаналитику будет известно n знаков гаммы, решив систему линейных уравнений он без труда вычислит секретный ключ и весь открытый текст.

Более подробно о требованиях, предъявляемых к генераторам гаммы, мы поговорим в следующей главе, а сейчас, раз мы говорим о стойкости шифра и уже определили понятие абсолютной стойкости, подумаем о том, что следует понимать под гарантированной стойкостью.

Любой программный генератор гаммы в принципе может быть вскрыт. Число начальных параметров ограничено, поэтому всегда можно предполагать, что потенциальный злоумышленник попробует их все перебрать. Но представим себе, что он попробует сделать это, как ранее в приведенном примере, когда каждый элемент ключа — байт, а $n = 17$. Количество всевозможных вариантов перебора составит

$$(2^8)^{17} = 2^{136} = 10^{40}.$$

(Здесь и дальше мы будем пользоваться распространенным у криптографов округлением $2^{10} \approx 10^3$.)

Если ключ (17 начальных значений) выбирался случайно и равновероятно, то по теории вероятности в среднем половину из этого (10^{40}) числа вариантов надо будет опробовать, пока не попадется истинный вариант. Говорить о половине при таких значениях несерьезно, это 5×10^{39}, т. е. почти те же порядки, при округлении мы огрубили эту величину гораздо больше. Как можно наглядно представить себе эту величину?

Производительность современного компьютера примем за 10^{10} простейших операций в секунду. Тогда за 1 час = 3600 сек компьютер выполнит $3{,}6 \times 10^{13}$ простейших операций, за сутки — $24 \times 3{,}6 \times 10^{13} \approx 10^{15}$, за год — $3{,}6 \times 10^{17}$, округляем до 10^{18}. Для перебора 10^{40} значений такому компьютеру потребуется 10^{22} лет. Ясно, что это нереальная задача. Предположим, что для перебора используется не один, а несколько (например, 10^6 — миллион) таких компьютеров. Им всем, работая в параллельном режиме, для перебора потребуется 10^{16} лет. Даже с учетом постоянно растущей производительности компьютеров, ясно, что задача перебора всевозможных значений имеет разумные пределы. Отодвинем ее до неразумных (с учетом постоянно возникающих сообщений о возможности появления сверхпроводниковых, квантовых, биологических и прочих новейших технологий) и примем величину в 10^{100} как гарантированную оценку стойкости.

Сложнее с другой стороной дела. В приведенном выше примере мы видели, что, несмотря на высокую оценку тотального перебора, шифр элементарно вскрывался за счет плохо выбранного генератора гаммы. А искусство выбора и всестороннего изучения генератора гаммы и есть искусство математика-криптографа. Об этом — в следующей главе.

Глава 3

Генератор гаммы

Поскольку качественный генератор гаммы является неотъемлемой частью шифрсистемы, то давайте поподробнее рассмотрим, каким требованиям он должен удовлетворять.

Шеннон писал: "С криптографической точки зрения секретная система почти тождественна системе связи при наличии шума". В теории связи термин "шум" имеет особое значение. Под шумом подразумевается любая помеха, создающая ошибки при передаче по каналу связи. Шеннон исходит из того, что шум схож с наложением гаммы. "Основное различие между ними заключается, во-первых, в том, что преобразование при помощи шифра имеет обычно более сложный характер, чем возникающее за счет шума в канале; во-вторых, в том, что ключ в секретной системе выбирается из конечного множества, тогда как шум обычно вносится в канал постоянно и выбирается из бесконечного множества".

Понятие "конечное — бесконечное множество" весьма относительно. Конечное множество, состоящее из 10^{100} элементов, с точки зрения практических задач организации его перебора, вполне можно считать бесконечным. И задача построения качественного генератора гаммы сводится как раз к тому, чтобы накладываемая на открытый текст гамма по своим характеристикам напоминала бы "белый шум", т. е. шум, который полностью заглушает все осмысленные параметры открытого текста. А принципиальное отличие от традиционного шума в том, что генератор гаммы должен уметь повторять с точностью до единого знака этот шум, с тем чтобы была возможность расшифровать зашифрованный ранее текст. А это, как правило, невозможно осуществить без использования такого традиционного математического аппарата, как рекуррентные последовательности, т. е. такие, в которых каждый последующий знак вычисляется как некоторая постоянная функция от n предыдущих. В примере из прошлой главы в роли такой функции выступала сумма n предыдущих знаков. На этом примере мы смогли убедиться, что не любая рекуррентная последовательность пригодна для

использования в качестве генератора гаммы. Давайте рассмотрим этот и некоторые другие примеры несколько подробнее.

Поскольку в этой книге речь, в основном, идет о защите компьютерной информации, то в качестве алфавита открытого и шифрованного текста у нас часто (но не всегда) будет выступать вся таблица ASCII-символов, т. е. множество всевозможных целых неотрицательных чисел от 0 до 255 с определенными на нем операциями сложения и вычитания по модулю 256. Такую конструкцию в математике еще называют кольцом вычетов по модулю 256 и обозначают как $Z/256$.[1]

$$x_{i+n} = x_i + x_{i+1} + \ldots + x_{i+n-1}$$

Предположим, что в качестве начальных значений этого рекуррентного соотношения выбраны все нули. Тогда совершенно очевидно, что вся вырабатываемая гамма также будет состоять из одних нулей, т. е. получается, что у такого генератора гаммы есть потенциально опасный ключ. Такую точку в теории рекуррентных последовательностей называют изолированной, а соответствующий ключ можно назвать критическим.

Предположим теперь, что все начальные значения — нечетные числа и n — нечетно. Тогда, очевидно, все значения гаммы наложения также будут нечетными. После наложения такой гаммы на открытый текст все четные знаки открытого текста станут нечетными, а нечетные — наоборот, четными. По четности знаков шифртекста мы сразу же определяем четность знаков открытого текста, и выработанная гамма заведомо не удовлетворяет условиям Шеннона. Внимательному читателю предлагаю самому представить, что будет в том случае, когда все начальные значения рекуррентного соотношения — четные (независимо от четности n).

Аналогично, в случае, если все начальные значения гаммы кратны 3, 5, 7 либо любому другому простому числу, то все знаки вырабатываемой гаммы также будут кратны этому числу.

Таким образом, среди всевозможных ключей в таком генераторе заведомо есть криптографически плохие, т. е. вырабатываемая ими гамма не удовлетворяет условиям Шеннона.

Предположим теперь, что $n = 5$ и начальные значения x_0, x_1, x_2, x_3, x_4 равны:

$x_0 = 32$;

$x_1 = 32 + 128$;

---

[1] Замечание "не всегда" связано с тем, что иногда приходится сталкиваться с проблемой отображения информации на экран, например, при работе с текстовым редактором. Там среди всевозможных ASCII-символов приходится отбирать только те, которые можно отобразить на экране и которые не вызовут каких-то конфликтов при работе программного обеспечения. Я предпочитаю в качестве таких символов использовать латинские буквы и цифры.

$x_2 = 32;$

$x_3 = 32 + 128;$

$x_4 = 32.$

Тогда

$x_5 = 32 + 128;$

$x_6 = 32;$

$x_7 = 32 + 128;$

$x_8 = 32;$

и т. д.

Здесь мы столкнулись с таким криптографически важным понятием, как повторяемость, и частным случаем повторяемости — периодичностью знаков гаммы. Этот пример сродни тому случаю, когда программист, строя собственную систему защиты, использует периодически повторяющийся пароль для наложения его на открытый текст, а это, как правило, первое, что приходит в голову человеку, неискушенному в вопросах криптографии. О повторяемости гаммы мы поговорим позже, сейчас же отметим, что при выборе генератора гаммы один из первых вопросов, который должен задать себе разработчик, — как избежать описанных ранее и подобных им казусов и обосновать какие-то свойства генератора гаммы? Такое обоснование — это весьма трудоемкий и сложный процесс, требующий знаний из алгебры, теории полей, теории подстановок, теории линейных рекуррентных последовательностей, а подчас просто интуиции и опыта криптографа. Описать все возможности даже в первом приближении в рамках этой книги не представляется возможным, да я и не ставил себе такой задачи. Но некоторые вещи можно обсудить.

1. Если посмотреть на пример, то легко увидеть, что одной из его особенностей является неправильный выбор ключевых параметров. В качестве секретного ключа мы выбрали начальное заполнение, от которого затем раскручиваются все остальные знаки. В процессе выработки очередного знака в нем принимают участие только n предыдущих. Поэтому если, в силу наличия стандарта в тексте, мы сумеем определить n подряд идущих знаков гаммы, то все следующие за ними будут вычисляться автоматически. В этом примере ключевые параметры хотя и принимают участие в выработке каждого знака гаммы, но это участие зависит от предыдущих знаков выработанной гаммы, что и привело в конечном счете к возможности вычисления последующих знаков гаммы по предыдущим. Поэтому ключевые параметры должны принимать участие в выработке каждого знака гаммы *независимо* от выработанных предыдущих знаков.
2. Криптоаналитику значительно облегчило работу то обстоятельство, что выбранная рекуррентная последовательность оказалась *линейной*. Легко

составляется и решается система уравнений, уравнения дают хорошие (с точки зрения криптоанализа) результаты при сложении или вычитании различных знаков гаммы, например, такие:

$$x_{i+n} - x_{i+n-1} = x_i + x_{i+1} + \ldots + x_{i+n-2}$$

$$x_{i+n+1} - x_{i+n} = x_{i+1} + x_{i+2} + \ldots + x_{i+n-1} =$$

$$= x_{i+n} - x_{i+n-1} - x_i + x_{i+n-1} = x_{i+n} - x_i.$$

Линейность надо убрать. Но представим, например, что, убрав линейность, мы выбрали функцию

$$x_{i+n} = x_i \times x_{i+1} \times x_{i+n-1}.$$

Это означает, что любое начальное заполнение, в котором есть хотя бы один ноль, будет давать полностью нулевую гамму. И не только это. Если, например, $n = 8$, то начальное заполнение, в котором все значения равны 2, будет давать тоже полностью нулевую гамму, так как мы выполняем все операции по модулю 256.

В алгебре есть понятие *подстановки*. В случае модуля 256 это просто таблица со всеми значениями от 0 до 255, но как-то перемешанными. Всего можно построить 256! всевозможных подстановок по модулю 256. Множество всевозможных таких подстановок принято называть *симметрической группой* S_{256}. Давайте посмотрим, как изменится линейное уравнение, если к нему добавить некоторую подстановку π

$$x_{i+n} = \pi(x_i + x_{i+1} + \ldots + x_{i+n-1}).$$

Нетрудно видеть, что вычитание различных знаков гаммы при этом уже не будет давать такого простого результата, как в случае, когда функция была линейной. Несомненно, что использование подстановки значительно усложнит задачу криптоаналитику. В то же время скорость реализации такого преобразования практически не уменьшится, поскольку замена значения по подстановке — это одна операция обращения к памяти.

3. А что если в приведенном выше примере сделать ключевым параметром подстановку? На предварительном этапе, один раз перед сеансом выработки гаммы, мы вычисляем случайную подстановку в зависимости от секретного ключа. На скорости реализации гаммы это практически не скажется. Зато задача анализа такого генератора, при неизвестной подстановке, намного усложнится.

4. Этот генератор работает достаточно быстро. А что если использовать не все знаки подряд, а через несколько? Это еще больше усложнит работу криптоаналитика.

Это — основные идеи. Конкретная реализация зависит от фантазии разработчика, его вдумчивости, опыта и целей. Однако могу заверить, что таким

путем можно построить криптографически качественный и высокоскоростной генератор гаммы, значительно превосходящий по скорости выработки гаммы алгоритмы типа DES и ГОСТ.

Теперь — о другой страшной опасности: перекрытиях гаммы.

Что такое наложение гаммы на открытый текст? Это, обычно, процесс познакового сложения открытого текста и гаммы. Например, если открытый текст обозначить как a_1, a_2, ..., a_n, гамму — x_1, x_2, ..., x_n, то шифртекстом будет последовательность $s_1 = a_1 + x_1$, $s_2 = a_2 + x_2$, ..., $s_n = a_n + x_n$. Предположим, что с некоторого места k гамма начала повторяться, т. е. $x_1 = x_k$, $x_2 = x_{k+1}$. Отсюда следует, что

$$s_1 - s_k = a_1 - a_k,$$

$$s_2 - s_{k+1} = a_2 - a_{k+1},$$

...

Таким образом, разность знаков шифрованного текста равна разности знаков открытого текста. Появляется возможность *бесключевого чтения*: предполагая некоторое вероятное слово или фразу в открытом тексте, определяем открытый текст через k знаков и по его читаемости принимаем решение, верно наше предположение или нет. Критериев читаемости или нечитаемости текста может быть множество, например, если речь идет о русском языке и в вычисленном тексте встретилась запретная для русского языка пара ЫЬ, то такой текст признается ложным.

Теперь нетрудно заметить, что точно такой же эффект будет и в том случае, если, использовав один раз генератор гаммы и осуществив с его помощью зашифровывание открытого текста, мы попытаемся точно такой же гаммой зашифровать другой открытый текст. Начинают срабатывать те же методы бесключевого чтения. Сам генератор гаммы может быть сколь угодно хорошим, вырабатываемая им гамма удовлетворять всем условиям Шеннона, но для определения открытого текста по шифрованному нам не потребуется знание гаммы. Предполагая наличие некоторого стандарта в одном тексте, проверяем читаемость вычисленного другого открытого текста и принимаем решение об истинности или ложности сделанного предположения.

Отсюда следует один из наиболее фундаментальных принципов построения качественного генератора гаммы.

Определение 3.1

Всякий раз гамма, вырабатываемая генератором гаммы, должна быть новой, отличной от всех предыдущих гамм, выработанных этим генератором.

Что это означает на практике? Всякий раз, перед началом выработки гаммы наложения, необходимо в параметры, влияющие на выработку гаммы, добавить некоторый дополнительный параметр, называемый в разной литературе

по-разному: разовым ключом, маркантом, ключом на телеграмму и т. п. Мне больше нравится слово *маркант*, поскольку этот параметр, как таковой, чаще всего *ключом не является*. Вы должны сообщить его тому, кто будет расшифровывать текст, чтобы он смог повторить, имея секретный ключ, точно такую же гамму. Маркант иногда просто дописывается *в открытом виде* в шифрованное сообщение и пересылается вместе с шифртекстом адресату. А алгоритм выработки гаммы надо строить с тем учетом, что маркант известен потенциальному злоумышленнику. Маркант чаще всего строится по текущему моменту времени, в который осуществляется зашифровывание, хотя возможны и другие варианты, например, когда сам пользователь, нажимая на клавиши, вырабатывает случайный маркант. Единственное требование: маркант практически никогда не должен повторяться. Повторение марканта равносильно повторению гаммы и бесключевому чтению.

Вот такие основные правила должен соблюдать разработчик, создавая собственный генератор гаммы. Но и это еще не все.

Предположим, что мы зашифровываем банковскую информацию. Например, сумму в 1 000 000 рублей в банковском платежном поручении. И в отличие от предыдущих рассуждений, будем предполагать, что открытый и шифрованный тексты состоят только из цифр, а сложение осуществляется по модулю 10.

Пусть, например, для зашифровывания этой суммы наш генератор гаммы выработал последовательность 5017329. Тогда шифртекст — сумма открытого текста и гаммы — будет 6017329 и этот шифртекст был послан в канал связи, причем такой, к которому потенциальный злоумышленник имеет доступ. Ключа к шифру злоумышленник не знает и при правильно построенном генераторе гаммы вычислить не может. Но это ему в некоторых случаях и не нужно. Предположим, что злоумышленник имеет возможность подменять шифртекст в канале связи (например, при пересылке платежного поручения по телеграфу, почти как по Пушкину: "и в суму его пустую суют грамоту другую"). Воспользовавшись этой возможностью, злоумышленник подменит шифртекст 6017329 на 7017329.

Что произойдет на приемном конце? Получив платежное поручение с зашифрованной суммой — 7017329 — операционистка, обладая ключом к выработке гаммы, выработает ее: 5017329, и произведя расшифрование, т. е. вычитание гаммы из шифртекста, получит сумму 2 000 000, которую и зачислит на счет получателя. Нетрудно предположить, что получатель и будет тем злоумышленником, который, подменив шифртекст, не зная ключа к шифру, тем не менее сумел получить таким образом лишний миллион рублей.

Следовательно, при использовании шифра гаммирования подмена некоторых знаков в шифртексте может быть не обнаружена при расшифровании. В криптографии свойство шифра обнаруживать подмену некоторых знаков

в шифрованном тексте называют *имитостойкостью*. И разработчик должен помнить, что шифр гаммирования не обладает имитостойкостью.

Начало сентября 1992 года. Старое здание ЦБ на Неглинке. Комната-пенал, два стола вдоль стены, никаких излишеств.

— Нам нужна Ваша помощь.

Это говорит, вставая из-за стола у окна, высокий стройный человек с седой и густой шевелюрой. Спокойный, уравновешенный, без начальственных привычек. Наверное, раньше был инженером, технарем. А в ЦБ я, действующий офицер ФАПСИ, 35-летний подполковник, без ведома большого начальства. В Конторе какие-то очень уж непонятные времена начались. Сразу после августовских событий 91-го года, как только Ельцин победил, буквально на следующий день нашего начальника Главка завалили рапортами об увольнении. В моем отделении едва ли не треть ребят подали рапорта. И все — молодые, не закостенелые, кому до смерти надоели эти постоянные разговоры ни о чем в курилках да игры на компьютере. Те, кто хочет еще как-то двигаться, а не просто ждать пенсии. Визировал их, как зам. начальника отделения, без сожаления. Сам тоже твердо решил: протяну как-нибудь до 1994-го года, там будет 20 лет выслуги, пенсия, и — сразу на свободу. Начальники тоже как-то притихли. В январе 1992 года, на ежегодном отчете нашего отделения, прямо сказали: "Учитесь зарабатывать деньги, развивайте коммерцию". Позаключали разных договоров. И вот в начале 1992 года — новая метла, новый самый большой начальник. Прозвали его "папой". В мае — очередная крутая смена курса. "Всякую коммерческую деятельность запретить, все договоры разорвать!" Ну, разорвать так разорвать, офицеру что прикажут — то и сделает. Но на душе как-то муторно. А когда будет очередной прогиб Генеральной линии? Через полгода, год, два? И в какую сторону?

— Надо защитить народные деньги.

Ну, это по моей части! Система электронной подписи к тому времени уже готова, и даже более того, готова комплексная программа "Криптоцентр", в которой наворочено все, что душе угодно: подпись, шифрование на индивидуальном секретном ключе, шифрование с использованием открытых ключей, журнал учета, помощь. С собой экзотический в те времена Notebook. Ставлю его на стол, показываю "Криптоцентр". Все в диковинку, смотрят внимательно, но что-то не особенно заинтересованно.

— Понимаете, это хорошо, но нам сейчас нужно не то. У нас около 2000 РКЦ, компьютеров почти нигде нет. Связь, в основном, по почте и телеграфу. А в Сибири есть и такие, до которых 3 дня на лодке надо плыть.

"Широка страна моя родная", что и говорить. Да, наверное они правы: "Криптоцентр" тут не всем подойдет.

— Нам бы как-нибудь ваш калькулятор приспособить и полгодика продержаться. А там что-нибудь придумаем.

Калькулятор — это "Электроника МК 85 С". В его "математике" не ГОСТ-монстр, а человеческий регистр сдвига. С ГОСТом он бы помер еще до рождения — слишком медленный этот брат-близнец американского DESa. Сколько людей написали диссертации на регистрах сдвига! Вот только реализация — советская. На каких трех китах держался мир социализма? На русской экономии и бережливости, на польском трудолюбии и на монгольской электронике.

— Авизовка — это платежное сообщение, которое затем передают по почте или обычному телеграфу. Нам бы туда добавить каких-нибудь проверочных знаков, которые будут на вашем калькуляторе вычисляться, но не больше 10, больше наши технологии не позволяют.

Чуть не падаю со стула. Не больше 10! Да там один маркант — 10 знаков! Сказали тоже — проверочная комбинация. Шифр гаммирования не является имитостойким, это нам еще на 4-м курсе Высшей школы твердо втолковали.

— У нас каждая операционистка в день может обрабатывать до трехсот авизовок.

Однако! Почище, чем в армии. Там, дай бог, десяток команд, да и то не каждый день. В общем, все ясно! Глухо дело. Не подойдет для них калькулятор.

— Ну, Вы подумаете, как можно нам помочь?

Вежливо прощаемся. Чисто символически, обещаю подумать. Все ясно — чего там думать! Не подойдет им калькулятор. 10 знаков! Тут электронная подпись нужна. Хотя в 10 знаков никакая электронная подпись не влезет — для нормальной подписи около 100 знаков надо. Нечего и голову ломать, напрасно терять время — пусть пишут ТЗ, открывают НИР, ОКР, года через 2 может чего от "папы" и получат. А я человек маленький, мне какое дело до их проблем!

"Бакс" уже за 200. Интересно, а его рост — это ограниченная функция? Если да, то какой величиной? За пару месяцев — почти вдвое! Если за каждую пару месяцев — вдвое, то к Новому году будет за 500. Это еще недавно была моя месячная зарплата в Конторе.

В "АиФ" статья Дунаева, замминистра МВД. Примерно треть бюджета украдена по фальшивым авизо и чекам "Россия". Воруют!

"Какой-то червяк начинает глодать". Ведь это же ЦБ, огромная сеть, в прямом смысле вся страна. Вот было бы внедрение диссертации на всю страну! При защите справку о внедрении, естественно, из пальца пришлось высасывать. Новое направление. Результаты. Это вам не DES в стандарты протаскивать, бумажки писать. Тут одни логарифмические подстановки чего стоили — построен предельный случай. Эту красоту бы в ЦБ! Но шифр гаммирования не является имитостойким, может те, кто делает фальшивые авизовки, этого и не знают, но в Конторе-то уж — наверняка! И там не простят ошибки!

А может взять какую-нибудь кодовую книгу, как в армии, что-то по ней закодировать, потом перешифровать на калькуляторе. Вот смеху-то будет. Назад, в Мазурские болота! ЦБ взяло на вооружение шифр русской армии 14-го года! Бред какой-то. А видно, им это очень надо. И срочно. Иначе пошли бы к "папе".

А что вообще-то есть там, в этом калькуляторе? Шифр гаммирования. Вводишь открытый текст, 10 раз жмешь на кнопку-генератор марканта, получаешь шифртекст. В цифровых пятизначных группах. Да если на каждую авизовку девочка будет по 10 раз нажимать кнопку марканта, 300 авизовок в день, то это 3000 нажатий на одну советскую кнопку в день! Этот день уж точно будет последним для этой кнопки.

А сейчас там, в этом ЦБ, есть что-нибудь? В смысле защиты? А как же! Еще со времен не то Николая II, не то Иосифа Грозного. Маскировка. Примерно такая же, что слово "доллар" словом "бакс" замаскировать. Никто ни в жизни не догадается! Вась, что мне всю ночь лимон с ножками снился? А, так вот кто вчера мою канарейку в чай выжал. Вот взять и перешифровать на калькуляторе эти "лимоны" и "канарейки". Круто! Что я знаю о шифрах? (Это так можно число π запомнить: по количеству букв каждого слова в этой фразе — 3,1416.) В них нежелательны стандарты. Предполагаешь некоторое слово в открытом тексте, определяешь по нему гамму, под эту гамму подставляешь, например, вместо лимона арбуз — и пошла авизовка, что шифровали, что нет, все едино. Засмеют, по крайней мере. А то и побьют. Шифр гаммирования не является имитостойким.

Ну ладно. Рассуждаем логически. Нужно использовать классические способы электронной подписи. Вычисляем хэш-функцию. Легко сказать — вычисляем хэш-функцию! А как? Руками? Бабушка из сибирской Панкрушихи руками вычисляет хэш-функцию. Заточила поострее карандашик, взяла листочек бумаги, обматерила того, кто все это придумал, и — пошла писать губерния! Через полчаса может что и вычислит. С ошибками. И все платежи враз встанут, как наш бронепоезд, в тупик на запасном пути. Калькулятор-то ведь не приспособлен ни для какой хэш-функции. А потом, что с ней делать, с этой хэш-функцией? Шифровать ее? Так ведь супостат быстрее бабушки сможет точно такую же функцию вычислить. И подставить свою вместо бабушкиной под гамму. Все. Нет никакой защиты. Ну нет имитостойкости у шифра гаммирования!

Совершенно тупиковая задача. И если б можно было ее на компьютере сделать — нет проблем! Придумать алгоритм выработки кода, подтверждающего подлинность авизовки, — милое дело. В хэш-функцию ввел ключевой параметр, хэш-функция вся просчитана, запрограммировал — готово! Но в Панкрушихе и Ребрихе нет компьютеров и долго еще не будет. Да и свет там, бывает, отключают. И обучить бабушку работе на компьютере Да, калькулятор им больше подойдет. И наштамповали их целую кучу для армии — девать теперь некуда. Будь она неладна, эта имитостойкость!

Но, однако, запросы у них! 10 знаков! Один мой, не содержащий ровно никакой полезной для ЦБ информации, маркант — это уже 10 знаков. Ну, допустим, сторгуемся на 20. Из них полезных — только половина. И это все случайные знаки, каждый из них надо будет руками набирать, несколько раз внимательно проверять в каждой из 300 авизовок каждый день. И половина этой работы — совершенно лишняя, из-за прихоти "яйцеголовых" криптографов. Попробуй, скажи девушке, у которой к концу рабочего дня круги в глазах от этой цифири, что половину всей работы она сделала впустую, только потому, чтобы избежать каких-то там перекрытий гаммы! Сразу запустит тебе в голову этим калькулятором. И правильно сделает! Думать надо было лучше!

Надо отказываться. Решено. Зачем с ЦБ связываться? Да еще напрямую, в обход начальства. Оно, наверное, какие-то свои виды на эту работу имеет. Проведут совещание, составят план, напишут ТЗ, откроют финансирование. Для защиты народных денег мы попросим у народа немного денег. А крадут по-черному. Бывает, что кражи раскрывают. Вот бежал один, опаздывал на самолет, споткнулся, упал, раскрылся чемодан, полный долларов. Обналичил фальшивое авизо. Вот так и раскрывают.

В ЦБ рассказывают случай. Пришла авизовка на крупную сумму. Девочка-операционистка сидит и плачет. Что делать? Зачислишь — а вдруг окажется, что авизо фальшивое? Виновата, что недосмотрела. Не зачислишь — а вдруг авизо подлинное? Задержка платежа. Опять виновата. Сиди и плачь. Тут точный критерий нужен. Код подтверждения достоверности. Совпал — зачисляй, не совпал — до свидания, имитостойкость на пару с маркантом!

Все! А жалко отказываться. Другой такой возможности в моей жизни уж точно не будет. Но нужно какое-то нетривиальное решение. Изюминка. Все традиционные методы не годятся. А как его найти, это то, сам не знаю что? Логическим путем? Если бы! Есть такое понятие — "черный ящик". В него подаешь на вход начальные условия, он что-то делает, а потом вдруг выдает готовое решение. Как он до него дошел — загадка, но дошел. Пытаюсь превратиться в такой черный ящик. Исходные данные: есть калькулятор. Задача: приспособить его для кодирования авизо в ЦБ, и не просто кодирования, а такого кодирования, чтобы людям удобно было его использовать. Входные данные поступили в черный ящик. Когда он выдаст решение — одному Богу известно, и выдаст ли вообще.

— Вы не оправдали оказанное вам высокое доверие.

— Невозможно работать. Вы даете невыполнимые задания. Этот, как его, волюнтаризм.

Точно как в фильме.

— Все в порядке. Кто нам мешает, тот нам поможет.

Телевизор — ко всем чертям! Есть! Есть! Есть! Есть нетривиальное решение! Изюминка!

Кто нам мешает? Маркант! Он нам и поможет!

В калькуляторе два режима: один — для шифрования осмысленного текста ("Батарея — огонь!" набираешь прямо словами и получаешь готовый шифртекст), другой — для использовавшихся еще со времен второй мировой войны кодов с перешифровкой. Набираешь в этом режиме цифры (это если по кодовой книге солдат уже заменил "Батарея — огонь!" на код команды), в качестве шифртекста получаешь тоже цифры. Первые 10 — маркант, а дальше сам шифртекст. Шифртекст зависит как от секретного ключа, записанного в памяти калькулятора, так и от марканта. Маркант в начале режима расшифрования объединяется с секретным ключом, да так, что попробуй измени в нем хоть один знак — все, гамма будет совсем другая! И подобрать другой маркант под такую же гамму практически невозможно. Вводим в цифровом режиме расшифрования в качестве первых 10 знаков — марканта — сумму платежа, выработанная гамма — код подтверждения достоверности! А можно и еще что-нибудь, помимо суммы, например счет, тоже ввести в качестве марканта. Бери от полученной гаммы хоть один, хоть два, хоть десять знаков, как тебе угодно. И без секретного ключа КПД не вычислить ни в жизнь! Это — 10^{100}.

29 сентября 1992 года подписан Договор. Первые две недели — затишье, ЦБ сбивает поднятую Конторой "пену". Подвиги Геракла. А дальше — дело техники. Написать в первом приближении систему выработки ключей, программную реализацию для крупных РКЦ, чтобы не ломали глаза за калькулятором — тоже пара недель. ЦБ оперативно организовывает обучение операционисток. Собирает этих милых девушек и бабушек в большом зале на Житной, я рассказываю им про калькулятор и как с ним работать. Понимают! Улыбаемся друг другу. Не стыдно людям в глаза смотреть.

С 1 декабря 1992 года все авизовки идут с КПД. Гордый заморский "бакс" от удивления на месяц замер: чем же это русский Иван-ЦБ-вич так шустро огромную дырку в своем кармане заштопал? Маркантом.

Глава 4

Ключевые системы

В прошлой главе мы познакомились с основными принципами построения качественного генератора гаммы. И там же видели, что для него нужны некоторые секретные параметры, которые позволят получателю шифрованного сообщения, имея точно такие же параметры, выработать точно такую же гамму и расшифровать сообщение. Такие секретные параметры в криптографии принято называть ключами.

Под *системой выработки ключей* будем понимать некоторый способ выработки секретных параметров, которые затем могут быть использованы в генераторе гаммы для зашифровывания или расшифрования текста, при осуществлении электронной подписи, при идентификации пользователя компьютерной системы, для принятия решения о допуске пользователя к тем или иным ресурсам, регистрации пользователя в различных системах учета и для других целей, требующих параметров, уникальных для данного пользователя и не подлежащих для разглашения посторонним. Эти параметры персональны, недоступны для других пользователей, потому и называются секретными. Каждый пользователь может иметь один или несколько секретных ключей, в зависимости от того, какие задачи он будет решать с помощью компьютерной системы обеспечения безопасности. Например:

- секретный ключ для хранения информации в зашифрованном виде;
- секретный ключ для ведения шифрпереписки со всеми другими пользователями;
- секретный ключ для ведения шифрпереписки с каждым пользователем в отдельности;
- секретный ключ для записи/чтения информации в определенной базе данных;
- секретный ключ для осуществления электронной подписи.

Этот список может быть продолжен до бесконечности.

Важно понять, что способ выработки секретных параметров, например, для генератора гаммы может быть никоим образом не связан с процессом шифрования, осуществляемым с помощью генератора гаммы. Более того, один и тот же генератор гаммы можно использовать при разных способах выработки секретных ключей, в зависимости от того, какая потребность возникает у пользователя. Например, описанный в прошлой главе генератор гаммы можно использовать как для хранения информации на компьютере в закрытом виде, применяя при этом только свой индивидуальный секретный ключ, так и для пересылки информации в зашифрованном виде, применяя при этом уже *систему с открытым распределением ключей*, о которой поговорим поподробнее чуть позже. Сейчас давайте задумаемся над тем, какие требования следует предъявлять к ключевой системе.

Выработка секретных параметров осуществляется путем выбора некоторого конкретного значения из множества всевозможных значений. Например, если ключом является двоичный вектор длиной 512 бит, то множеством всевозможных значений будет множество всевозможных двоичных векторов длиной 512 бит. Как осуществляется выбор? Как правило, случайно и равновероятно, это наиболее оптимальный вариант. Предполагается, что для каждого секретного ключа потенциальный злоумышленник имеет некоторый *критерий отсева ложных вариантов*. Таким критерием может быть наличие в ключе некоторых проверочных параметров, позволяющих контролировать правильность ввода ключа, а также результаты применения этого ключа — различные шифртексты, соответствующие им открытые тексты или некоторые стандарты в них, электронные подписи, проверочные и служебные комбинации и т. п. Таким образом, объем множества всевозможных значений секретного ключа должен быть таким, чтобы исключить возможность его полного перебора потенциальным злоумышленником. Какие это объемы — мы видели в *главе 2*, посвященной вопросам стойкости.

Если ключ выбирается не случайно и равновероятно, например, если по одной части ключа можно определить какую-то информацию о другой его части, то это, в конечном счете, может привести к возможности значительного сокращения опробования всевозможных *допустимых* вариантов ключа. Например, если ключ вырабатывается с помощью имеющейся в языке C функции `RAND`, то для опробования всевозможных вариантов такого ключа любой длины достаточно опробовать множество всевозможных значений, которые могут быть заданы функцией `SRAND`, определяющей начальную точку отсчета при развертывании генератора случайных чисел. Здесь ситуация похожа на ту, с которой мы сталкивались в прошлой главе при описании генератора гаммы. Но есть одно принципиальное отличие. Нам, вообще говоря, не требуется условие возможности повторно выработать тот же самый ключ, поскольку, выработав один раз секретный ключ, мы можем записать

его в файл, устройства типа Touch Memory, Smart Card и любое другое, подобное им, и пользоваться этим устройством в качестве носителя ключевой информации. В качестве источника выработки ключа могут выступать и аппаратные средства — шумовые диоды, вырабатывающие случайный и равновероятный белый шум, повторить который невозможно, может выступать и сам пользователь, нажимая в случайные моменты времени на клавиши клавиатуры. По моментам нажатия на клавиши строится секретный ключ. Но это бывает удобно далеко не всегда.

Хорошо проанализированный генератор гаммы также может использоваться в качестве источника при выработке секретных ключей. Эта ситуация имеет место, например, при централизованном оснащении секретными ключами большого числа пользователей. Из-за большого количества пользователей администратору, отвечающему за обеспечение их секретными ключами, бывает неудобно многократно вырабатывать секретный ключ, нажимая в случайные моменты времени на клавиши компьютера. Нажатий может потребоваться слишком много, и это будет свыше разумных пределов человеческих возможностей. Кроме того, у некоторого пользователя, например, дискета с ключом может в процессе эксплуатации прийти в негодность и потребуется ее замена. Необходимо будет повторно выработать точно такой же секретный ключ, а в этом случае одноразовые генераторы гаммы оказываются непригодными.

Получается, что при выработке секретных ключей с помощью генератора гаммы, обеспечивающего возможность повторной выработки точно таких же ключей, существует некоторый секретный ключ, который мы будем называть *базовым* и с помощью которого можно выработать все требуемые секретные ключи. Естественно, что требования к сохранности базового ключа должны быть намного выше, чем к сохранности отдельных, выработанных с его помощью секретных ключей, — ведь компрометация базового секретного ключа автоматически приводит к компрометации всех таких ключей.

У одного пользователя, как было описно ранее, тоже может быть несколько различных секретных ключей для различных целей. И все их можно аналогично объединить с помощью базового секретного ключа пользователя. Важно только, чтобы используемый при этом генератор гаммы был хорошо проанализирован и не давал возможности в случае компрометации одного из таких ключей определить какую-то информацию о любом другом и, тем более, добраться до базового секретного ключа.

Давайте теперь рассмотрим, какие основные потребности могут возникнуть у пользователя, для которых ему потребуются секретные ключи.

Схемы организации шифрованной связи с традиционной ключевой системой

Индивидуальная связь двух корреспондентов

В этом случае необходимо, чтобы каждый корреспондент имел идентичные ключи, которые он получает заранее и держит в строгом секрете. Например, один из корреспондентов является главным (по иерархии или по договоренности) и вырабатывает ключи на определенный период. Второй экземпляр он передает другому корреспонденту. Передача ключей должна происходить в обстановке строгой конфиденциальности (например, при личной встрече, с помощью доверенного лица и т. п.). Ключи ни в коем случае нельзя передавать в открытом виде (по Internet, электронной почте, телефону, телексу, факсу), так как весь смысл использования шифрованной связи при этом пропадает. Этот вариант можно посоветовать использовать руководителям фирм, выезжающим в командировки, на переговоры, на отдых и т. п. Перед отъездом вырабатывается ключ, который будет использоваться при связи с фирмой, например, по электронной почте. Шифрованные сообщения представляются, в зависимости от используемой шифрсистемы, либо в виде зашифрованных файлов, либо в виде зашифрованных писем какой-то почтовой системы (например, Microsoft Outlook). На приемном конце доверенное лицо руководителя (заместитель, секретарь, помощник), принимая эту шифровку, сразу же вводит ее в компьютер и осуществляет расшифрование.

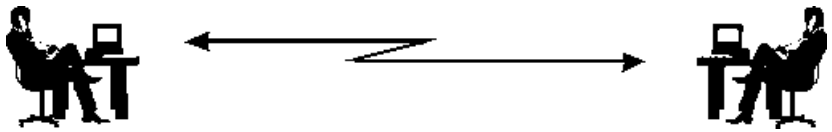

Рис. 4.1. Индивидуальная связь двух корреспондентов

Общая связь нескольких корреспондентов

Здесь все корреспонденты имеют идентичные ключи и могут передавать информацию "каждый — каждому" или один сразу всем остальным (так называемая "циркулярная" связь). Среди корреспондентов должен быть один "главный", **менеджер**, который вырабатывает ключи, размножает их в нужном количестве по числу корреспондентов в сети и организует рассылку ключей при обязательном соблюдении строгой конфиденциальности. Каждый корреспондент должен использовать одну и ту же систему шифрования.

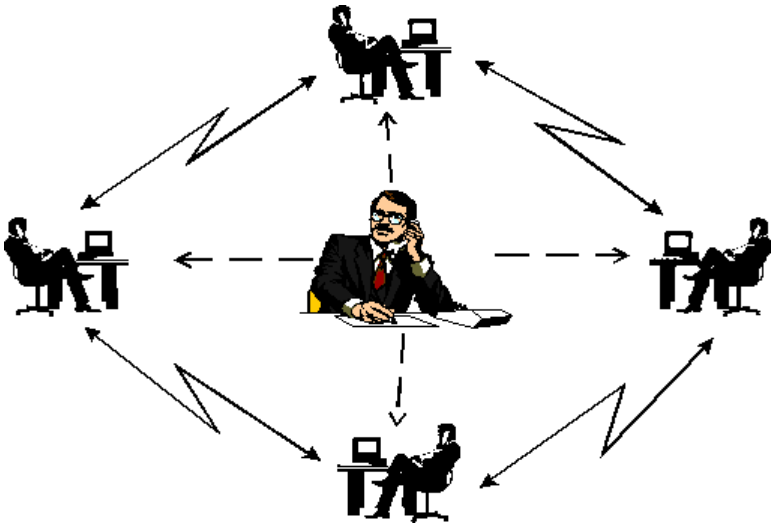

Рис. 4.2. Общая связь нескольких корреспондентов

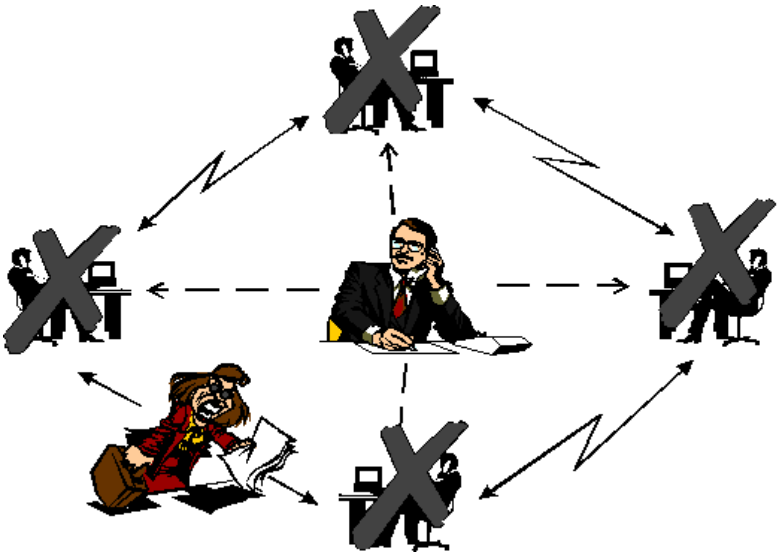

Рис. 4.3. Компрометация общей связи нескольких корреспондентов

Эта система может быть рекомендована для использования в локальных вычислительных сетях для рассылки конфиденциальных циркулярных указаний менеджера пользователям сети.

Главным **недостатком** такой схемы связи является ее уязвимость при компрометации ключа хотя бы у одного корреспондента. Такая компрометация

приведет к возможности чтения переписки по этому ключу у любого корреспондента данной сети.

Другой **недостаток** состоит в том, что информация, предназначенная одному корреспонденту, может быть прочитана всеми другими корреспондентами этой сети.

Сеть связи с индивидуальными ключами

Если каждый корреспондент будет иметь индивидуальный ключ для связи с другими корреспондентами, то такая сеть связи фактически распадается на совокупность индивидуальных связей двух корреспондентов. При этом каждый участник сети должен иметь набор ключевых таблиц по числу своих корреспондентов.

Недостатком этой схемы является громоздкость ключевой документации, увеличение возможности возникновения ошибок и другие неудобства.

Достоинством является то, что компрометация одного ключа не влияет на секретность связи других корреспондентов.

Возможны комбинированные схемы связи, когда используются общие и индивидуальные ключи, а корреспонденты могут разбиваться на группы ("кусты").

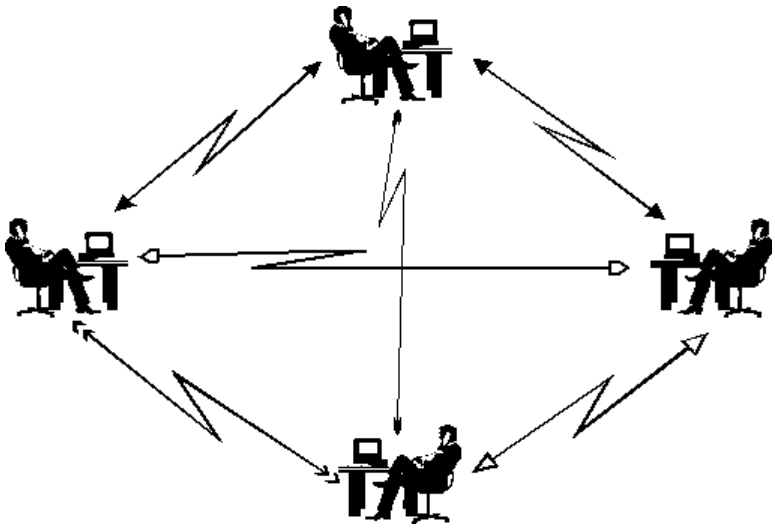

Рис. 4.4. Сеть связи с индивидуальными ключами

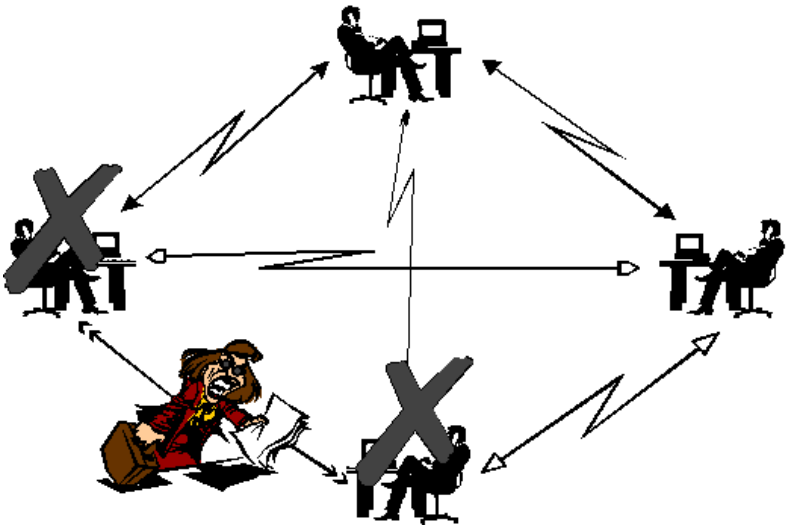

Рис. 4.5. Компрометация сети связи с индивидуальными ключами

Системы связи с "открытым" ключом

Многие недостатки и неудобства схем шифрованной связи с традиционным применением ключей удается преодолеть, используя идею так называемого "открытого" ключа. Принципиальная основа этой идеи состоит в использовании свойства некоторых математических операций, имеющих разную сложность выполнения в "прямом" и "обратном" направлениях. Приведем примеры: умножение простых чисел и разложение на простые множители; возведение многочлена в некоторую степень и нахождение степени, в которую надо возвести некоторый многочлен, чтобы получить заданный результат (так называемое "дискретное логарифмирование"). На основе этого свойства можно разработать такую систему шифрования, для которой процесс зашифровывания выполняется с помощью "открытого" ("общедоступного") ключа, а процесс расшифрования выполняется с помощью секретного ключа, который имеется только у адресата. Определение секретного ключа на основе знания "открытого" ключа (то есть решить для ключа "обратную" задачу) представляет весьма сложную проблему, что и определяет стойкость шифрсистемы.

Разумеется, "сложность" выполнения математической операции зависит от применяемого алгоритма и, в конечном счете, сводится к оценке числа так называемых "элементарных" операций, составляющих алгоритм вычисления. Если иметь в виду, что алгоритм математической операции реализуется на ЭВМ, то при данных параметрах ЭВМ (скорость вычислений, объем долговременной и оперативной памяти) "сложность" математической операции

может быть оценена временем ее выполнения на ЭВМ. Для рассмотренных примеров существуют тривиальные алгоритмы, основанные на последовательном переборе всех возможных вариантов решения и выделении среди них истинного варианта. Однако такой "силовой" метод требует больших временных затрат и на практике невыполним, если параметры задачи (в примерах — это величина числа, разлагаемого на простые множители; свойство многочлена и величина степени, в которую он возводится) достаточно велики.

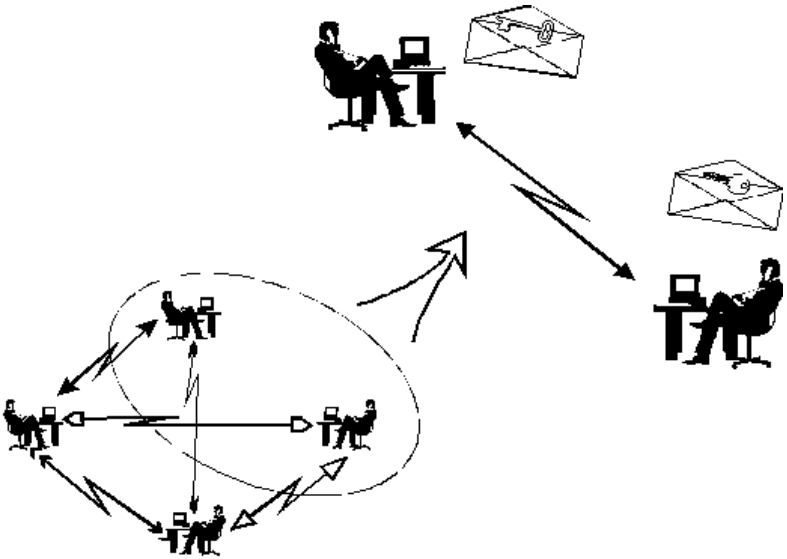

Рис. 4.6. Система связи с "открытым" ключом

Поиск и разработка эффективных алгоритмов для выполнения некоторых математических операций или доказательство невозможности их построения представляют сложную математическую проблему, над решением которой упорно работают математики всего мира. В последние годы получены некоторые успехи в этой области, что, конечно, заставило переоценить стойкость некоторых шифрсистем, построенных на основе применения "открытого" ключа. Однако существуют проблемы, где продвижения в поисках эффективных алгоритмов оказались недостаточными для их практической реализации или требуют чрезвычайно больших материальных и временных затрат. То, что над этими проблемами работают самые квалифицированные математики всего мира, пытаясь улучшить алгоритмы решения "обратных" задач, дает основание полагать, что на определенный период времени такого рода "сложные" математические операции могут обеспечивать криптографическую стойкость шифрсистем, построенных с их использованием. Именно к таким проблемам относится "дискретное логарифмирование".

Итак, пусть условно называемый элемент x является личным секретным ключом абонента A, а элемент y — секретным ключом абонента B. Предположим, что эти два абонента хотят установить между собой шифрованную связь таким образом, чтобы:

- сообщения могли расшифровывать только они двое и никто другой;
- не сообщать друг другу своих личных секретных ключей.

Для реализации этих на первый взгляд противоречивых требований поступают следующим образом. Абонент A вырабатывает свой так называемый открытый ключ, т. е. элемент a^x, а абонент B — a^y. Здесь a — заранее фиксированный элемент (т. н. основание, оно фиксировано), а операция возведения в степень осуществляется в конечном поле, элементы которого достаточно сложны — это могут быть векторы, эллиптические кривые и т. п. Функция возведения в степень в конечном поле имеет ту особенность, что вычислить ее в одну сторону достаточно легко (возвести в степень), а провести обратную операцию (логарифмирование в конечном поле — зная a^x, вычислить x) практически невозможно. Абоненты A и B обмениваются своими открытыми ключами по любому доступному каналу связи. Если какое-либо третье лицо перехватит открытый ключ, то вычислить секретный ключ оно не сможет — для этого придется решать задачу логарифмирования в конечном поле. Более того, абонент A не будет знать личного секретного ключа абонента B, также как абонент B не будет знать личного секретного ключа абонента A. Получив от B его открытый ключ a^y, абонент A вычисляет a^{yx} (возводит открытый ключ абонента B в степень своего секретного ключа), а абонент B возводит открытый ключ абонента A в степень своего секретного ключа y — вычисляет a^{yx}. Значение a^{xy} совпадает с a^{yx} и является тем самым ключом, на котором они будут шифровать сообщения при обмене конфиденциальной информацией друг с другом.

Преимущества систем с открытым распределением ключей очевидны: не нужно специального курьера или доверенного лица для рассылки ключей — можно рассылать открытые ключи по любым каналам в открытом виде, каждый пользователь системы имеет свой личный секретный ключ, который не может быть вычислен даже при компрометации ключей у других пользователей. Система с открытым распределением ключей наиболее предпочтительна в банковских системах (каждый клиент банка имеет свой секретный ключ, банк знает только его открытый ключ и принимает платежные поручения в шифрованном виде), при обмене шифрованной информацией по глобальным компьютерным сетям (в режиме электронной почты осуществляется обмен файлами с открытыми ключами и устанавливается шифрованная связь между потенциальными деловыми партнерами) и т. п.

К **недостаткам** системы с открытым распределением ключей следует отнести сложность подготовки шифрованных циркулярных сообщений для большого числа абонентов (для каждого абонента должна осуществляться индивидуальная шифровка). Однако система с открытым распределением ключей

может пригодиться и в этом случае: общий ключ, на котором будут шифроваться циркуляры, сам зашифровывается с использованием системы с открытым распределением ключей и рассылается в шифрованном виде всем абонентам по открытым каналам.

К другому недостатку систем с открытым распределением ключей относится то, что они все основываются на трудоемких, но сравнительно редких и *общеизвестных* математических задачах. Над их решениями трудятся математики всего мира и дамоклов меч решения кем-нибудь из них такой задачи будет означать компрометацию всех основанных на ней ключей. Вот, например, цитата статьи "Задача нахождения ключа шифрования RSA решена" из журнала "New Scientist".— 1994 г.— 142, N 1924 .— P. 5.

"Задача нахождения ключа шифрования RSA решена. Группе специалистов-любителей удалось решить криптографическую задачу, поставленную еще в 1977 г. Для ее решения потребовалось восемь месяцев совместной работы 600 человек на компьютерной сети Internet. Решение задачи показывает, насколько далеко продвинулись математики и вычислительная техника за эти 17 лет. В момент постановки задачи считалось, что она может быть решена только в XXI в. Как заявил один из членов группы, они бы нашли правильный ответ за восемь недель, если бы в их распоряжении были специализированные ЭВМ. По своей сути задача заключалась в разложении на простые сомножители 129-значного числа. Полученные числа используются в качестве ключа для раскрытия шифрованного сообщения. Разложение на простые множители больших чисел считалось тяжелой задачей, для решения которой, как раньше полагали, потребуется несколько сотен лет работы очень мощной ЭВМ. Группе специалистов американских университетов Оксфорда, Айовы и Массачусетского технологического института удалось разбить поставленную задачу на несколько значительно более легких. Каждый из 600 добровольцев из 25 стран установил в своем компьютере программу, которая занималась решением задачи в свободное от основной работы время. Готовые решения частных задач были введены в суперкомпьютер, на котором было найдено решение поставленной задачи. Задача была сформулирована в 1977 г. тремя математиками, которые позднее разработали известную систему шифрования с открытым ключом RSA. В этой системе ключ шифрования не является секретным. Разложение числа на простые сомножители, которое до сих пор считалось невероятно трудной математической задачей, позволяет по известному ключу шифрования найти секретный ключ расшифрования. Несмотря на сенсационное раскрытие ключа, криптосистема RSA продолжает считаться стойкой к дешифрованию.

Действительно, добровольцам удалось раскрыть всего один секретный ключ. Для раскрытия любого другого ключа той же длины потребовалось бы примерно столько же времени и средств. Кроме того, длина ключа в 129 цифр может считаться небольшой, хотя ключи такой длины широко используются для шифрованной связи. Увеличение длины ключа на единицу в шесть раз

усложняет его раскрытие. И все же решение задачи раскрытия 129-значного ключа показывает, что "криптоаналитики дышат в спину криптографам". Так что 155-значный ключ, который сегодня считается весьма надежным, через несколько лет вполне может оказаться поддающимся раскрытию. Поэтому для защиты ценной информации в шифрсистеме RSA рекомендуется пользоваться ключами длиной в 200—300 знаков".

Позволю себе немного прокомментировать это сообщение.

1. В системе с открытым распределением ключей разработчик не может значительно увеличивать размер секретного ключа, поскольку в этом случае значительно возрастает задача вычисления секретного ключа самим законным пользователем, что приводит к снижению скорости работы шифрсистемы. Поэтому при использовании системы с открытым распределением ключей разработчики и пользователи должны четко отдавать себе отчет и ответить на вопрос, что для них важнее: гарантированная стойкость или удобства в эксплуатации? Гарантированной стойкости (в нашем понимании этого слова — 10^{100}) ни в одной из современных систем с открытым распределением ключей достигнуть не удается.
2. Эти данные дают косвенное подтверждение того неоднозначного вывода, что системы, основанные на проблеме логарифмирования в конечных полях, несколько более надежны, чем те, которые основаны на задаче разложения на простые множители.

С появлением более скоростной вычислительной техники возможно появление новых, более стойких алгоритмов открытого распределения ключей. Вот что, например, пишут об этом А. В. Лунин и А. А. Сальников в своей статье "Перспективы развития и использования асимметричных алгоритмов в криптографии".

"Практически используемые алгоритмы асимметричной криптографии основаны на двух задачах: дискретного логарифмирования и факторизации. Существуют различные мнения о возможности решения этих задач в будущем и о том, как скоро это может произойти. Поэтому для специалистов в области защиты информации постоянно остается актуальной задача разработки альтернативных систем. При этом главными остаются проблемы существования односторонней функции и функции с секретом. Здесь следует выделить следующие направления исследований.

Глобальная теоретическая идея построения новых асимметричных криптосистем заключается в попытке порождения функций с секретом с помощью "маскирования" простых задач под сложные (NP-полные). Было предложено много вариантов, но все они оказались нестойкими.

Полученные результаты по вскрытию некоторых вариантов криптосхемы на основе задачи о "рюкзаке" сформировали в среде криптографов-практиков скептическое отношение ко всем подобным схемам. В то же время,

формально схема Шора-Райвеста не взломана по сей день. Ряд криптографов-теоретиков считают задачу о "рюкзаке" одной из самых перспективных для построения алгоритмов асимметричной криптографии, поскольку "сложность" заложена в самой ее природе. Кроме того, в случае успеха, такой схеме, по-видимому, не будет равных по эффективности (скорости работы).

Схема открытого распределения ключей с использованием некоммутативных групп была предложена лабораторией МГУ по математическим проблемам криптографии в 1993 году, что явилось принципиально новым подходом к этой задаче. Однако до сегодняшнего дня практически реализуемых схем, основанных на этих идеях, не предложено.

После того как Сидельников и Шестаков, используя быстрые алгоритмы декодирования, показали, что одна из схем типа МакЭлиса (схема Нидеррайтера) нестойкая, был предложен ряд вариантов схемы на основе теоретико-кодовых конструкций. Практического применения не нашла ни одна из них либо в силу своей громоздкости, либо в силу того, что ее стойкость вызывает большие сомнения у специалистов.

Реализация протоколов асимметричной криптографии (в первую очередь, схем электронной цифровой подписи) с использованием традиционных криптографических схем. Это направление, на наш взгляд, незаслуженно находится в тени. Теоретическая возможность построения таких систем показана Лампортом, Наором и Юнгом. Известна также интересная статья Березина и Дорошкевича, где приведены схемы ЭЦП, основанные на симметричных алгоритмах. Кроме того, корпорация IBM в некоторых из разработанных ею банковских систем реализует схемы ЭЦП на основе алгоритма DES с использованием защищенных модулей.

С начала 90-х годов прошлого века широко обсуждается возможность реализации протоколов асимметричной криптографии на основе квантово-механических эффектов".

Надеюсь, что описанный мною в прошлой главе пример с банковскими авизо помог читателю лучше понять те разнообразные и подчас неожиданные ситуации, с которыми иногда приходится сталкиваться криптографу в реальной жизни. Вся криптография и синтез и анализ криптосистем пропитаны оригинальными и нетривиальными решениями, стандарты ей решительно вредны!

Глава 5

Блочные шифры

Из предыдущих глав этой книги у читателя может сложиться впечатление, что широкая столбовая дорога при разработке шифрсистемы одна, ясная и понятная: строй хороший генератор гаммы, удовлетворяющий условиям Шеннона, не допускающий перекрытий гаммы, строй хорошую ключевую систему, можно даже с использованием этого генератора гаммы, только на секретных ключах или же с использованием системы с открытым распределением ключей, добавляй электронную подпись, поскольку шифр гаммирования не является имитостойким, накладывай гамму на открытый текст — и вперед, к светлым криптографическим целям. В принципе, так. Даже, более того, в большинстве практических случаев именно так. Но жизнь всегда ставит разные задачи. И может оказаться, что идти по этой столбовой дороге неудобно. Захочется свернуть с нее куда-нибудь в сторону и попробовать пройти этот путь как-то по-другому. Взять и использовать столько раз упоминаемую недобрым словом в этой книге простую замену.

Почему она плохая? Потому, что не скрывает статистики открытого текста. А в любом разговорном тексте, множестве ASCII-символов, кодовой таблице мощность используемого алфавита, по криптографическим меркам, раз, два — и обчелся. 33 (алфавит русского языка), 26 (латинский алфавит), 256 (ASCII-символы) — ну разве это величины, по сравнению с криптографическими 10^{10}, 10^{40}, 10^{100}, где счет обычно идет не на коэффициенты, а на порядки. И если мы берем, например, страницу текста на русском языке (2000—2200 символов), то на ней будет встречаться почти каждая буква русского алфавита и всячески выпячивать из-за преобладаемости в языке свою статистику, что было совершенно справедливо подмечено Шенноном. То же самое в компьютерном файле. Возьмите, к примеру, файл, содержащий текст, набранный в редакторе Microsoft Word, и подсчитайте в нем количество нулей — обычно явное преимущество перед серой массой остальных символов. Вот мы и закрывали эти преобладания случайной и равновероятной гаммой.

И тут можно свернуть с этой неоднократно проверенной дороги в кусты. Раздуем наш маленький алфавит до такой степени, чтобы вероятности повторения знаков в открытом тексте были бы практически нулевые! Вместо одного символа будем шифровать одновременно по несколько — столько, чтобы в получившемся после разбиения на такие *блоки* открытом тексте практически не было повторения блоков. Не всегда такой подход сразу применим. Например, в набранном в Microsoft Word текстовом файле блоки из одних нулей могут быть настолько велики, что любое разумное разбиение для них будет мало: ведь чем больше блок, тем труднее его зашифровать. Но для шифрования блоков будем применять антишенноновский шифр — простую замену.

Вот такие криптографические кусты выросли в середине 1970-х годов на шенноновской столбовой дороге. Попробуем разобраться, что к чему.

Если вспомнить главу 3 про генератор гаммы, где мы упоминали подстановки, то нетрудно видеть, что блочный шифр — это просто *подстановка большой степени*. Какой именно? Обычно, если речь идет о компьютерных приложениях, это $(2^8)^8 = 2^{64} = 10^{19}$, т. е. блок состоит из 8 байт открытого текста. Кстати, со значительным количеством одних и тех же блоков в тексте (как, например, с нулями в текстовом файле, набранном в Microsoft Word) легко можно бороться, добавляя при зашифровывании к блоку, например, его порядковый номер. Блоки будут с большой вероятностью отличаться друг от друга, а при расшифровании легко получить первоначальный текст.

Естественно, что ни о каком *табличном* задании такой подстановки не может быть и речи: слишком велик объем алфавита. Если в примере из главы 3 нам требовалось под подстановку 256 байт оперативной памяти, то здесь бы потребовалось 10^{19} байт — это 10^{13} Мбайт, или, говоря программистским языком, 10 триллионов мегабайт оперативки. Думаю, что ни сейчас, ни через ближайшие 10 лет эти величины недостижимы. Через 100 лет — не знаю. Но давайте думать о хлебе насущном, а не о светлом криптографическом будущем всего человечества.

Остается одно — задавать такую подстановку алгоритмически, т. е. придумывать алгоритмы, осуществляющие *регулярное* (взаимно однозначное) преобразование множества всевозможных блоков само в себя. Если преобразование не будет регулярным, то может получиться так, что мы не сможем однозначно расшифровать то, что зашифровали. Примером регулярного преобразования может служить преобразование, получаемое из упомянутого в главе 3 рекуррентного соотношения:

$$x_{i+n} = x_i + x_{i+1} + \ldots + x_{i+n-1}$$

В нем первые n блоков $x_0, x_1, \ldots, x_{n-1}$ — открытый текст, а шифртекст — значения $x_T, x_{T+1}, \ldots, x_{T+n-1}$ при некотором целом неотрицательном значении T.

Прочитал эти строки внимательный читатель и сразу же возмутился: обмануть хотел? Да в таком шифре и ключа-то нет! Какой же это шифр!

Каюсь, проверял бдительность.

Пусть k_0, k_1, ..., k_{T-1} — некоторый секретный ключ. Начинаем его позначно складывать со знаками рекуррентного соотношения:

$$x_{i+n} = x_i + x_{i+1} + ... + x_{i+n-1} + k_i.$$

И опять этот въедливый читатель хватает автора за руку. Аналогичный генератор гаммы был плохим. Наверное, и блочный шифр будет не лучше? Точно, не лучше. Почему?

Например, при $T = 0$ это просто преобразование любого блока самого в себя.

При $T < n$ в блоке шифртекста будет кусок блока открытого текста.

При любом значении *T-линейные* зависимости переходят из блоков открытого текста в блоки шифртекста.

Например, если есть два блока открытого текста

$$x_0, x_1, ..., x_{n-1}$$

$$y_0, y_1, ..., y_{n-1}.$$

то по разностям $r_0 = (x_0 - y_0)$, $r_1 = (x_1 - y_1)$, ..., $r_{n-1} = (x_{n-1} - y_{n-1})$ в блоках открытого текста легко вычисляются разности в блоках шифртекста — ведь при вычитании одни и те же знаки секретного ключа сокращаются! Точно так же по разностям в блоках шифртекста легко вычисляются разности в блоках открытого текста. Бесключевое чтение в чистом виде.

Вопрос на засыпку: как с этим бороться? Молчу 3 секунды. Ответ — на 5! Так же, как и в генераторе гаммы — вводя в это преобразование подстановку π из симметрической группы S_{256}, так же, как вводили ее в генератор гаммы.

$$x_{i+n} = \pi(x_i + x_{i+1} + ... + x_{i+n-1})$$

А куда поставить знак ключа? "Конечно же под подстановку!" — дружно загалдели продвинутые читатели. А вот и не угадали!

В *главе 3*, упоминая подстановки, мы заметили, особо не вдаваясь в детали, что множество всевозможных подстановок образует симметрическую группу. А во всякой группе, как сказывают математики, у любого ее элемента есть *обратный*, т. е. такой, домножение на который дает единицу группы. Чтобы программисты, далекие от математики, взяв в руки эту книгу, в этом месте не сделали ей полный ABORT, выполню роль функции RETRY и переведу ранее сказанное на их нормальный язык.

Подстановка — это, в нашем случае, просто таблица всевозможных неотрицательных чисел от 0 до 255. Массив unsigned char размера 256, в котором

как-то перемешаны все значения типа unsigned char. 0 может быть в нем на 35 месте, 1 — на 147 и так далее. Но в нем обязательно должны быть и 0, и 1, и вся остальная таблица ASCII-символов. А обратная подстановка — это когда в этой таблице адреса и значения меняются ролями: что было адресом — становится значением, а значение — адресом. И обозначается обратная подстановка как π^{-1}.

Давайте теперь составим *уравнение расшифрования*, т. е. опишем те преобразования, которые надо будет выполнить для расшифрования зашифрованного блока.

$$x_i = \pi^{-1}(x_{i+n}) - x_{i+1} - \ldots - x_{i+n-1}.$$

Пока все без ключа. Это мы сейчас о том научные дебаты ведем, куда его лучше подставить.

Если в уравнении зашифровывания подставим знак ключа *под* подстановку, то получим:

- уравнение зашифровывания: $x_{i+n} = \pi(x_i + x_{i+1} + \ldots + x_{i+n-1} + k_i)$;
- уравнение расшифрования: $x_i = \pi^{-1}(x_{i+n}) - x_{i+1} - \ldots - x_{i+n-1} - k_i$.

Если в уравнении зашифровывания подставим знак ключа вне подстановки, то получим:

- уравнение зашифровывания: $x_{i+n} = \pi(x_i + x_{i+1} + \ldots + x_{i+n-1}) + k_i$;
- уравнение расшифрования: $x_i = \pi^{-1}(x_{i+n} - k_i) - x_{i+1} - \ldots - x_{i+n-1}$.

Что же получается? Наш главный враг, Сонька Золотая Ручка, норовящая подобрать ключ к нашей криптографической квартире или как-то вскрыть ее без ключа, — это линейность. Мы от нее защищаемся подстановкой. Чем быстрее защитимся, тем лучше. А она может влезть или через дверь (уравнения зашифровывания) или через окно (уравнения расшифрования). Можно покрепче закрыть дверь, но при этом оставить широкую щель в окне. А можно оставить маленькую щелку в двери, но надежно законопатить окно.

Когда знаки ключа ставятся под подстановку, мы надежно (на первый взгляд!) защищаем дверь, но в окне остается широкая щель: первые n тактов в уравнения расшифрования знаки ключа входят линейно!

Когда знаки ключа ставятся вне подстановки, то в двери остается маленькая щелочка: первый раз знак входит линейно, но уже на следующем такте он попадает под подстановку. Зато окно уже надежно закрыто.

Рассмотрев пример регулярного преобразования, вернемся к обсуждению достоинств и недостатков блочных шифров по сравнению с шифрами гаммирования.

Очевидно, что с точки зрения имитостойкости блочные шифры более предпочтительнее, поскольку целенаправленно навязать в них нужные знаки открытого текста намного труднее, чем в шифрах гаммирования. Но в то же

время такие шифры *распространяют искажения*: неправильный прием одного знака шифртекста, вызванный, например, помехами в канале связи, приводит к неисправимому искажению целого блока открытого текста. Шифры же гаммирования искажений не распространяют.

В блочных шифрах, в принципе, не нужен маркант, однако, как правило, нужен некоторый механизм, призванный не допустить повторения блоков в открытом тексте. Ранее мы уже упоминали о добавлении к блоку его номера, возможны и другие варианты: зацепление блоков, предварительное наложение на открытый текст случайной, известной и не зависящей от ключа гаммы, специальные правила формирования блоков и тому подобное. Дело фантазии разработчика. И не исключено, что некоторое подобие марканта все-таки понадобится.

Очевидно, что максимальный объем информации, закрываемой с помощью блочного шифра, должен быть ограничен максимальным количеством блоков. Самый простой способ криптографического анализа блочного шифра — по *каталогу*. Собираем коллекцию блоков шифртекста и, если удается, какую-то информацию о соответствующих им блоках открытого текста. Короче говоря, составляем каталог. Если какой-то блок в открытом тексте повторится, то это сразу же можно отследить по каталогу.

Часто блочный шифр можно использовать и в качестве генератора гаммы, как мы могли в этом убедиться на приведенных примерах. И здесь проявляются некоторые подходы к обоснованию положительных качеств такого генератора. Например, как мы уже видели в *главе 3*, генератор гаммы не должен вырабатывать гаммы с коротким периодом. Гамма, вырабатываемая с помощью блочного шифра, — это просто некоторый отрезок, взятый из цикла подстановки большой степени, реализуемой блочным шифром. Если цикл подстановки короткий, то это означает появление периодичности у вырабатываемой таким способом гаммы. Поэтому при анализе блочного шифра стоит задуматься над тем, какими свойствами будут обладать вырабатываемые подстановки большой степени. Не будет ли у них коротких циклов? Какова будет группа, порожденная такими подстановками? При каких условиях она будет знакопеременной или симметрической? Поиск ответов на эти и подобные им вопросы как раз и составляет основную задачу современного криптоаналитика.

Рассмотрим теперь блочные шифры DES и ГОСТ 28147-89.

Вот цитаты из замечательной книга Брюса Шнайера "Прикладная криптография": "Стандарт шифрования данных DES (Data Encryption Standard), который ANSI называет Алгоритмом шифрования данных DEA (Data Encryption Algorithm), а ISO — DEA-1, за 20 лет стал мировым стандартом. Хотя на нем и появился налет старости, он весьма прилично выдержал годы криптоанализа и все еще остается безопасным по отношению ко всем врагам, кроме, возможно, самых могущественных".

"DES представляет собой блочный шифр, он шифрует данные 64-битовыми блоками... На простейшем уровне алгоритм не представляет ничего большего, чем комбинация двух основных методов шифрования: смещения и диффузии. Фундаментальным строительным блоком DES является применение к тексту единичной комбинации этих методов (подстановка, а за ней перестановка), зависящей от ключа. Такой блок называется этапом. DES состоит из 16 этапов, одинаковая комбинация методов применяется к открытому тексту 16 раз".

Ну что же, отправимся вслед за Шнайером по этапам DES.

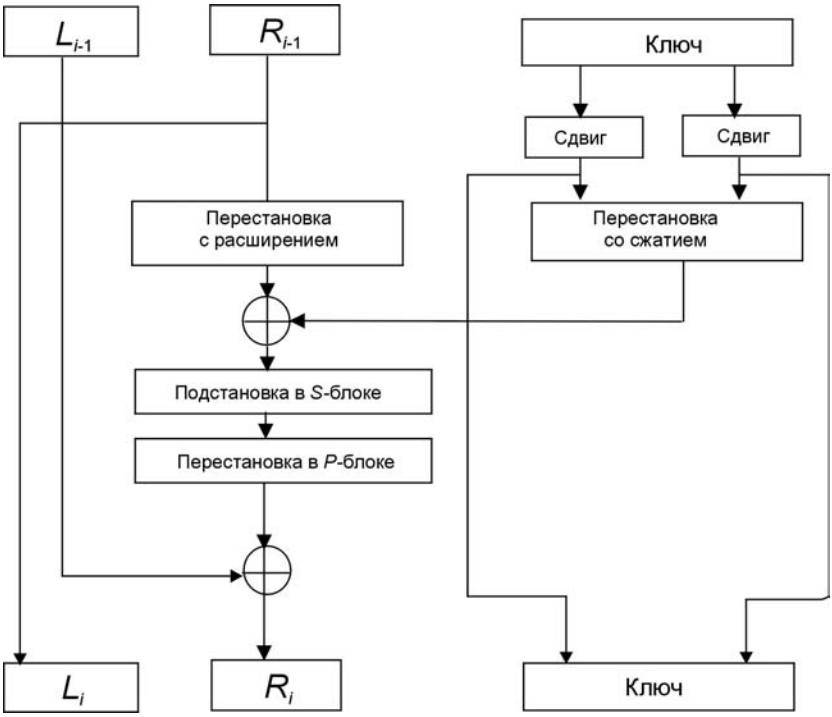

Рис. 5.1. Этап алгоритма DES

Явное преобладание слова "перестановка" на этапе. Да и подстановка какая-то не та, которую мы раньше рассматривали, а в таинственном "*S*-блоке". Читаем внимательнее.

Перестановка с расширением. Эта операция расширяет правую половину данных, R_i, от 32 до 48 битов. Так как при этом не просто повторяются определенные биты, но и изменяется их порядок, эта операция называется **перестановкой с расширением**. У нее две задачи: привести размер правой половины в соответствие с ключом для операции XOR и получить более длинный результат, который можно будет сжать в ходе операции подстановки.

Однако главный криптографический смысл совсем в другом. За счет влияния одного бита на две подстановки быстрее возрастает зависимость битов результата от битов исходных данных. Это называется **лавинным эффектом**. DES спроектирован так, чтобы как можно быстрее добиться зависимости каждого бита шифртекста от каждого бита открытого текста и каждого бита ключа".

Перечитав этот абзац, читатель, конечно же, обратил внимание на явное преобладание в нем слова "бит". А программист задумчиво почесал в затылке — ему потом под все эти хитрые манипуляции с битами программу писать...

А вот и картинка, наглядно показывающая перестановку с расширением. Вернее, не всю, а только одну половинку. Другая — такая же.

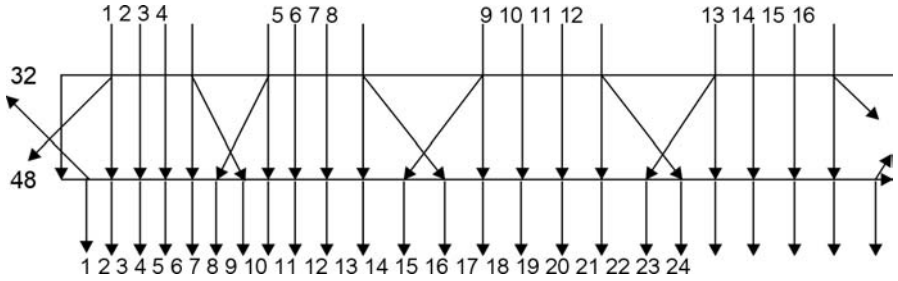

Рис. 5.2. Перестановка с расширением алгоритма DES

Давайте посочувствуем программисту. Язык ассемблер, типовые операции, работа, в основном, с байтами. Бит на свет божий вытащить в ассемблере можно, но это все-таки пусть небольшая, но проблема. Сдвигаешь на сколько надо байт, по состоянию Carry Flag определяешь значения бита. Но в ассемблере удобнее иметь дело с байтами. Проще получаются программы, да и работают они потом быстрее. Например, вместо такой перестановки с расширением реализовывать обычную подстановку из S_{256} намного проще. Поменял местами адрес и значение — и нет проблем! Если на С программу пишешь — это просто манипуляция со значками * и &. И никаких циклических сдвигов байт. А простота — родная сестра скорости и дополнительная гарантия от ошибок в программе.

Но тут таинственный "лавинообразный эффект". А может и подстановка из S_{256} дает эффект не хуже? Предлагаю читателю проверить это самостоятельно. Выписать немножко уравнений $x_{i+n} = \pi(x_i + x_{i+1} + ... + x_{i+n-1}) + k_i$ и проследить за тем, кто быстрее добивается "зависимости каждого бита шифртекста от каждого бита открытого текста и каждого бита ключа". Да еще в том случае, когда подстановка π сама формируется из ключа.

Так почему же в схеме DES выбраны такие, прямо скажем, неоптимальные с точки зрения реализации (а отсюда — и скорости!) криптографические

решения? А это с какой точки зрения неоптимальные. Забудем про компьютеры и язык ассемблер. Представим себе провода и контакты. Вот тут блок перестановки действительно оптимален! Две группы контактов: одна сверху, другая снизу. И провода-перепайки между ними. Старая-старая криптографическая сказка! Еще со времен первой и второй мировых войн. Отложим пока в сторону книгу Брюса Шнайера, а возьмем другую, уже не раз упоминавшуюся здесь книгу, — Дэвида Кана.

"Ранним декабрьским утром 1917 года симпатичный молодой человек стремительно промчался между массивными колоннами вестибюля здания компании AT&T... Молодого человека звали Гильберт Вернам... Коллеги считали Вернама весьма толковым инженером и способным изобретателем... У Вернама был редкий склад ума, который позволял ему придумывать оригинальную электрическую цепь и затем переносить ее на чертежный холст...

Работа над секретным проектом началась еще летом, несколько месяцев спустя после того, как Соединенные Штаты объявили войну Германии. Паркер[1] поручил нескольким своим подчиненным исследовать вопрос о возможности сохранять в тайне сообщения, передаваемые по телетайпу... Вернам предложил готовить перфоленту со случайными знаками (так называемую "гамму") заранее и затем электромеханически складывать ее импульсы с импульсами знаков открытого текста... Чтобы при шифровании суммировать импульсы электрически, Вернам сконструировал специальное устройство, состоящее из магнитов, реле и токосъемных пластин..."

Снова берем Брюса Шнайера и читаем дальше.

"**Подстановка с помощью *S*-блоков**. После объединения сжатого блока с расширенным блоком с помощью XOR над 48-битовым результатом выполняется операция подстановки. Подстановки производятся в восьми блоках подстановки, или *S*-блоках. У каждого *S*-блока 6-битовый вход и 4-битовый выход, всего используется восемь различных *S*-блоков...

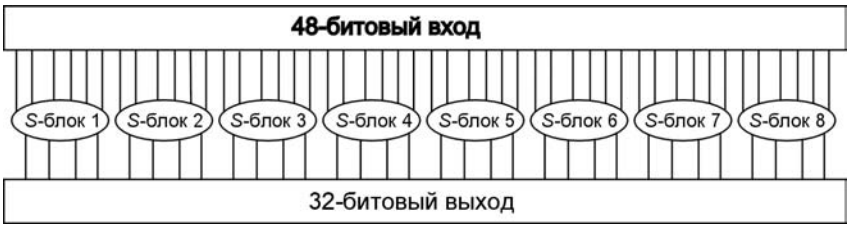

Рис. 5.3. Подстановка с помощью *S*-блоков в алгоритме DES

Каждый *S*-блок представляет собой таблицу из 2 строк и 16 столбцов. Каждый элемент в блоке является 4-битовым числом... Входные биты особым

[1] Начальник Вернама.

образом определяют элемент *S*-блока. Рассмотрим 6-битовый вход *S*-блока: b_1, b_2, ..., b_6. Биты b_1 и b_6 объединяются, образуя 2-битовое число от 0 до 3, соответствующее строке таблицы. Средние 4 бита, с b_2 по b_5, объединяются, образуя 4-битовое число от 0 до 15, соответствующее столбцу таблицы..."

Давайте послушаем разговор программистов. Винчестер — ... гига**байт**, оперативная память — ... мега**байт**, переменная char в программе — один **байт**. Явное преобладание слова **байт**, хотя и с коэффициентами типа кило, мега, гига. Хоть где-нибудь встречается слово **бит**? Да, безусловно. Но, в основном, когда речь идет о модеме, работающем на телефонной линии. Вот там 28, 8 кило**бит** в секунду. О байтах, как правило, в разговоре о модеме не упоминают. Переводят в биты. А почему?

Дело в том, что для увеличения *скорости* обработки информации по компьютерным шинам она, в основном, пересылается байтами. Восемь (а то и 16, 24, 32, ...) тоненьких проводов в одной связке. Но биты идут по ним рядами по 8. И процессор такими же рядами обрабатывает их. Сразу по 8. Пересылать и обрабатывать биты поодиночке было бы медленно и неудобно. Вот при проектировании компьютеров и приняли такое правило: все — сразу по 8 (а можно еще и с коэффициентами). Выделять из байта бит процессору неудобно. Не для него такая операция предназначена. А вот на телефонной линии всего один провод. Тут уж хочешь, не хочешь — надо под старые, с незапамятных времен существующие каналы связи подстраиваться. И компьютер подстраивается. Переводит свои байты в биты, выстраивает их в цепочку вместо шеренги и посылает в канал связи. Но скорость при этом теряется.

А теперь опять отложим в сторону Шнайера и возьмем в руки книгу Кана.

"...Случилось так, что в период между двумя мировыми войнами... сразу несколько человек, как и Вернам, побуждаемые широким использованием секретной связи в военное время и вдохновляемые наступлением эпохи механизации, независимо друг от друга изобрели машину, принцип действия которой на протяжении очень продолжительного времени находил наиболее широкое применение в криптографии. Этот принцип основывается на использовании колеса с перепайками — так называемого шифрдиска.... С обеих сторон шифрдиска по окружности на равном расстоянии друг от друга закреплены по 26 электрических контактов... Каждый контакт соединяется перепайкой с каким-либо другим контактом на противоположной поверхности шифрдиска. Таким образом, образуется электрическая цепь, которая начинается на одной стороне шифрдиска и заканчивается на другой.

Если условиться, что контакты на одной (входной) поверхности представляют буквы открытого текста, а контакты на другой (выходной) поверхности — буквы шифртекста, то проволочные перепайки между входной и выходной поверхностью обеспечивают преобразование открытого текста в криптограмму.... Для выполнения этих манипуляций шифрдиск устанавливается между двумя неподвижными круглыми пластинами, каждая из которых снабжена 26 контактами..."

Дисковые шифраторы. Целая эпоха в криптографии. Кто интересуется — почитайте о них подробнее у Дэвида Кана. Вот еще одна цитата, еще одно весьма известное в криптографическом мире имя.

"Борис Хагелин уменьшил это устройство до нужных размеров и назвал его "С-36"... При работе с "С-36" оператор сначала устанавливал ключ, а потом поворачивал ручку, расположенную слева от буквы открытого текста на клавиатуре, и вращал рукоятку, расположенную справа. При этом механизм делал один оборот, и маленькое колесико печатало выходной знак шифртекста на бумажной ленте... Скорость работы "С-36" составляла в среднем 25 букв в минуту".

А теперь вернемся к *S*-блокам алгоритма DES и опять посочувствуем программистам, которые возьмутся их программировать. Почти все операции — с битами, противоестественными для компьютерных технологий. А если посмотреть на них по другому? Что такое каждый из восьми *S*-блоков? Это 4 шифрдиска, правда не двигающиеся с помощью "рукоятки, расположенной справа", как в дисковых шифраторах Бориса Хагелина. Такие шифрдиски еще принято называть коммутаторами. Каждый — на 16 контактов, между контактами — проволочные перепайки, значения b_1 и b_6 — 4-позиционный переключатель, определяющий, какой из коммутаторов в данный момент использовать. Значение b_2, ..., b_5 — аналог буквы алфавита, подаваемой на проволочную перепайку. Легко реализуется аппаратно с помощью типовых логических элементов. Криптографические идеи 50—60 годов! На проводах-контактах реализовывать очень удобно, но для программной реализации, где счет идет на байты, а не биты, явно притянутые за уши. И поэтому не может такая программная реализация обеспечить высокую скорость работы криптографической системы. Да она изначально и не предполагалась для программной реализации. Вот что сказано об этом у Шнайера.

"Никогда до этого оцененный NSA (National Security Agency) алгоритм не был опубликован. Возможно, эта публикация была следствием непонимания, возникшего между NSA и NBS (National Bureau of Standards). NSA считало, что DES будет реализовываться только аппаратно. В стандарте требовалась именно аппаратная реализация, но NBS опубликовало достаточно информации, чтобы можно было создать и программную реализацию DES. Не для печати NSA охарактеризовало DES как одну из самых больших своих ошибок..."

И все это стало мировым стандартом. Не потому, что самое современное и оптимальное криптографическое решение, а потому, что ничего другого в то время (середина 1970-х годов) в открытом мире просто не знали. Шнайер: "В начале 70-х годов невоенные криптографические исследования были крайне редки. В этой области почти не публиковалось исследовательских работ. Большинство людей знали, что для своих коммуникаций военные используют специальную аппаратуру кодирования, но мало кто разбирался в криптографии как в науке..."

Вброс осуществлен... "Для оживления криптоанализа DES сделал больше, чем что-либо другое" (Б. Шнайер). Точно. Еще товарищ Сталин учил советских криптографов: "Читать всех, но наши переговоры и переписку читать никто не должен!" Вот и прочитаем DES!

Проблемы возникли с выполнением указаний товарища Сталина. А ответить на вызов обнаглевших янки необходимо. Вот мы и сделаем советский DES!

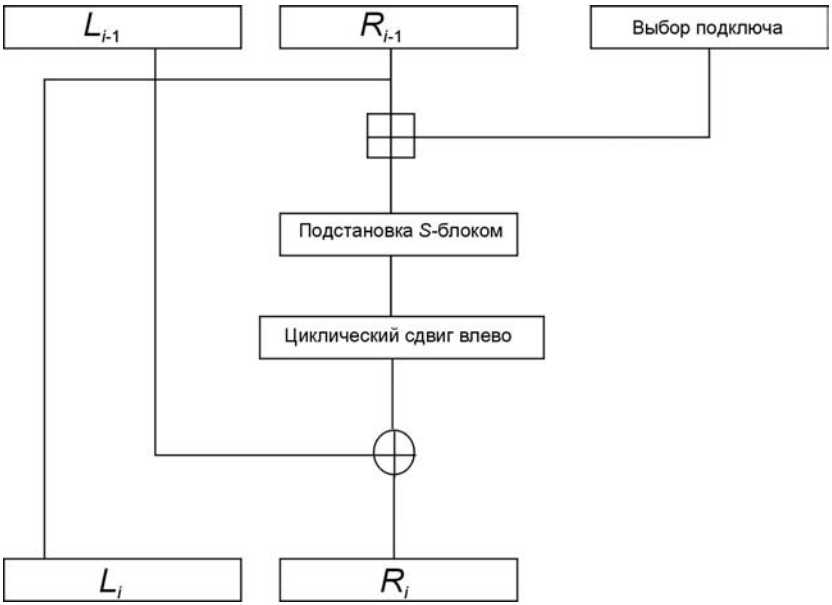

Рис. 5.4. Этап алгоритма ГОСТ 28147-89

Цитата из Шнайера: "Разработчики ГОСТ пытались достигнуть равновесия между безопасностью и эффективностью. Они изменили идеологию DES так, чтобы создать алгоритм, который больше подходит для программной реализации..."

Идеологию изменили, но как-то не очень сильно. Вместо привычных любой программе байт — по прежнему 32-битовые "кирпичи" да операции с битами. Шнайер: "Функция *f* проста. Сначала правая половина и *i*-ый подключ складываются по модулю 2^{32}.[1] Результат разбивается на восемь 4-битовых кусочков, каждый из которых поступает на вход своего *S*-блока[2]. ГОСТ использует 8 различных *S*-блоков... Каждый *S*-блок представляет собой пере-

[1] Значит, нужно при типовых операциях с байтами отслеживать и изменять знаки переноса, теряя на этом скорость!

[2] Опять лишние операции — разбивать на 4-битовые кусочки, потом опять собирать... Это отрицательно сказывается на скорости.

становку чисел от 0 до 15[1]... Выходы всех восьми S-блоков объединяются в 32-битовое слово, затем все слово циклически сдвигается на 11 битов".

Брюс Шнайер раздел "Криптоанализ ГОСТ" построил по принципу "найдите 5 отличий".

"Вот главные различия между DES и ГОСТ:

- DES использует сложную процедуру генерации подключей из ключей. В ГОСТ эта процедура очень проста.
- В DES 56-битовый ключ, а в ГОСТ — 256-битовый. Если добавить секретные перестановки *S*-блоков, то полный объем секретной информации ГОСТ составит примерно 610 бит.
- У *S*-блоков DES 6-битовые входы и выходы, а у *S*-блоков ГОСТ — 4-битовые входы и выходы. В обоих алгоритмах используется по восемь *S*-блоков, но размер *S*-блока ГОСТ равен одной четвертой размера *S*-блока DES.
- В DES используются нерегулярные перестановки, названные *P*-блоком, а в ГОСТ используется 11-битовый циклический сдвиг влево.
- В DES 16 этапов, а в ГОСТ — 32".

Может и пытались разработчики ГОСТ "достигнуть равновесия между безопасностью и эффективностью", но что-то уж какой-то странный они для этого избрали путь. Все равно, как в Москве с Ярославского вокзала на Казанский через Ясенево ехать. Ведь еще с середины 70-х годов было ясно, что достижение "равновесия между безопасностью и эффективностью" надо начинать с эффективности. В основу криптосхемы закладывать не биты, а байты, стараться не использовать совсем или использовать по минимуму операции с отдельными битами или группами бит, удобные для аппаратной реализации, но сильно тормозящие скорость программной реализации на типовом компьютере. А еще очень удобны (и криптографически весьма эффективны!) оказались подстановки. По сути — аналоги шифрдисков времен Бориса Хагелина. Но, опять же, не над алфавитом от 0 до 15, а над $Z/256$, над байтами. Меняй местами адрес и значение байта — готова подстановка! И никаких антикомпьютерных битовых операций, используем в криптосхемах только то, что легко и быстро можно реализовать с помощью типовых операций с байтами, заложенными в архитектуру процессора. Сделали высокоскоростную криптосхему, затем начинаем ее анализировать, обсчитывать, обосновывать безопасность, модернизировать. Но при одном условии: никакого насилия над любимыми процессором байтами! Без насилия безопасность получается ничуть не ниже. Ко взаимному согласию сторон. И велись на нашей Родине такие исследования не один год. Но нельзя было

[1] То есть шифрдиск времен Бориса Хагелина. Жаль, что не приделали еще "рукоятку, расположенную справа".

открывать нашей Главной Военной Тайны о том, что **мы знаем, что компьютер работает с байтами, а не с битами**...

А американцы опубликовали стандарт. Мы должны им ответить? Должны! Вот возьмем их стандарт, из его же деталей и блоков что-то такое сделаем, но своих тайн при этом не раскроем! Вот вам, заморские янки, наша открытая криптосхема!

Кто в середине 70-х годов прошлого века мог бы подумать, что через каких-то 25 лет появится такое чудо, как Internet, с его общедоступностью, открытостью, удобством. Что криптография перестанет быть монополией спецслужб, будет необходимой не только "коммунистическим диктаторам в России в их борьбе за власть" (Д. Кан), но и миллионам простых людей, общающихся друг с другом с помощью Internet, управляющих по электронной почте своим банковским счетом, создающим с помощью Internet удаленные рабочие места, позволяющие творчески трудиться не отходя от своего любимого домашнего компьютера, да и в сотнях других подобных случаях. И когда-нибудь люди обязательно поймут, что им нужна удобная и понятная криптография, та, которая до сих пор еще не вышла на свободу из сейфов спецслужб...

Глава 6

Электронная подпись

Хочется чего-то хорошего, изящного, оригинального. Красивых криптографических решений, нетривиальных методов, нетрадиционных подходов. И здесь одно из первых мест, безусловно, следует отдать У. Диффи и М. Хеллману, подметивших равенство $a^{xy} = a^{yx}$ и создавших в середине 1970-х годов криптографию с открытым распределением ключей. Об этом уже шла речь в *главе 4*, посвященной ключевым системам. Но это открытие привело к появлению еще одного, ставшего теперь обыденным, криптографического понятия — электронной подписи.

При передачи информации по каналам связи важное значение приобретает обеспечение имитозащиты сообщения, т. е. защиты, передаваемой по каналам связи информации от возможных умышленных или случайных изменений. Обычно защиту от случайных изменений (помех в канале связи) обеспечивает система коррекции ошибок, входящая в состав протокола обмена. Поскольку протоколы обмена общедоступны, то эта система не может защитить от попыток злоумышленника целенаправленно исказить передаваемое сообщение (например, увеличить сумму денежного перевода в платежном поручении). Для защиты от подобных целенаправленных искажений применяется система электронной подписи.

Вкратце суть электронной подписи сводится к следующему. К сообщению, которое необходимо подписать, строится проверочная комбинация, зависящая как от содержания сообщения, так и от секретного ключа, применяемого для подписи. Эта проверочная комбинация и будет электронной подписью сообщения. Основная ее особенность заключается в том, что для ее **построения** необходимо знать **секретный** ключ, но если подпись уже построена, то для **проверки** ее соответствия содержанию сообщения достаточно знать **открытый** ключ.

Таким образом, подписать сообщение может только обладатель секретного ключа, проверить же подлинность его подписи может любой абонент, знающий его открытый ключ.

Система электронной подписи нашла широкое применение при пересылке платежных документов. Клиент банка, обладающий собственным секретным ключом, подписывает на нем платежное поручение и посылает его вместе с подписью в банк. В банке хранится открытый ключ клиента, и, следовательно, имеется возможность проверить подлинность его электронной подписи. В такой системе клиент гарантирован от того, что кто-либо пошлет платежное поручение от его имени, а банк имеет возможность доказать клиенту, что платежное поручение пришло именно от него.

Это общая характеристика электронной подписи. Давайте более подробно поговорим обо всех связанных с ней деталях.

Интуитивно ясно, что процесс осуществления электронной подписи должен описываться с помощью каких-то формул, работающих с вполне определенными и конечными значениями. А сами сообщения могут быть короткими, средними, длинными и очень длинными. Следовательно, необходимо построить такой математический аппарат, такую функцию, которая каждому сообщению, независимо от его длины, ставила бы в соответствие некоторое значение, некоторый вектор одной и той же длины, который будет принимать участие в формулах электронной подписи. Такую функцию в криптографии принято называть хэш-функцией, а процесс ее вычисления — хэшированием.

Давайте задумаемся над тем, каким требованиям должна удовлетворять хэш-функция? Здесь рассуждения — число логические, в них практически нет никакой математики. В них также не рассматривается дальнейшая судьба хэш-функции — не важно, в каких конкретно алгоритмах она будет принимать участие.

1. Для простоты я бы сравнил хэш-функцию с ИНН — индивидуальным номером налогоплательщика — который наша налоговая инспекция пытается присвоить всем россиянам. Не важно количество денег у человека, количество выплачиваемых им налогов, его возраст, пол, пристрастия — любому человеку присваивается 10-значный ИНН. Так же и хэш-функция: любому файлу, любому сообщению электронной почты, в зависимости от содержания, присваивает тот или иной ИНН. Но *длина* такого ИНН не зависит ни от содержания, ни от типа файла. Длина для всех файлов и сообщений одна и та же. Иначе потом мы не сможем осуществить подпись.
2. Пользователь осуществляет подпись с помощью своего секретного ключа. Но проверка подписи будет осуществляться уже только с помощью открытого ключа, без знания секретного ключа. Следовательно, секретный ключ не должен принимать участие в вычислении хэш-функции, иначе подпись потом нельзя будет проверить.
3. Злоумышленник задается целью подделать подпись, причем наиболее простым способом, не вскрывая секретного ключа. Помните бесключевое

чтение? Каким будет аналог бесключевого чтения в системе электронной подписи? Подобрать ложное сообщение с точно такой же хэш-функцией. Поскольку в дальнейшем при осуществлении электронной подписи понятие "сообщение" и "его хэш-функция" неотличимы, то, подобрав ложное сообщение с идентичной хэш-функцией, злоумышленник просто подставит под него подлинную подпись и на приемном конце оно будет принято за истинное. Самое интересное состоит в том, что такое в принципе *всегда возможно*, поскольку сообщений, вообще говоря, бесконечно много, а количество значений хэш-функции ограничено. Внимательному читателю предлагаю самому подумать над тем, почему в качестве хэш-функции нельзя использовать, например, контрольную сумму.

4. Подобрать сообщение с идентичной хэш-функцией всегда возможно, весь вопрос в том, насколько это практически реально? Хэш-функция должна строиться с таким учетом, чтобы такая задача сводилась в конечном счете к большому перебору каких-то значений, неосуществимому за реальное время. Причем секретных ключей в хэш-функции нет! Что же тогда перебирается? Возможные варианты подписываемых сообщений.
5. А насколько подобранное сообщение устроит потенциального злоумышленника? Ведь ему может понадобиться вполне определенное ложное сообщение с точно такой же хэш-функцией. Возможно, что злоумышленника устроит не один, а несколько вариантов ложного сообщения. Какова мощность множества всевозможных ложных сообщений? Как его описать и построить?
6. Наверное те хэш-функции, которые каким-то образом позволяют описывать классы сообщений, имеющих одинаковые хэш-функции, следует признать криптографически слабыми. Примером слабой хэш-функции может служить контрольная сумма. А с помощью каких механизмов можно построить криптографически сильные хэш-функции? Наверное тех же, которые позволили построить криптографически качественные генераторы гаммы.

Вот примерный круг тех проблем и вопросов, которые должен задавать себе криптоаналитик при разработке алгоритма хэширования. В современных криптографических системах длина хэш-функции должна быть достаточно большой (около 64 байт) для того, чтобы обеспечить безопасность системы электронной подписи. Я бы сказал, что разработать хорошую функцию хэширования сложнее, чем хороший алгоритм шифрования. При шифровании вырабатываемая гамма накладывается на открытый текст за *один проход*, а при хэшировании количество проходов по хэшируемому тексту, как правило, *несколько*. И тогда особое значение приобретает скорость, с которой работают ваши криптографические алгоритмы, вычисляя хэш-функцию. Обзор различных известных методов хэширования можно найти в столько раз уже упомянутой ранее книге Брюса Шнайера (глава 18 "Однонаправленные хэш-функции").

Давайте поразмышляем, каким образом можно было бы построить хорошую хэш-функцию с использованием много раз уже упоминавшегося преобразования $x_{i+n} = \pi(x_i + x_{i+1} + ... + x_{i+n-1}) + k_i$? Вот один из возможных вариантов при $n = 8$.

- В начале процесса хэширования подстановка π принимает некоторое фиксированное значение, значения k_i — тоже некоторые фиксированные.
- Считываются два следующих друг за другом 256-байтных блока текста. Первый блок обозначим H_1, второй — H_2.
- Последние 8 байт блока H_1 выбираем в качестве x_0, x_1, ..., x_7, первые 20 байт H_2 — в качестве k_0, k_1, ..., k_{19}.
- Осуществляем преобразование — вычисляем x_{20}, x_{21}, ..., x_{27}.
- Складываем последние 8 байт блока H_1 с x_{20}, x_{21}, ..., x_{27}.
- В блоке H_1 по 8 адресам, определяемым x_{20}, x_{21}, ..., x_{27}, добавляем сами значения x_{20}, x_{21}, ..., x_{27}.

За этот этап алгоритма хэширования мы добились следующего: изменили последние 8 байт в блоке H_1 и еще какие-то случайно выбранные 8 байт. Некоторые из них могут попасть и на последний блок.

Всего в первом блоке 64 отрезка по 8 байт. Мы начали процедуру хэширования с последнего и за 64 шага проходим все, от последнего до первого. При этом место, с которого мы считываем во втором блоке значения k_0, k_1, ..., k_{19}, также циклически сдвигается, но уже вперед. В результате за 64 шага получаем полностью преображенный первый блок, в котором через значения k_0, k_1, ..., k_{19} замешаны и все знаки второго блока.

Из полученного преображенного первого блока строим новую подстановку π. Правила построения простые: начальной точкой отсчета при построении очередного элемента подстановки служит соответствующее значение элемента в блоке, значение увеличивается на 1 до тех пор, пока не появляется такое, которое не встречалось ранее. Таким образом, построенная подстановка фактически зависит от всех знаков в двух блоках.

Второй блок становится первым, в качестве второго считывается следующая порция из 256 байт текста, и процесс повторяется.

Весь процесс осуществляется за один проход, но после последнего этапа первые 256 байт файла считываются еще раз, но уже в качестве второго блока.

Значение хэш-функции нужной длины набирается на последнем шаге с помощью все того же преобразования, осуществленного несколько раз с подстановкой, выработанной за эти этапы и зависящей от всего текста.

Нетрудно заметить, что в таком подходе требуемый здесь лавинообразный эффект нарастания искажений (искажение одного знака дает искажение всего блока из 8 байт, эти 8 искажений также случайно разбрасываются по всему остальному тексту в 256-байтном блоке) в конечном итоге приводит

к тому, что вырабатываемые подстановки получаются совершенно различными, как бы случайными. А случайная подстановка является надежной защитой от разных методов взлома хэш-функции, да и не только хэш-функции. И еще одно преимущество: получив подстановку, зависящую от всего текста, легко построить хэш-функцию практически любой требуемой длины.

Теперь — о дальнейшей судьбе хэш-функции. Конечно же, основное ее предназначение — для электронной подписи. Для нее, собственно говоря, хэш-функция и разрабатывалась. Но бывают и иные, подчас самые неожиданные способы применения хэш-функции. И раз уж здесь зашла речь о хэш-функции, поделюсь одним из способов ее использования не по основному назначению.

Предположим, что вам необходимо создать систему для выработки секретных ключей для большого числа пользователей. О такой задаче мы уже упоминали в *главе 4*, посвященной ключевым системам. Есть некоторый базовый секретный ключ и короткие идентификаторы пользователей, например USR01, USR02 и т. д. Как построить алгоритм выработки уникальных секретных ключей для пользователей с помощью базового ключа и идентификаторов? Имея хэш-функцию, очень просто. С помощью базового ключа каждый идентификатор шифруется (с помощью блочного шифра), затем хэшируется, после чего значение хэш-функции шифруется и выдается в качестве индивидуального секретного ключа пользователя. В этом случае хэш-функция выступила в роли своеобразного "расширителя" количества индивидуальных знаков, содержащихся в коротком идентификаторе пользователя.

Вернемся к электронной подписи. Алгоритмов осуществления электронной подписи существует несколько, но отличия в них, как правило, в параметрах. Во-первых, в способе вычисления хэш-функции. Но здесь мы сознательно отделяем вычисление хэш-функции от осуществления подписи, поскольку цели и используемые методы при вычислении и использовании хэш-функции для осуществления или проверки подписи различные. Наиболее известны два алгоритма подписи: американский стандарт Digital Signature Algorithm (DSA) и наш ГОСТ Р 34.10 — 94. Но в обоих из них в качестве принципиального метода, с помощью которого осуществляется и проверяется подпись, используется метод Эль-Гамаля, основанный на задаче логарифмирования в конечных полях.

Рассмотрим алгоритм осуществления и проверки подписи по методу Эль-Гамаля. Но сначала — еще одна цитата из книги Шнайера.

"**Патенты.** ElGamal незапатентован. Но прежде чем двигаться вперед и реализовывать алгоритм, нужно знать, что PKP (Public Key Partners — компания из Саннивейла, Калифорния, обладающая лицензиями на наиболее известные методы открытой криптографии — ММ) считает, что этот алгоритм попадает под действие патента Диффи-Хеллмана. Однако срок действия

патента Диффи-Хеллмана заканчивается 29 апреля 1997 года, что делает ElGamal первым криптографическим алгоритмом с открытыми ключами, пригодным для шифрования и цифровых подписей и несвязанных в Соединенных Штатах патентами".

Итак, пусть:

- m — значение хэш-функции подписываемого сообщения;
- k — некоторое случайное число (про него ниже);
- $r = a^k$;
- x — секретный ключ;
- $s = k \times m + x \times r$.

Пара чисел (s, r) называется электронной подписью сообщения.

На практике, для *осуществления* электронной подписи по методу Эль-Гамаля, необходимо:

- вычислить m — хэш-функцию подписываемого сообщения;
- выработать некоторое случайное число k и вычислить $r = a^k$;
- имея секретный ключ x, вычислить $s = k \times m + x \times r$.

Проверка подписи, т. е. пары (s, r).

- вычислить m — хэш-функцию проверяемого сообщения;
- вычислить $r^m = a^{km}$;
- зная открытый ключ a^x, вычислить $(a^x)^r = a^{xr}$;
- перемножить $r^m \times a^{xr}$;
- вычислить a^s;
- если подпись верная, то $a^s = r^m \times (a^x)^r$, если нет, то равенство не выполняется.

Очень удачно пристроил Эль-Гамаль задачу логарифмирования в конечных полях. Осталось додумать чуть-чуть.

Если наличие в подписи хэш-функции понятно (подпись должна зависеть от содержания сообщения), то почему Эль-Гамаль включил туда случайное число k? Давайте-ка внимательно поглядим на формулы в подписи Эль-Гамаля и представим, что это случайное число k в них отсутствует или, более общий вариант, равно некоторой константе, которая известна и потенциальному злоумышленнику. Поглядим теперь повнимательнее на вторую часть подписи — число $s = k \times m + x \times r$. Что же получается, если k — известно, s — известно (вторая часть подписи), m — известно (хэш-функция), r — известно (первая часть подписи). А тогда становится известным и секретный ключ x! Следовательно, случайное число k криптограф должен сделать действительно случайным, которое будет неизвестно потенциальному злоумышленнику.

Теперь поразмышляем немного об организации процесса проверки электронной подписи. Дело в том, что осуществление электронной подписи происходит с помощью секретного ключа, а для проверки достаточно иметь только открытый ключ. Организация, осуществляющая проверку подписей, например, банк, проверяющий электронные подписи клиентов на платежных документах, может вообще не иметь никаких секретных ключей и использовать в банковских программах только открытые ключи клиентов. И тогда может возникнуть небрежное отношение к открытым ключам: зачем и от чего их защищать, они же открытые! Давайте это обсудим.

Да, без секретного ключа подделать электронную подпись в правильно созданных системах электронной подписи потенциальный злоумышленник не в состоянии. Но он опять постарается использовать обходные пути. А именно: попытается подменить открытый ключ клиента, используемый для проверки его подписи, на свой, и подписать фальшивый платежный документ на своем секретном ключе, использовав все прочие остальные реквизиты клиента. Следовательно, при проверке подписи должен быть какой-то механизм, позволяющий проверяющему убедиться в том, что используемый для проверки открытый ключ — подлинный. Что это может быть за механизм?

Нужна еще одна подпись. Подпись самого открытого ключа клиента. Но подписать его должно нейтральное лицо, которому доверяют обе стороны. Нейтральное в том смысле, что при возникновении каких-то недоразумений между банком и клиентом (например, если клиент заявляет, что не отправлял платежный документ, что открытый ключ, использовавшийся для проверки, не его) это лицо не будет заинтересованным ни для одной из сторон. Сам банк как заинтересованная сторона, как правило, не может быть таким независимым арбитром и, следовательно, не может подписывать открытые ключи клиентов. Такой независимый арбитр, мы будем иногда называть его еще менеджером системы безопасности, по взаимной договоренности сторон должен быть определен заранее. И стороны подписывают у него свои открытые ключи. Такую процедуру в дальнейшем мы будем называть *сертификацией* открытых ключей, а подпись менеджера СБ — *сертификатом* открытого ключа.

Подпись менеджера СБ необходимо проверять перед использованием открытого ключа для проверки подписи клиента. Следовательно, необходимо иметь открытый ключ менеджера СБ и опять же гарантировать его от подмены. Но такой открытый ключ уже один, а открытых ключей клиентов может быть очень много. Поэтому открытый ключ менеджера СБ целесообразно (но необязательно) хранить в секрете, вместе со своими индивидуальными секретными ключами. Главное — с помощью чисто организационных мер гарантировать защиту от подмены этого одного открытого ключа. А он уже будет гарантировать от подмены открытые ключи многих клиентов.

Но и это еще не все. Потенциальный злоумышленник, не имея возможности взломать систему электронной подписи или подменить открытый ключ

клиента, может попытаться подменить *программу, реализующую процедуру проверки подписи.* При расшифровании, например, это не имело никакого смысла, поскольку подмена программы расшифрования не давала возможности без знаний секретного ключа расшифровывать сообщения. А подмена программы проверки подписи может оказаться равносильной подделке подписи. Создается внешне точно такой же интерфейс, но результат проверки подписи все время выдается положительный — любая подпись верна! Поэтому следует задуматься и о том, как защитить от подмены саму программу, реализующую процедуру проверки подписи.

Получается, что и сама программа проверки подписи должна иметь электронную подпись. Иначе нет гарантии от подмены и подделок, за исключением чисто организационных мер, которые могут создавать значительные неудобства при эксплуатации. Защита от подмены программы проверки подписи — это уже проблема того, кто осуществляет проверку подписи. Лицо, подписывающее сообщение, к этому отношения не имеет: в этом случае при возникновении конфликта легко доказать свою правоту.

Следовательно, добавление электронной подписи к выполняемому файлу (exe-файлу, динамической библиотеке DLL) не должно влиять на выполнение их основных функций. Подпись необходимо добавлять в конец файла. А вот с подписью BAT-файла могут возникнуть проблемы. Наличие подписи в таком файле может вызвать выполнение неизвестных команд и сбои в работе программного обеспечения.

В случае, когда, например, банку приходится иметь дело с проверкой подписей многих клиентов, в открытых ключах клиентов должен быть какой-то идентификатор, уникальный для каждого клиента. Этот же идентификатор должен проставляться в подписываемое сообщение вместе с подписью. Мы будем называть его еще *меткой ключа* абонента. По метке ключа программа проверки подписи должна выбирать соответствующий открытый ключ из базы данных всех открытых ключей. Метки ключа не являются секретными, но они должны быть определены заранее с использованием какой-то простой и понятной логики. Нежелательно использование длинных меток ключа, поскольку эта информация каждый раз будет включаться в электронную подпись и передаваться по каналу связи вместе с подписанным сообщением.

Открытые ключи, применяемые для проверки подписи, вместе с подписью менеджера СБ могут содержать в себе также информацию о сроках их действия. Мы уже упоминали о том, что любой секретный ключ целесообразно периодически менять с тем, чтобы локализовать последствия при его возможной компрометации. Эта информация, вместе с меткой ключа, также должна защищаться электронной подписью менеджера.

И, наконец, последнее. Возможны ситуации, когда подписей будет несколько. Например, платежное поручение в банк обычно подписывают два лица —

директор и главный бухгалтер. Возможны ситуации, когда, кроме подписи, сообщение необходимо будет еще зашифровать с использованием системы с открытым распределением ключей и добавить некоторую служебную информацию об этом зашифровывании (от кого, кому предназначается открытый ключ, который необходимо использовать при расшифровании, и т. п.). Поэтому в подписи должны быть некоторые служебные символы, позволяющее легко выделять ее начало и конец, отделять ее от служебной информации, добавляемой при зашифровывании. Читатель, наверное, обратил внимание на включение открытого ключа, необходимого для расшифрования, в служебную информацию и, следовательно, становящегося доступным потенциальному злоумышленнику. Такое можно делать только тогда, когда для зашифровывания и подписи используются *разные* секретные ключи, выработанные из одного и того же базового секретного ключа пользователя. Тогда возможность подмены открытого ключа, используемого для расшифрования, не даст возможности потенциальному злоумышленнику подменить открытый ключ, используемый для подписи, и тем самым подделать подпись.

А самое последнее — это то, что файл с открытым ключом, используемым для проверки подписи, должен быть распечатан, распечатка подписана с помощью обычной подписи, на ней должна стоять обычная печать, и оператор должен иметь возможность в любой момент по этой распечатке визуально проверить истинность хранящегося у него открытого ключа. Но — это уже в самом крайнем случае.

Часть II

Криптографическое ядро и интерфейсная оболочка

Глава 7. Что понимать под криптографическим ядром и интерфейсной оболочкой?

Глава 8. Выработка секретного ключа

Глава 9. База данных открытых ключей

Глава 10. Процедура шифрования

Глава 11. Процедура подписи

Пора за работу. Создавать собственную криптографическую систему. Вариантов собственных систем может быть множество: система защиты баз данных, система защиты электронной почты, система защиты отдельных важных файлов, система защиты протоколов и прочая, прочая, прочая. Всех ситуаций, в которых может потребоваться криптографическая защита, описать невозможно. Но "правила хорошего криптографического тона" при их создании очень похожи: секретные ключи должны храниться действительно в секрете, следует избегать перекрытий гаммы и возможностей бесключевого чтения, при организации системы электронной подписи следует особое внимание обратить на сохранность открытых ключей, используемых для проверки подписи и т. д. Все те, о которых шла речь в предыдущей главе.

Вот давайте и проведем первое практическое занятие. Посмотрим, как можно применять теорию на практике.

Глава 7

Что понимать под криптографическим ядром и интерфейсной оболочкой?

Я бы, говоря программистским языком, сравнил криптографическое ядро с набором прерываний операционной системы. Например,

- **INT 10H** — прерывание, отвечающее за отображение информации на экране (Video and Screen Service);
- **INT 13H** — прерывание, отвечающее за обмен информацией с жестким диском и дискетами (Disk I/O Services, Floppy and Hard Disks);
- **INT 14H** — прерывание, отвечающее за обмен информацией с коммуникационными портами (Serial I/O Services, Communications Ports);
- **INT 16H** — прерывание, отвечающее за обмен информацией с клавиатурой (Keyboard I/O Services);
- **INT 17H** — прерывание, отвечающее за принтер (Printer I/O Services)

и некоторые другие.

Каждое прерывание — это маленькая программа, автоматически загружаемая в оперативную память при включении компьютера и постоянно находящаяся там. В зависимости от параметров, подаваемых на вход такой программы, она может выполнять те или иные простейшие функции. Так, например, прерывание 10H при одних параметрах может устанавливать на экране позицию и размер курсора, при других — высвечивать на экране точку в определенном месте и определенного цвета и тому подобное. Все программы, использующие операционную систему, прямо или косвенно (чаще, конечно же, косвенно) используют эти прерывания. Набор прерываний я бы назвал *ядром* операционной системы. Все остальное, в том числе и современные сложнейшие программные комплексы, — это уже *интерфейсная оболочка*, работающая с этим ядром. Здесь вполне понятная и ясная логика: от простого к более сложному. На первом этапе появления персональных компьютеров создавалось ядро операционной системы в виде наборов прерываний. Это простейшие, но основополагающие для дальнейшего развития программы. Их очень хорошо отладили, протестировали, провели

всю необходимую оптимизацию, задокументировали, и после этого они стали почти незаметны. Программисты начали использовать их как нечто само собой разумеющееся. Большинству современных программистов, пишущих свои программы на MS Visual C++ , Borland C++, Delphi и других языках, уже, как правило, нет никакого дела до того, например, с помощью каких процедур нажатая на клавиатуре клавиша отображается на экране или каким путем информация из оперативной памяти попадает на жесткий диск.

Аналогичным путем хотелось бы строить и нашу криптографическую систему. Создать некоторый набор простейших криптографических процедур, хорошо их отладить, протестировать, оптимизировать, сложить в какую-то динамическую библиотеку, OBJ-файл, LIB-файл или в любое другое удобное место (возможно, даже "прошить" в ПЗУ) и после этого только использовать как нечто само собой разумеющееся. Начать разработку криптографической системы с *криптографического ядра*, на которое затем будет наслаиваться интерфейсная оболочка. Хотя, конечно, такое разделение весьма условно, провести четкую границу между ядром и оболочкой нельзя, это, часто, дело вкуса и привычек человека, взявшегося создавать собственную систему защиты информации. Но в дальнейшем мы будем относить к криптографическому ядру только самые необходимые криптографические процедуры, полагая, что все остальное входит в понятие интерфейсной оболочки.

По собственному опыту, в качестве первой криптографической процедуры могу предложить процедуру `CryptBlock`. Помните *главу 5* про блочные шифры? Вот процедура `CryptBlock` и будет программно реализовывать один переход в подстановке большой степени, формируемой блочным шифром.

Возьмем не раз уже упоминавшееся ранее преобразование

$$x_{i+n} = \pi(x_i + x_{i+1} + \ldots + x_{i+n-1}) + k_i$$

и посмотрим на него повнимательнее. Наша задача — найти в нем оптимальное соотношение между "безопасностью и эффективностью". Значение n примем равным 8 и попытаемся порассуждать на тему, как из такого и ему подобных преобразований выбрать наиболее безопасное и эффективное. Нельзя ли в нем сократить количество операций сложения?

С точки зрения безопасности это преобразование должно быть регулярным, т. е. не допускать того случая, когда два различных блока переходят на выходе в один и тот же. Примем этот постулат за аксиому и все нерегулярные преобразования сразу же откинем как заведомо непригодные. Тогда как минимум один элемент — x_i — под подстановкой должен заведомо присутствовать. Для регулярности этого оказывается *достаточно*. Можете проверить. Может быть и выбрать такое преобразование в качестве блочного шифра? Давайте выпишем уравнение и посмотрим на него внимательнее.

$$x_{i+8} = \pi(x_i) + k_i.$$

Рассмотрим два блока

$$x_0, x_1, ..., x_7,$$

$$y_0, y_1, ..., y_7,$$

причем такие, у которых $x_0 = y_0$, $x_1 = y_1$, ..., $x_6 = y_6$, $x_7 \neq y_7$. Тогда, независимо от подстановки π, имеем: $x_8 = y_8$, $x_9 = y_9$, ..., $x_{14} = y_{14}$, $x_{15} \neq y_{15}$, $x_{16} = y_{16}$.

Таким образом, на *специально подобранной* паре блоков открытого текста зависимости переходят в блоки шифртекста. Одной переменной под подстановкой π оказывается явно недостаточно с точки зрения безопасности. Переменных должно быть как минимум две, причем вторую, как это видно из приведенного выше примера, следует выбрать x_{i+7}.

$$x_{i+8} = \pi(x_i + x_{i+7}) + k_i.$$

Возьмем для него пару блоков такую, у которых $x_0 = y_0$, $x_1 = y_1$, ..., $x_6 \neq y_6$, $x_7 = y_7$. Имеем: $x_8 = y_8$, $x_9 = y_9$, ..., $x_{13} = y_{13}$. Но дальше уже эти зависимости кончаются, свое дело делает вторая переменная.

Напоследок заметим, что, как мне кажется, и двух переменных маловато. 3—4 — уже куда ни шло. Да и расположить их надо аккуратно, не где попало. Но на этом закончим с формулами и математикой, а пойдем дальше создавать программное криптографическое ядро.

Описанный выше блочный шифр на C программируется в 2—3 строчки. Параметры на входе — начальное заполнение x_0, x_1, ..., x_8, подстановка π да значения k_0, k_1, ..., k_T. Само значение T (количество отрабатываемых циклов) также может быть параметром. Результат работы — значения x_T, x_{T+1}, ..., x_{T+7}. Процедура легко может быть реализована с использованием быстрой (регистровой) памяти.

Не могу утверждать, что достиг абсолютных истин в программировании. Возможно, читателю удастся предложить более быструю реализацию процедуры `CryptBlock`, чем та, которая получилась у меня. Но чтобы было с чем сравнивать, приведу свою программу для проверки скорости работы процедуры `CryptBlock` и результаты ее работы при $n = 8$, $T = 40$.

Исходные данные: процессор — Intel Celeron 800 MHz; язык программирования — Borland C++ Builder, ver. 3.0, операционная система — Windows 98.

Программа для определения скорости работы процедуры CryptBlock

```
#include <vcl.h>
#include <stdio.h>
#include <stdlib.h>
USERES("SpeedCB.res");
```

```
USELIB("Mcs_std.lib");

void CryptBlock(unsigned char *x, unsigned char *Pi,
                unsigned char *K, char KLength);

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
unsigned int i, j;
unsigned char Pi[256]; // Подстановка Pi
// Вспомогательный массив, используемый при построении подстановки
bool PiCnt[256];
unsigned char empt;   // Вспомогательная переменная
unsigned char x[8];
unsigned char k[40];
char buff[128];
TTime * Time;
TTime * TimeStart;
TTime * TimeEnd;
unsigned short Hour, Min, Sec, MSec;

// Задаем начальную точку генератора случайных чисел
    srand(27531);
    memset(PiCnt, 0, 256); // Обнуление контрольного массива

// Сначала строим подстановку Pi
    for(i = 0;i<256;i +  + )
    {
        empt = (unsigned char)rand();
        while(PiCnt[empt] =  = true)empt +  + ;
        Pi[i] = empt;
        PiCnt[empt] = true;
    }
// Проверяем, что получилась действительно подстановка

    memset(PiCnt, 0, 256);
    for(i = 0;i<256;i +  + )PiCnt[Pi[i]] = true;
    for(i = 0;i<256;i +  + )
        if(PiCnt[i] =  = false)
        {
```

```
            MessageBox(0, "Error!", "", MB_ICONERROR);
            break;
        }

    if(i! = 256)return(-1); // Если была ошибка

// Начальные заполнения
    for(i = 0;i<8;i +  + )x[i] = (unsigned char)rand();
    for(i = 0;i<40;i +  + )k[i] = (unsigned char)rand();
    Time = new TTime();
// Запоминаем время в начале эксперимента
    TimeStart = new TTime(Time->CurrentTime());

    for(i = 0;i<10000;i +  + )
        for(j = 0;j<10000;j +  + )CryptBlock(x, Pi, k, 40);
// Запоминаем время в конце эксперимента
    TimeEnd = new TTime(Time->CurrentTime());
    delete Time;
    Time = new TTime(TimeEnd->Val — TimeStart->Val);
    Time->DecodeTime(&Hour, &Min, &Sec, &MSec);

    sprintf(buff, "\
    N = 10**8\n\
    Time: %d:%d:%d.%d", \
    Hour, Min, Sec, MSec);
    MessageBox(0, buff, "", MB_OK);

    return 0;
}
```

Рис. 7.1. Результат работы программы проверки скорости CryptBlock

Как показано на рис. 7.1, за 134 секунды программа смогла выполнить 10^8 циклов, т. е. в среднем шифровала около 730 000 блоков в секунду в стандартной среде Windows при наличии в оперативной памяти других

вспомогательных программ. Скорость шифрования составляет примерно 5, 8 Мбайт/сек. Я думаю, что это еще не предел.

С `CryptBlock` закончим. Но, естественно, в наше криптографическое ядро добавим процедуру `DecryptBlock`, реализующую обратные преобразования. Формулы для обратных преобразований уже приводились ранее в *главе 5*. Но эта процедура будет использоваться значительно реже. Дело в том, что основные объемы информации мы будем шифровать с помощью шифра гаммирования, используя в качестве генератора гаммы блочный шифр. В нем процедура `DecryptBlock` не нужна. Но в некоторых, я бы их назвал "точечных", случаях (дальше о них еще будет сказано) удобнее оказывается использовать `CryptBlock` напрямую. Там уже потребуется `DecryptBlock`.

Следующей занесем в наше криптографическое ядро процедуру `Crypt`. Как уже отмечалось выше, для шифрования мы будем использовать шифр гаммирования, а гамму вырабатывать с помощью блочного шифра. Процедура `Crypt` будет отличаться от `CryptBlock` тем, что в ней результат применения `CryptBlock` будет использоваться в качестве 8 знаков гаммы для наложения на 8 знаков открытого текста, после чего `CryptBlock` повторяется снова и так до тех пор, пока не будет наложена гамма на весь открытый текст.

Аналогично, добавим в криптографическое ядро процедуру `Decrypt`, отличающуюся от `Crypt` лишь тем, что знаки гаммы в ней не добавляются, а вычитаются из текста. Эта процедура предназначена для расшифрования. Заметим, что в `Decrypt` будет по-прежнему использоваться процедура `CryptBlock`. Процедура `DecryptBlock` там не нужна.

Из этих процедур в дальнейшем будем строить различные генераторы гаммы, использовать их как по прямому, так и по косвенному назначению. Об одном таком косвенном назначении уже шла речь в предыдущей главе — при вычислении хэш-функции. Другое косвенное назначение — выработка отдельных секретных ключей для различных целей из одного и того же базового секретного ключа, создание различных проверочных комбинаций, по которым нельзя было бы определить сам секретный ключ, для создания случайных чисел, маркантов и т. п. Про все это еще пойдет речь при описании различных интерфейсных оболочек.

Теперь займемся процедурами, необходимыми для осуществления операций с открытым распределением ключей.

Здесь ограничимся перечнем процедур и кратким описанием их предназначения.

- `ExpA` — процедура для вычисления значения a^x;
- `ExpCom` — процедура для вычисления общего ключа a^{xy};
- `Sign` — процедура для вычисления пары (s, r) по методу Эль-Гамаля;
- `TestSign` — процедура проверки равенства $a^s = r^m \times (a^x)^r$ при проверке подписи по методу Эль-Гамаля.

В двух последних процедурах предполагается, что все остальные переменные, требуемые для осуществления и проверки подписи (хэш-функция, секретный ключ, случайное число, открытый ключ), подаются на вход в качестве параметров — см. *главу 6*.

Основание a во всех этих процедурах одно и то же. Для реализации этих процедур могут использоваться все различные способы: операции с простыми числами, операции с многочленами в поле $GF(2^n)$, операции с эллиптическими кривыми. Более подробно останавливаться на способах реализации этих алгоритмов мы здесь не будем, интересующемуся читателю я могу порекомендовать опять же книгу Б. Шнайера.

Можно привести некоторые другие примеры подобной ("ядро" — "интерфейсная оболочка") организации системы защиты информации. Так, на сайте МО ПНИЭИ **www.security.ru** предложена единая технология подхода к построению криптографически защищенных информационных и телекоммуникационных сетей (ИТКС) — Технология "Витязь". Одной из компонент этой технологии являются "библиотечные модули — это объектные модули или модули командной строки, предназначенные для вызова криптографических функций непосредственно из приложения, осуществляющего обработку конфиденциальной информации. Для осуществления криптографических операций над файлами или блоками памяти наиболее удобным и эффективным может быть использование не отдельных специализированных приложений, а криптографических библиотек, встроенных в прикладное программное обеспечение (ППО), непосредственно осуществляющее подготовку, обработку или передачу информации".

Другим примером является пакет "Домен-К", выпускаемый компанией ИНФОТЕКС (**www.infotecs.ru**). Вот выдержка из приведенного на сайте описания системы:

> ***Основные характеристики криптографического ядра "Домен-К"***
>
> Средства криптографической защиты информации (СКЗИ) "Домен-К" предназначены для использования в государственных и коммерческих структурах путем встраивания в прикладное программное обеспечение, реализующее хранение, обработку и обмен персональных данных, конфиденциальной, служебной, коммерческой и др. информацией, не составляющей государственной тайны... СКЗИ "Домен-К" предназначены для работы с использованием ПЭВМ с процессорами типа Intel Pentium под управлением операционных систем Windows 95/98ME, Windows NT 4.0/2000. ПО СКЗИ "Домен-К" представляет собой пакет библиотек и программ, реализующих ключевой центр, прикладной криптографический интерфейс, криптографический драйвер.

В обоих этих примерах в качестве алгоритмов шифрования и подписи используются только алгоритмы ГОСТа. А вот компания "ЛАН-Крипто" (**www.lancrypto.com**) предлагает и собственные разработки быстрых алгоритмов шифрования. Но в конечном итоге сам пользователь, руководствуясь

теми или иными соображениями, принимает решение об использовании конкретного криптографического ядра. Функциональные возможности у всех примерно одни и те же: осуществление шифрования, подписи, работа с секретными и открытыми ключами. Различие — в скорости, сложности и особенностях реализации, организации интерфейса, стоимости. Однако в России почему-то сложилась такая ситуация, что ФАПСИ сертифицирует только те криптографические средства, в которых алгоритмы шифрования и подписи соответствуют ГОСТу. Почему возникла такая ситуация — это предмет особого разговора. Но у многих специалистов сложилось мнение, что это неправильно.

"Российский ГОСТ был принят с большим количеством ошибок и уже не соответствовал современным требованиям, — говорит президент ассоциации "РусКрипто" Алексей Волчков. — Во всем мире давно отказались от кулуарной разработки стандартов, которые будут использоваться в том числе для коммерческих целей. Последний пример такого рода — разработка американским Агентством национальной безопасности стандарта DES. С тех пор на протяжении 30 лет ни один стандарт ни на подпись, ни на защиту данных в США не разрабатывался спецслужбами. Он разрабатывался органами стандартизации с привлечением широкого круга экспертов, в том числе из других стран".

Вернемся к нашему криптографическому ядру. В нем мы постарались оставить минимальный набор криптографических процедур, в которых не используются алгоритмы типа ГОСТ, не имеющие ясной и понятной логики и усложняющие программную реализацию. Прежде чем приступать к созданию различных интерфейсных оболочек, давайте задумаемся над тем, какими возможностями будет обладать потенциальный злоумышленник при осуществлении попытки взлома криптографической системы. Отсюда будут вытекать основные *требования* к нашей криптографической системе, на 95—99% состоящей из интерфейсной оболочки.

- Полностью известны все используемые алгоритмы криптографических преобразований, потенциальный злоумышленник имеет точную копию программной реализации криптографической системы, может задавать для нее любые секретные ключи, подавать на вход (при заданных им самим секретных ключах) любые тексты и получать результаты их обработки. Ему известны все используемые при шифровании и подписи форматы, с помощью которых формируется результат обработки текста, все служебные и вспомогательные символы, правила ввода-вывода и отображения обрабатываемой информации.

  Это вполне ясное и понятное требование. Многие современные криптосистемы общедоступны, используемые в них алгоритмы либо открыто заявляются компанией-изготовителем, либо их можно вычислить с помощью анализа текста программной реализации, путем целенаправленного "тестирования" системы, подавая на ее вход специальные тестовые

тексты. Из наиболее часто открыто заявляемых алгоритмов выделяются DES и ГОСТ.

Но, по большому счету, широко публиковать криптографические алгоритмы, тем более с контрольными примерами, как ГОСТ, неправильно. Используемый алгоритм — это один из потенциальных рубежей защиты от действий хакера. И если ему придется восстанавливать алгоритм по тексту exe-файла или с помощью тестирования "черного ящика", то на это потребуются определенные усилия и время. Злоумышленник не обязательно может быть весьма квалифицированным и упорным. Необходимость восстановления алгоритма может исчерпать его ресурсы до того, как он приступит к непосредственному взлому. Поэтому при разработке криптосистемы я бы советовал рассчитывать на то, что криптографические алгоритмы известны, но без особой надобности их конкретные параметры не публиковать и не распространять. Только общие данные, подтверждающие надежность алгоритма.

- Потенциальный злоумышленник имеет доступ к шифртексту, может осуществлять его перехват, целенаправленное искажение, знает вероятностные характеристики открытого текста, наиболее часто употребляемые в нем стандарты, служебные символы, может устанавливать соответствующие им отрезки шифртекста.

  Портрет злоумышленника может быть разным. Но в современных условиях, при использовании средств криптографической защиты в роли злоумышленника может выступать человек, недовольный своей заработной платой. Мотивы его действий могут быть самые различные: от банального "назло начальству" до попыток выдать коммерческие секреты конкурентам. Такой человек может, например, иметь доступ к базе данных, в которой сведения хранятся в зашифрованном виде. Общее представление о характере и особенностях этой базы данных человек имеет, сам раньше работал с ней, а теперь хочет вытянуть из нее новые данные, ставшие ему недоступными. Появляется даже такое понятие: *защита от своих.*

- Потенциальный злоумышленник имеет некоторые возможности тестирования криптографической системы на реальном секретном ключе, не обладая при этом самим секретным ключом.

  Под тестированием криптографической системы мы будем понимать подачу на вход некоторых *специально подобранных* текстов и получение результатов их обработки. Ранее, в примерах с блочными шифрами, мы уже сталкивались с тем, что для некоторых близких блоков зависимости в блоках открытого текста сохраняются и переходят в блоки шифртекста. Но вероятность появления близких блоков при случайном и равновероятном выборе может быть весьма незначительной, поэтому потенциальный злоумышленник предпримет все возможные действия для того,

чтобы получить *искусственным путем* результат работы криптосистемы на близких блоках. Он, например, может направить от своего имени в банк два близких по содержанию платежных документа и перехватить результаты их зашифровывания. Он, если сам работает в банке, может попытаться создать сбойную ситуацию и под предлогом наладки подать на вход криптосистемы специально подобранную тестовую комбинацию.

В конечном итоге, на каждую подобную ситуацию должна даваться точная математическая оценка трудоемкости определения секретного ключа. Если обнаруживается, что с появлением у злоумышленника новых возможностей эта трудоемкость снижается, то такую криптографическую систему следует признать некачественной. Например, если при появлении возможности тестирования криптосистемы удается обнаружить методы определения секретного ключа, значительно менее трудоемкие, чем все известные методы без такой возможности, то эта система может оказаться ненадежной. В идеальном случае вопрос нужно ставить так: можно осуществлять любые манипуляции с открытым или шифрованным текстом, какова в этих условиях оценка стойкости? И она должна быть примерно такой же, что и без этих условий. Некоторый аналог размышлений Шеннона об абсолютно стойком шифре.

Но практически у любой оценки трудоемкости определения секретного ключа есть еще одна характеристика — *надежность*. Всегда можно предложить, например, такой метод: выбираем наугад любой ключ, и может так сильно повезет, что этот ключ окажется истинным. Тогда трудоемкость такого определения можно считать за 1 элементарную операцию, но понятие "сильно повезет" очень неопределенное. Если переходить к терминологии теории вероятностей, то оно равносильно вероятности того, что выбранный ключ окажется истинным при условии, что выбор осуществлен случайно и равновероятно из множества всевозможных ключей. Если, к примеру, число всевозможных ключей равно 10^{100}, то надежность такого метода будет 10^{-100}.

В *главе 2* мы уже касались вопроса о том, что можно считать гарантированной оценкой трудоемкости перебора всевозможных ключей. Теперь же, аналогично, подумаем над тем, какую надежность считать предельно допустимой.

Каждый человек всегда подвергается некоторым потенциальным опасностям. У самолета, в котором он летит, может отказать двигатель; у автомобиля, на котором он едет, в самый неподходящий момент могут отказать тормоза; здание банка, в которое он зашел, может быть в этот момент разрушено землетрясением; авария на заводе недалеко от города, где он живет, может привести к выбросу в атмосферу ядовитых отходов и отравить весь город и масса других подобных опасностей. Угадывание злоумышленником секретного ключа, с помощью которого человек, например, управляет своим счетом в банке, можно также отнести к чрезвычайным происшествиям. Оно, в принципе, всегда возможно, весь вопрос в том, какую вероятность считать

опасной? Здесь уже чисто житейские соображения берут верх над математическими расчетами. Каждый выбирает себе сам предельно допустимый уровень опасной вероятности. Один человек может считать опасным событие, если оно может произойти в одном случае из 10 000, другой — из 100 000, третий — из 1 000 000. Но все-таки есть разумная граница. Если ее перейти, то вероятность каких-то других чрезвычайных происшествий, для человека гораздо более опасных, чем компрометация секретного ключа, будет выше. Давайте примем ее за 10^{-10} — один шанс из 10 миллиардов. Если вероятность меньше — считаем, что такое в реальной жизни просто невозможно.

Таким образом, все методы определения секретного ключа, надежность которых меньше, чем 10^{-10}, считаем неопасными — такого не может быть, потому что не может быть никогда!

Нетрудно видеть, что, к примеру, если множество всевозможных секретных ключей составляет 10^{100}, то для того, чтобы при тотальном переборе получить метод с надежностью, не меньшей, чем 10^{-10}, нам потребуется перебор в среднем примерно 10^{90} вариантов.

Теперь, возвращаясь к нашей криптографической системе, заметим, что дополнительные манипуляции с открытым или шифрованным текстом не должны увеличивать надежность различных методов взлома системы. Методы, которые без подобных действий (специального подбора текстов) мы считали неопасными из-за их низкой надежности, не должны становиться опасными при допущении возможности специального подбора текстов.

А теперь спустимся ближе к реальной жизни. Потенциальный злоумышленник, убедившись, что классическими криптографическими методами определить секретный ключ нельзя, попытается его украсть. Скопировать его с оставленной без присмотра ключевой дискеты (Touch Memory, Smart Card, etc.) или вытащить из оперативной памяти компьютера при работе с криптографической системой. Вторая опасность намного более серьезная, чем первая.

Простейший метод действий злоумышленника — копирование содержимого всей оперативной памяти при работе криптосистемы, после чего — полный перебор всего, что скопировал. Объем оперативной памяти современного компьютера — до 1 гигабайта. Это — всего 10^9 байт. По криптографическим меркам — ничтожная величина. И способ защиты от такого метода может быть только один — организационные меры. Нельзя допускать копирования злоумышленником оперативной памяти при работе криптографической системы. Нельзя допускать появления на вашем компьютере различных программных закладок и "криптовирусов", стремящихся вытащить и спрятать секретный ключ. Если организационных мер нет, то никакая криптографическая система не обеспечит гарантированной стойкости. Если нет строгого

ограничения допуска к компьютеру с криптосистемой, то это — потенциальная угроза безопасности. Подключен сетевой кабель — угроза считывания оперативной памяти. Подключен модем — угроза считывания памяти. Насколько используемые сетевые протоколы гарантируют невозможность сканирования памяти посторонним пользователем? Какова их оценка стойкости? Не получится ли так, что гарантированная и просчитанная оценка стойкости криптографической системы будет перечеркнута совершенно непонятной и в большинстве случаев весьма низкой защитой от несанкционированного доступа, реализованной в сетевом протоколе?

Масса проблем и вопросов. Не отказываться же от компьютерных сетей! Но в любом случае разработчик и пользователь криптографической системы должны очень четко представлять себе все возможные опасности и всяческими путями стремиться их минимизировать. Не допускать записи секретного ключа в реестр Windows!

Глава 8

Выработка секретного ключа

Создание интерфейсной оболочки начнем с программных процедур, предназначенных для выработки индивидуальных секретных ключей. Давайте сначала обсудим, что будет представлять из себя индивидуальный секретный ключ пользователя.

В *главе 4* мы уже немного рассуждали на эту тему. Секретный ключ — это некоторая случайная последовательность, индивидуальная для данного пользователя, не подлежащая для разглашения посторонним. С ее помощью пользователь получает различные криптографические возможности: вырабатывает уникальную гамму и шифрует текст, подписывает сообщения, получает доступ к различным ресурсам и т. п. И этот ключ должен как-то *храниться* у пользователя. Причем храниться так, чтобы оставаться в секрете от посторонних. А наиболее простой способ хранения — это пароль, запоминаемый пользователем. Из пароля программа вырабатывает случайную последовательность необходимой длины и формата, которую затем использует в криптографической системе.

Несмотря на кажущуюся надежность, это пример одного из *наиболее плохих* способов организации ключевой системы. Системы, создающей видимость защиты, успокаивающей пользователя, настраивающей его на потерю бдительности. Настоятельно рекомендую не пользоваться в серьезных приложениях криптографическими системами, у которых в качестве секретного ключа пользователя запрашивается только пароль! Несмотря на то, что множество *всевозможных* ключей, которые будет использовать такая система, может быть достаточно большим (например, $2^{512} = 10^{153}$), число *допустимых* ключей, т. е. таких, которые могут быть реально выработаны, не превосходит множества всевозможных паролей.

Человек должен помнить пароль. Возникает естественное желание придумать пароль покороче, попроще, такой, который легко будет запомнить. Включить в него, например, дату своего рождения, номер телефона, счет в банке, имена жены, детей, их даты рождения. Все это легко перебирается!

Все эти дни рождения и телефоны легко вычисляются потенциальным злоумышленником. Гарантированная криптографическая защита с помощью пароля подменяется кроссвордом на сообразительность, небольшой головоломкой из научно-популярного журнала для детей и юношества. Если же вы не используете эти даты, а берете какое-нибудь слово, то посмотрите внимательнее в раздел "Правописание" редактора Microsoft Word. В нем уже есть словари, содержащие большой словарный запас и автоматически исправляющие ошибки. Ведь хакер простым перебором всех слов из этого уже готового и чрезвычайно распространенного словаря сможет легко вычислить и ваше "секретное слово".

А нестандартное или бессмысленное слово — риск его потом забыть. Поэтому нормальный человек, чтобы не создавать себе лишних проблем, возьмет и запишет его где-нибудь в своей записной книжке. Или с обратной стороны своего рабочего стола. Гарантированная криптографическая защита опять же сводится к "ключу под ковриком у входной двери". Самообман, не более.

Для гарантированной защиты нужен и гарантированный секретный ключ. Действительно случайная последовательность, полный перебор которой физически невозможен. Не содержащая зависимостей, все возможные ключи в ней действительно должны быть допустимыми, т. е. появление каждого ключа должно быть реально возможно.

Многие авторы любят пофилософствовать на тему генерации действительно случайных последовательностей. Заглянем опять к мистеру Шнайеру.

"Теперь мы вторгаемся в область, принадлежащую философам. Существует ли такая вещь, как случайность? Что такое случайная последовательность? Как узнать, что последовательность случайна? Является ли "101110100" более случайной, чем "101010101"? Квантовая механика убеждает нас в том, что в реальном мире существует настоящая случайность. Но как сохранить эту случайность в предопределенном мире компьютерных микросхем и конечных автоматов?"

Пофилософствовав немного, Брюс Шнайер делает весьма материалистический вывод: "В сторону философию!" Еще немного — и можно дойти до "бытие определяет сознание", а затем идти сдавать экзамен кандидатского минимума по марксистско-ленинской философии. Нет, лучше уж создавать генератор случайных чисел своими руками. Полностью согласен в этом с г-ном Шнайером.

С чего начать? С отсева заведомо плохих генераторов случайных чисел. На эту тему мы уже рассуждали в *главе 4*, рассматривая случайную, а точнее *псевдослучайную* последовательность, вырабатываемую встроенной в язык программирования C функцией `RAND`. Но чтобы больше не мелочиться с одним генератором из C, отсеем их сразу оптом, большой партией, процитировав, опять же, Брюса Шнайера.

"Генератор случайных чисел встроен в каждый компилятор, обычный вызов функции. Почему бы не использовать его? К сожалению, эти генераторы случайных чисел почти наверняка недостаточно безопасны для криптографии и, возможно, даже не совсем случайны. Большинство из них весьма плохи.

Генераторы случайных чисел на самом деле совсем не случайны, потому что им и не нужно быть такими. Для большинства приложений, например для *компьютерных игр* (курсив мой — М.М.), требуется так мало случайных чисел, что их неслучайность вряд ли будет заметна. Однако криптография очень чувствительна к свойствам генераторов случайных чисел. Примените плохой генератор, и у вас появятся таинственные корреляции и странные результаты. Если ваша безопасность зависит от генератора случайных чисел, таинственные корреляции и странные результаты являются абсолютно не тем, чего бы вы желали добиться.

Проблема в том, что генератор случайных чисел не создает случайной последовательности. Он, возможно, не выдает ничего, даже отдаленно напоминающего случайную последовательность. Конечно, невозможно создавать на компьютере что-то по-настоящему случайное. Дональд Кнут приписывал фон Нейману следующие слова: "Каждый, кто занимается арифметическими методами получения случайных чисел, определенно грешит". Компьютеры — это детерминированные бестии: закладывается известный материал, выполняются полностью предсказуемые действия, и что-то отличное выползает с другого конца. Подача одного и того же на вход в двух различных случаях приведет к одному и тому же результату. Заложите одинаковые исходные данные в два идентичных компьютера, и оба они подсчитают одно и то же. Компьютер может находиться только в ограниченном числе состояний (очень большом, но все же ограниченном), и выдаваемый результат всегда будет строго определяться исходными данными и текущим состоянием компьютера. Это значит, что любой генератор случайных чисел на компьютере (по меньшей мере, на конечном автомате), по определению, периодичен. А все, что периодично, по определению, предсказуемо. А все, что предсказуемо, не может быть случайным. Для настоящего генератора случайных чисел нужно подавать на вход что-нибудь случайное, компьютер же не может обеспечить это требование".

В России еще как может! Только на вход надо подать не какой-то непонятный "известный материал", а, например, прапорщика, находящегося на ночном дежурстве. Что будет делать прапорщик, поданный на вход компьютера? Всю ночь на нем играть! В стрелялки, бродилки, страшилки, QUEST, QUAKE, STRIKE. И пока все их не пройдет, от компьютера его ни за что не оттащить. Неоднократно проверено на практике. Ну а где же генератор случайных чисел?

Сам пользователь. Вернее, те моменты времени (с точностью до сотых долей секунды), в которые он осуществляет нажатие на клавиши или щелкает по кнопке мыши в моменты своего компьютерного экстаза. Как он играет —

неважно, в какую игру — тоже неважно, результат игры не имеет никакого значения. Главное, что в процессе игры пользователь нажимал на клавиши или щелкал мышью, создавал *случайные события*. По моменту реакции пользователя на задаваемые компьютерной игрой ситуации строится действительно случайная последовательность. "Детерминированную бестию" — компьютер — дополнила недетерминированная бестия — человек, со своей непредсказуемой логикой и реакцией. А зная, как в России любят компьютерные игры, смею утверждать, что это практически неограниченный источник действительно случайных чисел.

Задача разработчика криптографической системы сводится только к тому, чтобы научиться отлавливать моменты времени, в которые осуществлялись нажатия на клавиши или кнопки мыши. Да, пожалуй, еще проконтролировать пользователя, чтобы честно играл, а не держал постоянно нажатой клавишу. Но обычно более актуальной всегда является задача контроля, чтобы целый рабочий день работник не занимался только выработкой случайных чисел.

Можно и самому, не связываясь с классическими играми, написать простую программу, в которой попросить пользователя осуществить некоторые действия, а по его реакции построить случайную последовательность. Приведу пример одной из таких программ, которую я написал для выработки случайной последовательности длиной 100 десятичных цифр. В ней пользователю предлагается поймать мышью маленький "бегающий ключик", по моментам "поимки" строятся случайные десятичные числа, окно программы показано на рис. 8.1.

Для реализации случайных движений маленького ключика использовался обычный генератор случайных чисел, реализованный в C++:

```
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
  Panel1->Color = clWhite;
  Image1->Left = rand()%100;
  Image1->Top = rand()%100;
}
```

При щелчке мышью по прыгающему ключику меняется цвет панели и к массиву, предназначенному для секретного ключа, добавляется очередной случайный знак:

```
void __fastcall TForm1::Image1MouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
  struct time t;
```

```
  gettime(&t);
  key[n_key]=t.ti_hund%10 + '0'; // секретный ключ

  Panel1->Color=clAqua;
  n_key++; // Указатель в массиве с секретным ключом
  Label1->Caption = IntToStr(n_key);
  ...
}
```

Результат работы программы показан на рис. 8.2.

Этот рисунок наверняка вызовет у читателя несколько вопросов.

- Почему секретный ключ представлен десятичными цифрами, а не байтами?
- Что такое пароль?
- Что такое метка?

Давайте разбираться.

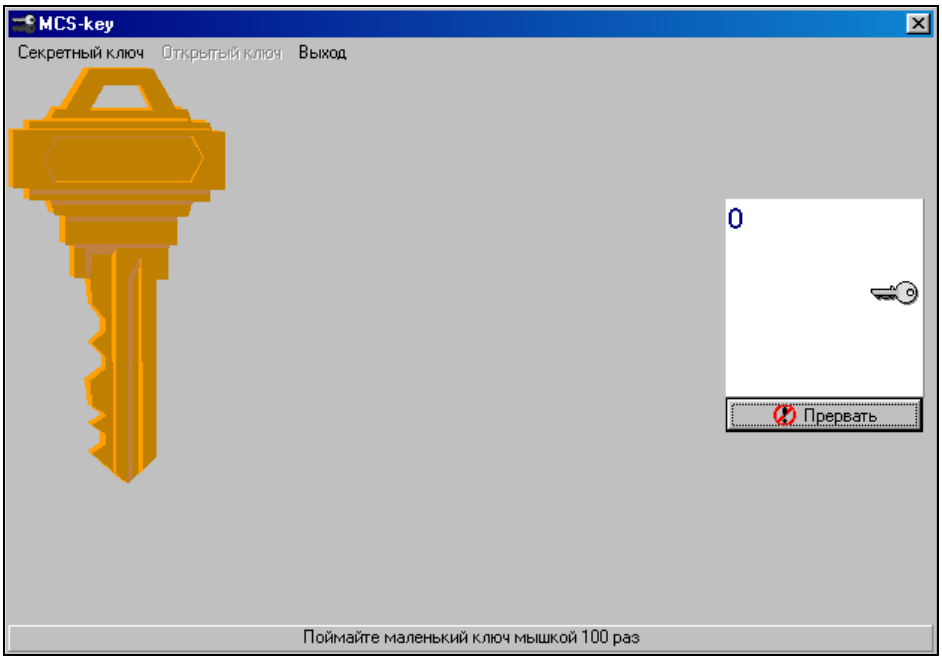

Рис. 8.1. Программа выработки случайной последовательности

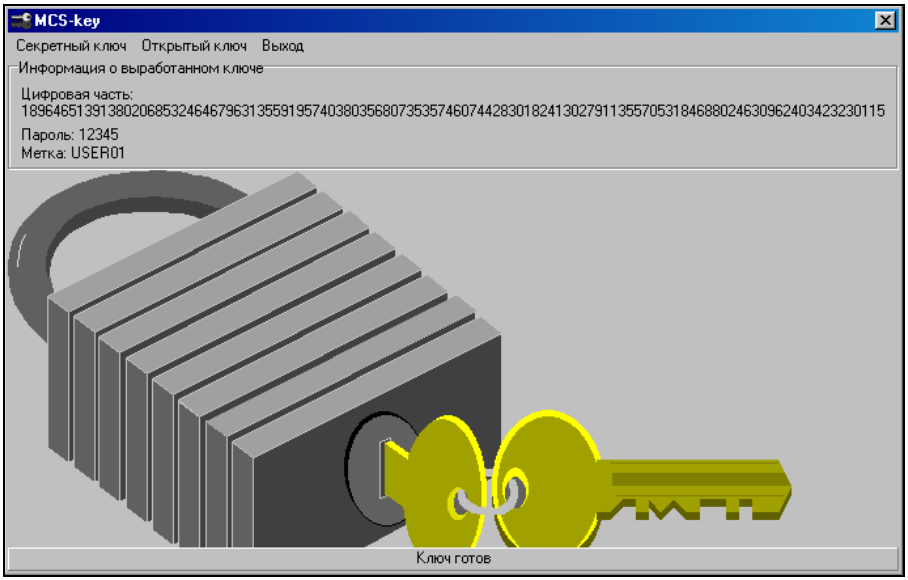

Рис. 8.2. Результат работы программы выработки случайной последовательности

Почему секретный ключ представлен десятичными цифрами?

При выработке ключа мы брали в качестве одного знака одну сотую часть секунды, приведенную по модулю 10. Полагая, что сотая доля секунды выбирается случайно и равновероятно из множества чисел от 0 до 99, получаем, что каждый знак нашего ключа выбирается случайно и равновероятно из множества чисел от 0 до 9. Представление полученного таким образом случайного знака в виде байта или иного числа с *двоичной арифметикой* (т. е. из 0 и 1) могло бы привести к появлению *неравновероятности*, что, конечно же, нежелательно. Кроме того, в реальной жизни возможны всякие ситуации. Например, может потребоваться распечатать секретный ключ и отдать распечатку корреспонденту, с которым затем будет налаживаться закрытая связь. Корреспондент же по распечатке сможет ввести этот ключ вручную с клавиатуры. О различных вариантах организации закрытой связи мы уже говорили в *главе 4*, это один из возможных вариантов. Представление ключа в виде десятичных цифр, разбитых на 5-значные группы, является классическим криптографическим приемом. Вспомните детективы, фильмы про Штирлица и прочих разведчиков — все писали шифровки в виде 5-значных цифровых групп. Так привычно и удобнее. А для записи на специализированный носитель (дискета, Touch Memory, Smart Card) этот ключ можно перевести в другой формат. Это не составит особого труда. Да и число всевозможных ключей легко подсчитать — 10^{100}.

Что такое пароль?

Недавно, еще в начале этой главы, автор ругал пароль последними словами, а тут вдруг взял и сам стал зачем-то его использовать. Что случилось?

Ничего не случилось. Гарантированную защиту по-прежнему обеспечивают 100 случайных цифр. Но их запомнить невозможно, их надо записать на носитель. На дискету, например. Дискету носить всегда с собой, не оставлять без присмотра. А такое в реальной жизни бывает очень редко. Очень часто дискеты оставляют на рабочем столе без присмотра. У потенциального злоумышленника появляется возможность скопировать файл с ключом. Так давайте же усложним ему жизнь, не отдадим наши секреты без боя даже в этом случае. 100 цифр он получит, но этого оказывается недостаточно для вычисления *реальных* секретных ключей, с помощью которых осуществляются операции шифрования и подписи. Сделаем процедуры, перемешивающие эти 100 цифр с паролем, причем так, что без тотального опробования всевозможных паролей добраться до реальных секретных ключей было бы невозможно. А поможет нам в этом наша замечательная процедура `CryptBlock`, которую мы самой первой положили в криптографическое ядро. Вот один из примеров того, как это можно осуществить.

Пример процедуры перемешивания 100-значного ключа с паролем

```
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
  FILE *KeyFile;
  unsigned char Key[100];
  bool PiCnt[256];
  unsigned char empt; // Вспомогательная переменная
  unsigned int i,j;
  unsigned char Pi[256]; // Подстановка Pi
  unsigned char PiGam[256]; // Вспомогательный массив
  unsigned char x[8];
  unsigned char Pass[128]; // Массив под пароль

  // Открываем файл со 100-значным ключом и считываем его
  KeyFile = fopen("KeyFile","rb");
  fread(Key,100,1,KeyFile);
  fclose(KeyFile);

  // Задаем начальную точку генератора случайных чисел
  srand(15268);
```

```
memset(PiCnt,0,256); // Обнуление контрольного массива

// Сначала строим фиксированную подстановку Pi
for(i=0;i<256;i++)
  {
      empt=(unsigned char)rand();
      while(PiCnt[empt]==true)empt++;
      Pi[i]=empt;
      PiCnt[empt]=true;
  }

// Теперь строим подстановку, зависящую от 100 знаков ключа
  for(i=0;i<8;i++)x[i]=(unsigned char)rand();
    for(i=0;i<32;i++)
    {
      CryptBlock(x,Pi,Key,100);
      memcpy(PiGam+8*i,x,8);
    }
    memset(PiCnt,0,256);
    for(i=0;i<256;i++)
    {
      while(PiCnt[PiGam[i]]==true)PiGam[i]++;
      Pi[i]= PiGam[i];
      PiCnt[PiGam[i]]=true;
    }

// Вводим пароль
  memset(Pass,0,128);
  InputPassword(Pass);

// Строим подстановку, зависящую как от 100 знаков ключа,
// так и от пароля
  for(i=0;i<32;i++)
  {
    // Пароль может быть коротким, поэтому повторяем процедуру
    // несколько раз
    for(j=0;j<16;j++)
      CryptBlock(x,Pi,Pass,strlen(Pass));
    memcpy(PiGam+8*i,x,8);
```

```
    }
    memset(PiCnt,0,256);
    for(i=0;i<256;i++)
    {
      while(PiCnt[PiGam[i]]==true)PiGam[i]++;
      Pi[i]= PiGam[i];
      PiCnt[PiGam[i]]=true;
    }
  // Получившаяся подстановка зависит уже от 100 знаков ключа и от пароля
  // С ее помощью строим секретные ключи для различных целей
...
    return 0;
}
```

Некоторые комментарии по поводу приведенной программы. В ней опять же ключевую роль играют подстановки из S_{256}. На первом этапе мы задаем фиксированную подстановку с помощью генератора случайных чисел `RAND`. С помощью этой подстановки, 100 цифр ключа и процедуры `CryptBlock` строим массив из 256 байт, который затем переделываем в подстановку. Получившаяся подстановка зависит уже от 100 знаков ключа. С ее помощью, используя опять же `CryptBlock` и пароль, строим аналогичным образом подстановку, зависящую уже от 100 цифр ключа и от пароля.

Даже если потенциальный злоумышленник будет обладать этим исходным текстом, а также сумеет получить файл с ключом, ему придется проводить полное опробование всех возможных паролей. Других способов получения итоговой подстановки я, например, не вижу.

Процедура `CryptBlock` аналогичным образом может применяться также для построения различных *проверочных комбинаций*, необходимых для осуществления контроля за правильностью ввода ключа. При считывании 100 знаков ключа из файла необходимо проконтролировать правильность их считывания. Для этого в файл необходимо дописать некоторую проверочную комбинацию, получающуюся в результате зашифровывания некоторого фиксированного блока на этом ключе с помощью процедуры `CryptBlock`. При считывании результат контрольного зашифровывания сравнивается с получившимся результатом, после чего принимается решение о правильности считывания. Аналогично проверяется правильность ввода пароля.

Контроль за процессом ввода секретного ключа

```
KeyFile = fopen("KeyFile","rb");
fread(Key,100,1,KeyFile);
// Контрольный блок для проверки правильности считывания из файла
```

```
fread(ControlFile,8,1,KeyFile);
// Контрольный блок для проверки правильности ввода пароля
fread(ControlPass,8,1,KeyFile);
fclose(KeyFile);

// Задаем начальную точку генератора случайных чисел
...
// Сначала строим фиксированную подстановку Pi
...

// Теперь строим подстановку, зависящую от 100 знаков ключа
...

// Вычисление контрольного блока
CryptBlock(x,Pi,Key,100);

// Сравнение контрольного блока с эталонным
if(memcmp(x,ControlFile,8)!=0)
{
  MessageBox(0,"Ошибка при считывании ключа из файла!","",MB_OK);
  return(-1);
}

// Вводим пароль
...

// Строим подстановку, зависящую как от 100 знаков ключа,
// так и от пароля
...

// Вычисление контрольного блока
for(j=0;j<16;j++)
  CryptBlock(x,Pi,Pass,strlen(Pass));
  // Сравнение контрольного блока с эталонным
  if(memcmp(x,ControlPass,8)!=0)
  {
    MessageBox(0,"Ошибка при вводе пароля!","",MB_OK);
    return(-1);
  }
...
```

Что такое метка?

В *главе 6*, посвященной электронной подписи, мы уже упоминали о метке ключа как о некотором универсальном идентификаторе, по которому система проверки электронной подписи будет отыскивать в базе данных всевозможных открытых ключей ключ данного пользователя. Представленный ранее пример секретного ключа можно рассматривать как пример *базового* секретного ключа пользователя, из которого затем, с использованием похожих механизмов и функции `CryptBlock`, строятся различные секретные ключи для конкретных криптографических задач — отдельный ключ для шифрования без использования системы с открытым распределением ключей, отдельный ключ для шифрования с использованием системы с открытым распределением ключей, отдельный ключ для подписи. Но метка, т. е. идентификатор, определяющий пользователя, у всех этих ключей будет одна и та же. Так удобнее потом будет строить различные системы автоматизированного электронного документооборота. Значение метки при проверке подписи мы уже знаем из *главы 6*, подобную же роль играет метка при шифровании с использованием системы с открытым распределением ключей. Только в этом случае к файлу добавляется не его подпись, а метки ключа отправителя и получателя и открытый ключ отправителя. Для расшифрования такого файла вообще говоря даже не требуется база данных с открытыми ключами. Но, тем не менее, во избежание различных недоразумений, ее, т. е. базу данных с открытыми ключами, использующимися для шифрования, все-таки лучше иметь. Для дополнительной гарантии открытый ключ, присоединенный к зашифрованному файлу, сравнивается с открытым ключом, находящимся в базе данных получателя. Если они не совпадают, то это повод для размышлений о причинах такого несовпадения. Оно может быть вызвано организационными неурядицами (отправитель сменил свой ключ и забыл отослать новый открытый ключ своим адресатам) или же попытками потенциального злоумышленника послать свою зашифрованную информацию от чужого имени.

Таким образом, мы научились строить секретные ключи для каждого пользователя. Давайте теперь обсудим те проблемы, с которыми столкнется управление безопасности крупной организации при централизованном изготовлении секретных ключей для большого числа пользователей.

Очевидно, что если количество пользователей составляет несколько десятков или даже сотен, то щелкать мышью 100 раз для каждого из них для сотрудников этого управления будет утомительно. Кроме того, необходимо будет иметь возможность повторной выработки секретного ключа в случае, если дискета с ним придет в негодность. Таким образом, в этом случае удобнее использовать *программный* способ выработки большого числа секретных ключей. В чем его особенности?

- Существует некоторый Главный секретный ключ, который вырабатывается с помощью случайной последовательности и из которого затем программным путем строятся все остальные секретные ключи пользователей.
- В выработке секретного ключа пользователя принимают участие Главный секретный ключ и метка (идентификатор) пользователя.
- Компрометация секретного ключа одного пользователя не должна приводить к возможности вскрытия секретных ключей каких-либо других пользователей.
- Все секретные ключи должны быть одного и того же формата, чтобы допускалась возможность их использования в единой системе электронного документооборота.

Для удобства *выработки* секретных ключей здесь мы сознательно отказываемся от их *случайности* и переходим к заведомо *псевдослучайным* секретным ключам. Такое возможно только в том случае, если есть твердая уверенность в *однонаправленности* преобразований Главного секретного ключа в индивидуальные секретные ключи пользователей. В том, что по какому-то скомпрометированному индивидуальному секретному ключу (а такое событие в большой сети бывает часто) потенциальному злоумышленнику не удастся вычислить Главный секретный ключ.

В *главе 6*, посвященной электронной подписи, мы уже упоминали о том, как можно в этом случае приспособить хэш-функцию. Она будет выполнять роль своеобразного "расширителя", т. е. формировать по сравнительно короткой метке ключа индивидуальную последовательность практически любой требуемой длины. Теперь эту последовательность требуется просто зашифровать с использованием *блочного* шифра и Главного секретного ключа. Блочный шифр в этом случае удобнее, поскольку после хэш-функции мы уже шифруем практически случайные и равновероятные блоки, а передача и искажение шифртекста здесь неактуальны. Потенциальный злоумышленник, получив скомпрометированный секретный ключ, фактически получает несколько переходов в подстановке большой степени, реализуемой блочным шифром, — ведь идентификатор пользователя, алгоритм его хэширования и, следовательно, хэш-функция предполагаются известными. Предположим, что длина секретного ключа — 1024 бит: это 128 байт или 16 блоков по 8 байт. Это 16 переходов в подстановке большой степени, реализуемой блочным шифром. Всего в подстановке $(2^8)^8 = 2^{64} = 10^{19}$ переходов. Вероятность того, что случайно выбранный переход попадет в число 16 известных составляет, таким образом, $2^{-60} = 10^{-18}$. В прошлой главе мы договорились все методы, надежность которых меньше, чем 10^{-10}, считать неопасными. Таким образом, классический метод анализа блочного шифра — метод "каталога" — в данном случае следует считать неопасным.

Потенциальному злоумышленнику остается одно: пытаться *вычислить* Главный секретный ключ по ставшими ему известными переходам в подстановке большой степени, реализуемой блочным шифром. Тут все уже зависит от того, как построить блочный шифр. Конечно же, с помощью процедуры `CryptBlock`! Но соблюдая при этом следующие правила "криптографической гигиены":

1. Используемая в `CryptBlock` подстановка π должна зависеть от Главного секретного ключа.
2. Количество тактов в цикле (параметр T) не должно быть маленьким, здесь особая скорость не нужна, лучше не экономить.
3. Знаки Главного секретного ключа, помимо формирования подстановки, подаются на вход в качестве параметров k_i.

Приведенный ранее пример программы перемешивания случайного секретного ключа с паролем может также служить примером соблюдения этих правил. Хотя это далеко не единственный способ. И здесь опять же поле для фантазии разработчика системы защиты. Выдумывайте свои способы формирования подстановки, свои процедуры `CryptBlock` с различными параметрами, тем самым вы зададите дополнительные проблемы потенциальному злоумышленнику. Но процедура выработки индивидуальных секретных ключей из Главного ключа все-таки требует к себе более пристального внимания, чем обычные процедуры шифрования и подписи.

И, наконец, еще об одном условии — едином формате. Выработанный секретный ключ пользователя также хотелось бы представить в виде цифровой последовательности из 100 цифр и пароля. Начнем с пароля. Дело вкуса и возможностей пользователей. При централизованном обеспечении секретными ключами всех пользователей можно заранее попросить их придумать себе пароли, а можно сделать генератор случайных простых паролей и заставить пользователей их запомнить. А можно вместо пароля использовать отпечаток пальца пользователя, но об этом пойдет подробный разговор в *части VI*. Давайте лучше пока поговорим о 100 цифрах.

Наша процедура `CryptBlock` на выходе выдает байты, т. е. элементы из $Z/256$. Псевдослучайные и практически в неограниченном количестве. Нужна еще процедура получения из них 100 псевдослучайных цифр. Если мы просто будем приводить каждый байт по модулю 100, то получившаяся таким образом последовательность уже не будет равновероятной. Элементы от 0 до 55 будут иметь в ней преобладание, поскольку при приведении элементов из $Z/256$ по модулю 100 число значений, дающих результат от 0 до 55, будет на 1 больше, чем тех, которые дают результат

от 56 до 99. Появляются "элементы с привилегиями". Их надо отсеивать, т. е.:

- если значение байта от 0 до 199, то оно приводится по модулю 100 и включается в секретный ключ;
- если значение байта от 200 до 255, то оно пропускается.

Получается некоторый *вероятностный процесс*, но поскольку мы можем вырабатывать практически неограниченное количество байт, он будет *сходящимся*, т. е. с вероятностью, близкой к 1, давать нам требуемый результат из 100 случайных цифр.

Ну и напоследок: никогда не рассылайте изготовленные секретные ключи пользователям по обычной электронной почте, не шифруйте их на старых ключах! Лучше всего передавать их лично из рук в руки, или, в крайнем случае, использовать для рассылки шифрование с помощью одноразовой случайной гаммы. Но это также требует организационных мер и специальных программных средств для работы с одноразовой гаммой.

Глава 9

База данных открытых ключей

Секретные ключи созданы. С их помощью теперь будем создавать открытые ключи, рассылать их адресатам, а также формировать базу данных открытых ключей. Кое-что на эту тему мы уже затрагивали в *главе 6*, посвященной электронной подписи, сейчас же настало время углубиться в эти вопросы посильнее, тем более, что все проблемы, связанные с открытыми ключами, их изготовлением, хранением и сменой, в большинстве существующей литературы по криптографии упоминаются вскользь как не имеющие особого значения. Для практической же эксплуатации криптографических систем бережное и чрезвычайно деликатное обращение с открытыми ключами имеет первостепенное значение, к манипуляциям с открытыми ключами потенциальный злоумышленник может приступить после того, как убедится в бесплодности прямых попыток взлома системы. Давайте заранее обсудим все возможные особенности изготовления, рассылки, хранения и смены открытых ключей, чтобы и здесь быть гарантированными от возможных неприятностей.

Начнем с изготовления. Для этого будем использовать метод Диффи-Хеллмана и процедуру `ExpA`, которую мы положили в наше криптографическое ядро. Конкретные параметры — основание a, длина секретного и открытого ключа, используемый принцип возведения (операции с простыми числами, многочлен в поле $GF(2^n)$ или эллиптические кривые) — все это определяется самим разработчиком криптографической системы в зависимости от потребностей заказчика и условий ее эксплуатации. Мы считаем, что процедура `ExpA` уже существует, на ее вход можно подавать некоторый секретный ключ, а на выходе получать готовый открытый ключ в виде последовательности байт. Как распорядиться полученным открытым ключом, куда его положить и как с его помощью затем организовывать эксплуатацию криптографической системы? Давайте разбираться.

1. В предыдущих главах мы уже упоминали о том, что секретный ключ для шифрования с использованием системы с открытым распределением

ключей и секретный ключ для подписи должны быть различными, хотя могут строиться из одного и того же базового секретного ключа. Эта же точка зрения выражена и в книге Б. Шнайера: "Любая хорошая реализация криптографии с открытыми ключами должна использовать разные ключи для шифрования и для цифровых подписей". Поэтому каждый открытый ключ должен состоять из двух частей: открытый ключ для шифрования и открытый ключ для подписи. Пару этих открытых ключей мы в дальнейшем будем называть просто открытым ключом пользователя.

2. В файле с открытым ключом пользователя обязательно должна быть метка ключа или любой другой идентификатор пользователя. Иначе при проверке подписи невозможно будет найти нужный ключ среди множества всевозможных открытых ключей различных пользователей.

3. Должна быть предусмотрена возможность *визуального* контроля файла с открытым ключом, т. е. все входящие в файл символы должны быть отображаемыми на экране с помощью типовых текстовых редакторов. Помимо чисто психологических аспектов, визуальный контроль также гарантирует возможность *распечатки* файла и заверения распечатки с помощью обычных подписей и печати. Это требование, безусловно, приводит к увеличению длины файла, ведь в открытом ключе содержатся случайные ASCII-символы и многие из них не отображаются на экране с помощью типовых текстовых редакторов: 13 — возврат каретки, 10 — перевод строки, 255 — пустой символ и т. п. Для выполнения этого требования открытый ключ можно представлять в HEX-формате, когда каждому ASCII-символу ставится в соответствие пара символов от 00 до FF.

4. Ключи необходимо периодически менять. При смене секретных ключей автоматически меняются соответствующие им открытые ключи. Следовательно, в файле с открытым ключом необходимо указать дату ввода соответствующего секретного ключа в эксплуатацию.

Смена ключей настолько жизненно важная проблема, что здесь необходимо остановиться на ней как можно подробнее.

Итак, если в крупной организации функционирует система защищенного электронного документооборота, использующая криптографические процедуры, связанные с секретными и открытыми ключами, то для локализации последствий возможных компрометаций ключей их необходимо периодически менять. Компрометация ключей может произойти как в силу каких-то чрезвычайных обстоятельств (потеря контроля над дискетой с ключом), так и просто в силу того, что работник, имевший доступ к секретному ключу, решил уволиться. Тот ключ, к которому он имел доступ, после его увольнения считается скомпрометированным.

Смена ключей может быть как плановая, так и внеплановая. Плановая смена осуществляется регулярно, вне зависимости от каких-либо обстоятельств. Это просто полная смена всех ключей всем пользователям. Периодичность

такой смены определяется исходя из количества корреспондентов, принимающих участие в электронном документообороте, ценности и объемов обрабатываемой и пересылаемой информации, возможности прервать на некоторое время отлаженную процедуру документооборота для замены ключей, наличия специальных программных средств, предназначенных для облегчения процедуры смены ключей. При плановой смене ключей назначается день и час "X", начиная с которого старые ключи прекращают свое действие, а вся дальнейшая информация обрабатывается только на новых ключах. И здесь тоже возможны накладки. Если сеть корреспондентов раскинулась по большой территории, то, к примеру, когда в Москве 15 часов дня, в Петропавловске-Камчатском полночь. Следовательно, час "X" должен быть не просто "1 января 2000 года 0 часов 00 минут 00 секунд", а обязательно с добавлением, например, "по Московскому времени". Иначе потом корреспонденты не поймут друг друга, система будет выдавать ошибки при расшифровании или проверке подписи.

Следовательно, в базе данных, содержащей открытые ключи, в момент смены должно быть либо два открытых ключа: старый и новый, либо это просто должны быть две разные базы: одна со старыми ключами, другая — с новыми.

И здесь сразу же возникает вопрос: а почему необходимо жестко задавать час "X", до которого действуют старые ключи, а после — новые? Может быть, проще было бы сделать некоторый интервал, в течение которого действительны оба открытых ключа, как старый, так и новый?

Все дело в том, что помимо плановой смены всех ключей возможна внеплановая замена скомпрометированных ключей. И в этом случае интервал категорически недопустим! Ключ — в руках у злоумышленника, чем быстрее его заменим, тем лучше. Скомпрометированный ключ надо немедленно, без всяких интервалов запретить к употреблению. Интервал в этом случае — это дополнительный шанс получить фальшивый платежный документ, злоумышленник на время действия интервала оказывается в тех же условиях, что и законный пользователь. Делать же программные средства под две различных системы смены ключей — плановой и внеплановой — и дорого, и больше вероятность программных сбоев или ошибок персонала.

Итак, в файле с открытым ключом, использующемся при электронном документообороте в крупной организации, обязательно должен присутствовать час "X" — время, с которого этот ключ вступает в действие. Причем либо час "X" привязан к местному времени, либо таймеры на компьютерах всех корреспондентов должны быть синхронизированы под одно и то же поясное время (например, все настроены на Московское время). Иначе в переходный период возможны сбои. Программа, реализующая проверку подписи, должна сравнить текущее время с часом "X". Если час "X" не наступил, то необходимо в базе данных открытых ключей искать другой ключ.

А теперь рассмотрим проблему сертификации открытого ключа. Правильнее было бы сказать не просто открытого ключа, а всего файла с открытым ключом, включающего помимо ключа также его идентификатор и время ввода в эксплуатацию. Об этом мы уже упоминали в *главе 6*, посвященной электронной подписи, теперь настало время разобраться во всех этих проблемах досконально.

В некоторых криптографических приложениях в качестве сертификата открытого ключа предлагается система поручительств. Вот что написано о ней у Шнайера: "Поручители — это пользователи системы, которые подписывают открытые ключи своих друзей. Например, когда Боб создает свой открытый ключ, он передает копии ключа своим друзьям — Кэрол и Дэйву. Они знают Боба, поэтому каждый из них подписывает ключ Боба и выдает Бобу копию своей подписи. Теперь, когда Боб предъявляет свой ключ чужому человеку, Алисе, он предъявляет его вместе с подписями этих двух поручителей. Если Алиса также знает Кэрол и доверяет ей, у нее появляется причина поверить в правильность ключа Боба. Если Алиса знает Кэрол и Дэйва и хоть немного доверяет им, у нее также появляется причина поверить в правильность ключа Боба. Если она не знает ни Кэрол, ни Дэйва, у нее нет причин доверять ключу Боба".

Боб, Кэрол, Дэйв, Алиса, милый пикничок на лужайке. Все друг другу верят и закрепляют это своими электронными подписями. Для обмена любвеобильными посланиями по электронной почте, пожалуй, неплохая система. А если эти ребята — сотрудники банка, имеющие дело с денежными документами, то они по-прежнему с такой же детской наивностью будут выдавать деньги под электронные подписи, заверенные друг другом? А если банк имеет филиальную сеть от Москвы до Владивостока и вся система документооборота будет построена на подобной системе поручительств? А если "подруга Кэрол", раздав кучу поручительств, решит уволиться из банка? Сколько усилий и средств потребуется на то, чтобы в такой системе вывести из эксплуатации скомпрометированный ключ Кэрол? А сколько таких Кэрол во всех филиалах банка?

Пробовали ли вы принести в банк карточку с образцом своей подписи, заверенной не у нотариуса, а другими клиентами банка? Нет, в наших суровых российских условиях для сертификации открытых ключей лучше все-таки иметь специализированные сертификационные центры, некоторые аналоги нотариальных контор. Эти сертификационные центры объективно не должны отражать интересы одного или нескольких участников системы электронного документооборота, а быть заинтересованными только в одном: в исправном ее функционировании. Возможно, что и в развитии, реализации новых идей, оснащении участников всеми необходимыми средствами для осуществления документооборота, обучении персонала. В периодическом контрольном криптографическом анализе используемых средств электронной подписи. В вовлечении в сферу документооборота новых участников.

В минимизации бюрократических процедур, сведении процедуры регистрации пользователя к минимуму, необходимому для исправного и безопасного функционирования всей системы. Центр должен иметь возможность быстро и оперативно реагировать на случаи компрометации секретного ключа: новый открытый ключ должен быть в кратчайшие сроки разослан всем участникам системы документооборота. Центр должен заранее готовиться к плановой смене ключей одним или несколькими абонентами. Центр должен вести архив всех действующих и выведенных из действия открытых ключей с указанием срока их действия. В конечном итоге, в бесперебойном функционировании сертификационного центра должны быть заинтересованы все участники системы документооборота, он должен быть *добровольным*, не навязываемым сверху могущественным монополистом.

Итак, каждый открытый ключ, включенный в базу данных открытых ключей, должен пройти сертификацию в сертификационном центре. При сертификации он подписывается уполномоченным представителем центра, которого ранее в *главе 6* мы назвали менеджером системы безопасности.

Давайте теперь рассмотрим один возможный пример базы данных с открытыми ключами.

```
*****181200A01\001 7fc0bc8eb06ddd05c9032414c1ff48b54a8bf776ad4cd2a8b...
01.01.2001 0:00:00
*****201200KEYMAN  12353534331516480 86f6cde6a89126d3583432b6dc881bf7...

*****211200A01M01  4645d577fe5d1e747f8590a7f70570530ec6b3276df488ca3...
01.01.2001 0:00:00
*****231200KEYMAN  20253452f8d570f975b8e32037999c9f1451b2fed7adec2be...

*****151200A01P01  7cb85559746532314cbb2da9c6936049a89b691c3f28500e2...
01.01.2001 0:00:00
*****181200KEYMAN  091418967db1c6c6d562e64e325c847ad0adf7e36e6a15ffe...

*****231200A02\001 f684579d8a596cc91b18382f426af233458a74b8eff82b9a3...
01.01.2001 0:00:00
*****251200KEYMAN  1120266190d4ee174c9d24b79734c071e2fd901223d78ec9a...
```

Здесь 4 открытых ключа, заверенных менеджером СБ из сертификационного центра. Идентификатор менеджера СБ — `KEYMAN`. Идентификаторы пользователей:

```
A01\001
A01M01
A01P01
A02\001
```

Каждый открытый ключ начинается с символов *****. Далее следует дата изготовления открытого ключа, идентификатор пользователя и сам открытый ключ. В следующей строке, сразу после открытого ключа, указан час "X" — время ввода его в эксплуатацию. После этого стоит цифровая подпись менеджера СБ, также начинающаяся с символов ***** и защищающая как сам ключ, так и все его атрибуты. Обратите внимание, что подпись менеджера СБ стоит не после всего файла с базой, а после каждого открытого ключа в отдельности. Вся база при этом не подписывается.

В подписи менеджера СБ мы можем найти дату и время сертификации соответствующих ключей:

```
A01\001        20.12.00   12:35:35.34
A01M01         23.12.00   20:25:34.52
A01P01         18.12.00   09:14:18.96
A02\001        25.12.00   11:20:26.61
```

Час "X", т. е. время ввода всех этих ключей в эксплуатацию одно и то же — 01.01.2001 00:00:00. Формат записи даты и времени ввода в эксплуатацию должен быть распознаваем операционной системой, в которой будет использоваться эта база данных с открытыми ключами.

В программе, обращающейся к этой базе данных, обязательно должна быть процедура контроля за текущим временем. Это может быть визуальная процедура (при запуске программы оператору предлагается подтвердить текущие дату и время), процедура контроля по времени, установленному на некотором эталонном компьютере в сети, процедура контроля времени по Internet (не очень хороший вариант!) и т. п. При необходимости выбора из базы данных открытого ключа конкретного пользователя программа первым делом должна запрашивать текущее время. Ключ отыскивается по идентификатору пользователя, затем текущее время сравнивается с часом "X", прописанном в ключе. Если час "X" еще не наступил, то отыскивается другой ключ с таким же идентификатором. Если в базе данных отыскиваются два ключа одного и того же пользователя, для которых час "X" уже наступил, то используется ключ с более поздним часом "X". Вот пример соответствующей программы.

Пример процедуры выбора нужного открытого ключа из базы данных открытых ключей

```
// UserDir — рабочий каталог пользователя
// Abonent — идентификатор требуемого открытого ключа
// PublKey — имя файла, в который будет записан выбранный открытый ключ

int CreatPublFile(char *UserDir,char *Abonent,char *PublKey)
```

```
{
  FILE *fl_pbl;  // Выбранный открытый ключ будет записан в этот файл
  FILE *fl_base; // файл с базой данных открытых ключей

  unsigned char buff[1024]; // Вспомогательный буфер
  long BaseLength;

  // Количество найденных открытых ключей с данной меткой
  unsigned char NKeys = 0;

  // DateTime3 — самая последняя дата ввода ключа в эксплуатацию
  // CurDateTime — текущая дата.

  // Имя файла с требуемым открытым ключом
  sprintf(PublKey,"%s\\%s.PBL",UserDir,Abonent);

  // Имя файла с базой данных открытых ключей
  sprintf(buff,"%s\\PUBLKEYS.BAS",UserDir);

  if((fl_base=fopen(buff,"rb"))==NULL)
    return(-1);
  BaseLength=filelength(fileno(fl_base));

  DateTime = new TDateTime();
  CurDateTime = new TDateTime(DateTime->CurrentDateTime());

  // Первый проход — определяем даты ввода ключей в эксплуатацию
  while(ftell(fl_base)+PUBLKEYLENGTH<=BaseLength)
  {

    // Считываем весь открытый ключ
    fread(buff,PUBLKEYLENGTH,1,fl_base);

    // Сравниваем метку ключа с требуемым идентификатором
    if(memicmp(Abonent,buff+ABNIDOFF,strlen(Abonent))!=0)
      continue;

    // Если ключ с такой меткой до этого не встречался
    if(NKeys==0)
```

```
  {
    // Запоминаем час X первого найденного ключа
    DateTime1 = new TDateTime(buff+DATEXOFF);
    DateTime3 = new TDateTime(DateTime1->Val);
    NKeys=1;
  }

  // Если ключ ранее уже встречался
  else
  {
    if(NKeys==1)
    {
      delete DateTime3;
      // Запоминаем час X второго найденного ключа
      DateTime2 = new TDateTime(buff+DATEXOFF);
      // DateTime3 — наиболее поздний час X
      if(DateTime1->Val>=DateTime2->Val)
        DateTime3 = new TDateTime(DateTime1->Val);
      else
        DateTime3 = new TDateTime(DateTime2->Val);
      NKeys=2;
    }
    else
     return(-1); // Наличие третьего ключа — ошибка
  }

} // Конец первого прохода

// Если ни одного ключа за первый проход не нашли, то аварийный выход
if(NKeys==0)return(-1);

  // Второй проход — включаем нужный ключ в файл
  fseek(fl_base,0,SEEK_SET);
  while(ftell(fl_base)+PUBLKEYLENGTH <= BaseLength)
  {
    fread(buff,PUBLKEYLENGTH,1,fl_base);
    if(memicmp(Abonent,buff+ABNIDOFF,strlen(Abonent))!=0)
      continue;
    delete DateTime1;
```

```
      DateTime1 = new TDateTime(buff+DATEXOFF);

      // Если нашли самый последний ключ и
      // дата ввода его в эксплуатацию уже наступила
      if(DateTime1->Val==DateTime3->Val &&\
      DateTime1->Val<=CurDateTime->Val)
        goto include;

      // Если нашли старый ключ, но дата ввода
      // в эксплуатацию нового еще не наступила
      if(DateTime1->Val!=DateTime3->Val &&\
      DateTime3->Val>CurDateTime->Val)
        goto include;
    } // Конец второго прохода
  return(-1); // Если второй проход не дал результата

include:

  if((fl_pbl=fopen(PublKey,"w+b"))==NULL)
    return(-1);
  fwrite(buff,PUBLKEYLENGTH,1,fl_pbl);
  fclose(fl_pbl);
  return(0);
}
```

После формирования файла с требуемым открытым ключом проверяется присутствующая в нем подпись менеджера СБ.

Все эти усложнения программной реализации предназначены только для одного — облегчить чрезвычайно болезненную процедуру смены ключей в большой организации, дать возможность подготовиться к ней заранее, занести новые открытые ключи в базу данных за несколько дней до наступления часа "X". После наступления часа "X" все старые ключи перестают работать. Их нужно вообще удалить из базы данных. Необходимость наличия в базе двух ключей — старого и нового — одного пользователя возникает только *до* часа "X", в период подготовки к нему. К моменту наступления часа "X" все документы, подписанные или зашифрованные на старых ключах, должны быть обработаны. После часа "X" они уже будут забракованы.

Менеджер СБ рассылает отдельные секретные ключи, заверенные своей электронной подписью, по обычной электронной почте. Естественно, что при обслуживании больших организаций менеджеру СБ целесообразно иметь специализированное программное обеспечение для автоматизации

рассылки открытых ключей большому числу адресатов. У него должна вестись база данных использованных и используемых открытых ключей, а также база данных осуществленных рассылок.

В свою очередь, каждый пользователь системы документооборота должен подтвердить менеджеру СБ факт получения и установки новых открытых ключей. Осуществление подобной операции каждым пользователем в отдельности подчас бывает неудобным и трудоемким. Несколько пользователей могут объединиться в группы и использовать для обеспечения себя необходимыми открытыми ключами *регионального* менеджера, в задачи которого будет входить получение необходимых открытых ключей на всех пользователей сразу от центрального менеджера СБ, установка их на рабочие места пользователям, отправка отчетов менеджеру СБ о проведенных установках. Но обо всех этих вопросах мы еще поговорим в *части V*, когда будем рассматривать конкретный пример организации системы электронного документооборота.

Глава 10

Процедура шифрования

Ключи готовы, можно приступать к шифрованию? Нет, еще рановато. Давайте сначала определим, что и как мы будем шифровать. Тут в литературе можно встретить множество экзотических терминов:

- симметричные и асимметричные алгоритмы шифрования;
- блочные и потоковые шифры;
- режим электронной шифровальной книги;
- режим сцепления блоков шифра;
- самосинхронизирующиеся потоковые шифры;
- режим обратной связи по шифру;
- синхронные потоковые шифры;
- режим выходной обратной связи;
- режим счетчика;
- другие режимы блочных шифров.

Давайте не будем пугаться этих терминов, постараемся спокойно разобраться, что к чему.

Основным объектом, с которым нам придется иметь дело при шифровании, будет файл. Иногда, правда, будем шифровать и другие объекты, например, сообщения в хранилищах MAPI, поля в базах данных Lotus Notes, но в основном все-таки будем иметь дело с файлами. Наша задача при зашифровывании состоит в том, чтобы вместо осмысленного открытого текста, представляющего последовательность байт (ASCII-символов), в файле появился бы шифртекст, опять же состоящий из ASCII-символов, но уже хаотических и не имеющих никакого смысла. Какой тип файла был до зашифровывания (текст, рисунок, исполняемая программа, база данных и т. п.), какую он содержал информацию — это для осуществления зашифровывания не должно играть никакой роли. Открытый текст — это просто любая последовательность

ASCII-символов, может быть даже представляющая кем-то ранее зашифрованный текст.

Зашифровывание файла будем осуществлять путем наложения на его знаки гаммы — последовательности байт, вырабатываемых некоторым генератором гаммы. Пользователь, обладающий ключом к шифру, всегда должен иметь возможность повторить начальное состояние генератора гаммы, выработать точно такую же гамму и расшифровать файл. При этом не имеет принципиального значения, каким путем у пользователя появляется ключ, с помощью которого он восстанавливает начальное состояние генератора гаммы. Этот ключ, как мы уже знаем из *главы 3*, обязательно должен быть *одноразовым*, различным для различных зашифрованных файлов.

"Одноразовость" вырабатываемой гаммы достигается за счет марканта. Но маркант при этом должен каким-то образом перемешиваться с постоянным секретным ключом, используемым при зашифровывании файла. А что представляет из себя постоянный секретный ключ? Тут возможны два варианта: либо это секретный ключ, вырабатываемый из базового секретного ключа пользователя и используемый только для шифрования, либо это общий секретный ключ, вырабатываемый по схеме с открытым распределением ключей. Но мы будем предполагать, что *формат* постоянного секретного ключа, используемого для шифрования файла, один и тот же. Применяем ли мы для выработки гаммы только свой базовый секретный ключ или же используем систему с открытым распределением ключей, формат (но не содержание!) секретного ключа для задания начального состояния генератора гаммы будет одинаковым. Поэтому процедура шифрования файла будет одной и той же, что для шифрования, только на своем индивидуальном ключе, что для шифрования с использованием системы с открытым распределением ключей. Все различие будет только в том, какие ключевые параметры (одинакового формата) подать на ее вход.

А какие ключевые параметры мы будем подавать на вход процедуры зашифровывания файла? Давайте вспоминать наше криптографическое ядро и предыдущие примеры. В первую очередь вспомним процедуру `CryptBlock`, а также впервые употребим и процедуру `Crypt`, отличающуюся от `CryptBlock` тем, что она накладывает на текст гамму, вырабатываемую по схеме блочного шифра. А какой самый криптографически главный элемент в этих процедурах? Конечно же, подстановка π из симметрической группы S_{256}. Вот ее-то мы и сделаем первым и криптографически самым главным ключевым параметром нашей процедуры зашифровывания файла. Как вырабатывать подстановки, зависящие от базового секретного ключа, — это мы видели в *главе 8*. Аналогичным способом можно вырабатывать и подстановки, зависящие от общего ключа, получаемого с помощью процедуры `ExpCom`, содержащейся в нашем криптографическом ядре *(см. главу 7)*.

Ну и пожалуй еще значения k_i, входящие в формулу преобразований нашего блочного шифра, также сделаем ключевыми параметрами. Выработать их

еще проще, чем подстановки, в них, в отличие от подстановок, допускается повторяемость символов. Это может быть любая случайная последовательность, зависящая от ключа.

Сама процедура шифрования осуществляется путем считывания в некоторый буфер открытого текста, наложения на него гаммы с помощью процедуры `Crypt` и записи зашифрованного текста из буфера на то же самое место в файле. Один байт открытого текста переходит в один байт шифрованного текста, длина шифртекста в буфере не увеличивается. А теперь настало время поговорить о марканте.

Для удобства работы с процедурой `CryptBlock`, выберем длину марканта, равной 8 байт. Всего различных маркантов будет $(2^8)^8 = 2^{64} = 10^{19}$, при случайном выборе двух маркантов вероятность их совпадения будет 10^{-19}, что является неопасным. Маркант строится по текущему времени в момент зашифровывания файла, например, таким путем:

Выработка марканта

```
getdate(&d);
x[0]=d.da_year/256;
x[1]=d.da_year%256;
x[2]=d.da_mon;
x[3]=d.da_day;
gettime(&t);
x[4]=t.ti_hour;
x[5]=t.ti_min;
x[6]=t.ti_sec;
x[7]=t.ti_hund;
```

Для зашифровывания будем использовать *одноразовые* подстановку π и значения k_i, которые строятся из аналогичных значений, поданных на вход процедуры, и марканта с использованием процедуры `CryptBlock`.

Построение одноразовых ключевых параметров

```
// Получаем три различных зашифрованных образа марканта
// Это необходимо для идентификации ключа перед расшифрованием
for(i=0;i<3;i++){
  CryptBlock(x,pi,k,40);
  memcpy(x_c[i],x,8);
}
// Вырабатываем одноразовые значения k
```

```
memcpy(k_u,k,40);

for(i=0;i<5;i++){
  CryptBlock(x,pi,k_u,40);
  // Подправляем кусок вектора k,
  // подправленный вектор сразу же используем
  // для подправления другого куска
  j=0;
  while(j<8){
    k_u[8*i+j]=k[8*i+j]+x[j];
    j++;
  } // Подправили кусок из 8 байт
} // Целиком выработали вектор k_u

// Вырабатываем одноразовую подстановку pi_u и подправляем вектор k_u
for(i=0;i<32;i++){
  CryptBlock(x,pi,key_u,40);
  for(j=0;j<8;j++){
    pi_u[8*i+j]=x[j];
    key_u[j]=key_u[j]+x[j];
  }
} // Создали массив для выработки подстановки

// Преобразуем массив в подстановку
memset(pi_c,0,256); // Контрольный массив

for(i=0;i<256;i++){
  while(pi_c[pi_u[i]]!=0)
    pi_u[i]++;
  pi_c[pi_u[i]]=true;
}
```

К этому листингу необходим комментарий. Маркант нужен для обеспечения "одноразовости" гаммы, т. е. для гарантии отсутствия перекрытий гаммы при шифровании различных файлов. Это его основная задача. Но здесь мы приспособили его еще и для других целей. Применили к марканту процедуру `CryptBlock` три раза, запоминая каждый раз результат работы в массиве `x_c`. Получили 24 знака, вообще говоря, случайных и равновероятных. Если теперь последние 8 байт *расшифровать* 2 раза с использованием процедуры `DecryptBlock`, а средние 8 байт расшифровать 1 раз, то все 3 части по 8 байт

должны быть одинаковыми. Но это будет только в том случае, когда при расшифровании в процедуре `DecryptBlock` мы будем использовать те же самые подстановку π и значения k_i, которые использовались при зашифровывании в `CryptBlock`. Таким образом, мы получаем *критерий истинности* постоянного секретного ключа, который предполагается использовать для расшифрования. Перед тем, как осуществлять расшифрование файла, по этому критерию можно заранее убедиться, тот ли мы имеем ключ для расшифрования.

Эти 24 байта, которые мы назовем *обобщенным маркантом*, дописываются в конец шифртекста после окончания процедуры наложения гаммы на весь открытый текст. Таким образом, процедура зашифровывания добавляет в файл 24 дополнительных знака обобщенного марканта.

Следовательно, перед началом зашифровывания необходимо убедиться в том, что диск, содержащий шифруемый файл, позволит записать в шифруемый файл дополнительные 24 байта. Если этого не сделать, то при переполненном диске мы можем просто потерять файл: зашифровав его, мы не сможем записать обобщенный маркант, а тогда не сможем и расшифровать.

Еще одна потенциальная опасность, которую следует учитывать при реализации процедуры шифрования файла, да и не только шифрования, а и вообще любой операции записи на диск при наличии в оперативной памяти секретных ключей — это проблема "хвоста" файла.

Давайте вспомним организацию файловой структуры на компьютере. Каждому файлу соответствует определенный набор *кластеров*, состоящих из нескольких *секторов* по 512 байт. Количество секторов в кластере зависит от объема носителя, типа операционной системы и некоторых других факторов. Для нас важно то, что все операции записи-считывания информации из оперативной памяти на диск и наоборот осуществляются кластерами. Если вы считываете из файла 1 байт, то физически в оперативную память может попасть весь кластер, содержащий этот байт. Если вы хотите создать файл, состоящий всего из 1 байта и записываете этот байт из оперативной памяти в файл, который будет находиться на диске, то физически вы записываете на диск целый кластер, размер которого может составлять несколько килобайт. Операционная система будет отображать только находящийся в файле 1 байт, но с помощью специальных средств прямого просмотра кластеров можно будет получить содержание всего соответствующего кластера.

Если длина записываемого файла не кратна длине кластера, то при записи содержания файла из оперативной памяти на диск туда, в конец последнего кластера, или в "хвост" файла, попадают посторонние элементы, не входящие в файл. И может случиться так, что эти элементы окажутся частью секретного ключа использовавшегося для зашифровывания файла! А могут оказаться частью открытого текста, который мы зашифровали. В любом случае, если оставить "хвост" без присмотра, то он, случайно или даже не случайно,

может сыграть с нашей процедурой шифрования злую шутку: выдать потенциальному злоумышленнику часть конфиденциальной информации, которая облегчит ему процедуру взлома. Поэтому после окончания процедуры зашифровывания, "хвост" все же лучше почистить: дописать в файл столько нулей, чтобы длина его стала кратной размеру кластера, а затем поставить метку конца файла в нужном месте. Или использовать проверенные буферы для записи-считывания информации из файла.

Если речь идет о шифровании только с использованием своего секретного ключа, то на этом процедуру шифрования можно закончить, а зашифрованный файл закрыть. Его расшифрование будем осуществлять в таком порядке.

- Считываются 24 байта обобщенного марканта.
- Последние 8 байт расшифровываются с помощью процедуры `DecryptBlock` 2 раза, а средние 8—1 раз.
- Полученные три 8-байтовых вектора побайтно сравниваются. С учетом возможных искажений, на каждой координате из трех значений как минимум два должны совпадать. Если это условие не выполнено, значит, либо зашифрованный файл был сильно искажен, причем искажены именно последние 24 байта, либо у нас не те ключи для его расшифрования. Во всех этих ситуациях правильное расшифрование невозможно.
- Аналогично процедуре зашифровывания, строятся *одноразовые* подстановка π и значения k_i. С помощью находящейся в нашем криптографическом ядре процедуры `Decrypt` по частям расшифровываем весь файл и записываем расшифрованные значения на место зашифрованных.
- Обобщенный маркант больше не нужен, поэтому после расшифрования его из файла убираем.

Ну и, конечно же, при операциях зашифровывания-расшифрования файла можно осуществлять различные манипуляции с атрибутами файла. Здесь могут быть любые фантазии, эта часть с криптографией уже никак не связана. По своему опыту могу сказать, что удобно бывает отслеживать дату и время создания файла и следить за тем, чтобы при зашифровывании-расшифровании она не менялась. Придумывать различные хитрые имена зашифрованным файлам, на мой взгляд, не всегда целесообразно: у нас есть хороший криптографический критерий определения зашифрованных файлов — это обобщенный маркант.

А теперь рассмотрим некоторые особенности для того случая, когда мы используем систему с открытым распределением ключей. Основная процедура зашифровывания остается той же, но после ее выполнения целесообразно добавить в файл некоторую дополнительную служебную информацию:

- идентификатор ключа отправителя;
- идентификатор ключа получателя;
- открытый ключ отправителя.

Имея такую служебную информацию в зашифрованном файле, мы всегда будем уверены в том, что получатель сумеет его расшифровать, даже в том случае, если в его базе данных открытых ключей нет ключа отправителя. Он в этом случае не сможет проверить подлинность расшифрованного сообщения, но всегда может быть уверенным в том, что расшифрование прошло успешно. Мы будем называть эту информацию *открытым маркантом*. Вот пример открытого марканта:

```
*****ALBMSL  MANAGER 7269d9160e2030db7991ec290b625a0fe5240ce7...
```

Таким образом, к файлу в процессе зашифровывания всегда дописывается обобщенный маркант, а при использовании системы с открытым распределением ключей еще дописывается открытый маркант. И это еще не все.

Зашифрованный файл часто бывает необходимо подписать. Подпись файла также добавляется в конец файла. И если в процедуре расшифрования не предусмотреть возможности расшифровывать подписанный файл, то мы не сможем подписывать зашифрованные файлы. А иметь возможность подписи зашифрованных файлов во многих практических случаях целесообразно. Поэтому в процедуре расшифрования необходимо предусмотреть возможность отделения подписей и открытых маркантов от остального текста, т. е. выбрать такой формат подписи и открытого марканта, который позволял бы выделять их в конце текста с наименьшими затратами по времени.

Исходя из этих соображений, целесообразно для подписи и открытого марканта взять одну и ту же длину L, один и тот же признак начала, и по этому признаку, не обращая внимание на остальное содержание, выделять входящие в них символы в конце текста. Например, в качестве такого признака можно взять последовательность из 5 подряд идущих символов $*$. Тогда вероятность случайного появления в равновероятном шифртексте такого признака будет равна $(2^{-8})^5 = 2^{-40} = 10^{-12}$. Это неопасно. По признаку $*****$ мы будем определять начало подписи или открытого марканта. Процедура расшифрования, таким образом, должна начинаться с проверки наличия последовательности $*****$ на расстоянии L байт от конца файла, если это условие выполнено, то проверка этого же условия еще через L байт и т. д. Файл, вообще говоря, может содержать несколько подписей и все их необходимо отсечь.

Все подписи зашифрованного файла должны быть проверены *до начала* процедуры расшифрования. После расшифрования эти подписи будут заведомо неверными и ненужными, поэтому мы их убираем без ущерба для безопасности. Если в зашифрованном файле есть открытый маркант и подписи, то все подписи должны находиться *после* открытого марканта, причем открытый маркант может быть только один.

Открытый маркант при расшифровании играет вспомогательную роль, позволяющую ускорить процесс расшифрования. В принципе, имея базу данных с открытыми ключами абонентов, получатель зашифрованного сообщения

может перебрать все возможные варианты имеющихся у него открытых ключей и по обобщенному марканту определить требуемый ключ для расшифрования. Но такая процедура может быть очень громоздкой и долгой, а наличие открытого марканта значительно ускоряет этот процесс и, кроме того, позволяет расшифровывать файл даже в том случае, когда открытый ключ отправителя в базе данных отсутствует. После расшифрования открытый маркант, так же как и подписи зашифрованного файла, удаляется.

И еще об одной особенности, которую должен помнить разработчик и пользователь системы шифрования. Часто зашифрованный файл затем передается по каналам связи, а тогда для уменьшения объемов передаваемой информации его необходимо сжать, т. е. применить к нему стандартную операцию архивирования. И вот эту операцию обязательно необходимо сделать *до* зашифровывания. После шифрования текст в зашифрованном файле становится случайным и равновероятным, он теряет свою избыточность, и поэтому применять операцию архивирования к зашифрованному файлу практически бесполезно.

Вместе с процедурами зашифровывания-расшифрования можно создать отдельную процедуру проверки, являлся ли файл зашифрованным на данном ключе. Критерием опять же будет обобщенный маркант, а точнее упоминавшийся выше результат расшифрования трех его частей. Такая процедура бывает полезной для поиска зашифрованных файлов, создания процедур автоматической сортировки, выбора ключа для расшифрования. Ранее мы упоминали об одном таком возможном способе выбора открытого ключа из базы данных открытых ключей, однако в том случае использование открытого марканта оказывалось более предпочтительным. Но иногда имеет смысл отказаться от системы с открытым распределением ключей и использовать *ключевые таблицы* для парно-выборочной связи. Вот там уже критерий выбора секретного ключа для расшифрования, основанный на обобщенном марканте, будет играть первостепенное значение. Об этом подробнее пойдет речь в *части V*, посвященной автоматизированной системе электронного документооборота.

Глава 11

Процедура подписи

Электронная подпись отправляемых сообщений при развитии электронной коммерции стала иногда даже более значимой, чем шифрование. При шифровании предполагается наличие потенциального злоумышленника, который стремится прочитать зашифрованное сообщение, осуществляет контроль за линией связи, использует при этом дорогостоящую аппаратуру для перехвата сообщений, достаточно ограничен в возможностях. Подпись же необходима даже в том случае, если потенциальных злоумышленников нет вообще. Из двух деловых партнеров один может оказаться недобросовестным и отказаться от взятых на себя обязательств. Если эти обязательства принимались по электронной почте, то другой партнер при наличии электронной подписи сможет доказать свою правоту и потребовать их выполнения.

При передаче информации по каналу связи с помощью подписи можно осуществлять контроль за правильностью приема отправленных сообщений, отсутствием в них искажений, вызванных случайными помехами в канале связи. Подпись может выполнять также функции контроля за сохранностью файлов, гарантировать их неизменность, выявлять воздействие на файл компьютерных вирусов, т. е. выполнять все функции обычной проверочной комбинации типа контрольной суммы, только гораздо более сложной для вычисления и проверки. Проблемам вычисления и проверки подписи будет посвящена эта глава.

В *главе 6* мы уже рассматривали общие вопросы, связанные с осуществлением и проверкой электронной подписи. Сейчас настало время углубиться в детали этих процессов, особенности их реализации с помощью процедур, входящих в наше криптографическое ядро. Начнем с хэш-функции, общие принципы построения которой были изложены в *главе 6*. Рассмотрим ее программную реализацию.

Программная реализация хэш-функции

```
FILE * fl;
unsigned char Hash[64];   // значение хэш-функции
unsigned char Buff1[256]; // буфер № 1 для вычисления хэш-функции
unsigned char Buff2[256]; // буфер № 2 для вычисления хэш-функции

// подстановка, используемая для вычисления хэш-функции
unsigned char Pi[256];

// контрольный массив, используемый при построении подстановки
bool PiCnt[256];
unsigned char empt,x[8],x_c[8],k[100]; // вспомогательные переменные
int i,j,i_off;
long FileLength; // длина хэшируемого файла
TTime * Time;
TTime * TimeStart;
TTime * TimeEnd;
unsigned short Hour,Min,Sec,MSec;

  Time = new TTime();
// Запоминаем время в начале эксперимента
  TimeStart = new TTime(Time->CurrentTime());

// Задаем начальную точку генератора случайных чисел
  srand(8601);
  memset(PiCnt,0,256); // Обнуление контрольного массива

// Сначала строим подстановку Pi
  for(i=0;i<256;i++)
  {
    empt=(unsigned char)rand();
    while(PiCnt[empt]==true)empt++;
    Pi[i]=empt;
    PiCnt[empt]=true;
  }
// Начальное значение параметра для CryptBlock
  for(i=0;i<100;i++)
    k[i]=(unsigned char)rand();

// Открываем хэшируемый файл
  fl=fopen("TEST","rb");
  FileLength = filelength(fileno(fl));
```

```
// Определяем количество байт, которые надо будет считать в первый буфер
  if(ftell(fl)+256<FileLength)
    j=256;
  else
    j=FileLength-ftell(fl);

// Считываем данные в первый буфер
  fread(Buff1,j,1,fl);
// Если в файле было менее 256 байт
  if(j<256)
    for(i=j;i<256;i++)
    Buff1[i]=(unsigned char)rand();

// Основной цикл вычисления хэш-функции
main_ckl:
// Определяем количество байт, которые надо будет считать во второй буфер
  if(ftell(fl)+256<FileLength)
    j=256;
  else
    j=FileLength-ftell(fl);

  // Считываем данные во второй буфер
  fread(Buff2,j,1,fl);
  // Если в файле осталось менее 256 байт
  if(j<256)
    for(i=j;i<256;i++)
    Buff2[i]=(unsigned char)rand();

  j=0;     // Счетчик числа блоков в первом буфере. Всего 64 блока
  i_off=248; // Указатель блока в первом буфере

// Цикл по всем блокам в первом буфере
  while(j<64)
  {
    memcpy(x,Buff1+i_off,8);
    for(i=0;i<KLENGTH;i++)
    k[i]+=Buff2[(unsigned char)((j<<2)+i)];
    CryptBlock(x,Pi,k,KLENGTH); // преобразовали считанный блок
    memcpy(x_c,x,8);      // запомнили преобразование
```

```
    CryptBlock(x,Pi,k,KLENGTH); // преобразовали еще раз

    // Меняем блок в первом буфере
    for(i=0;i<8;i++)
    Buff1[i_off+i]+=x_c[i];

    // Меняем еще 8 случайных точек в первом буфере
    for(i=0;i<8;i++)
      Buff1[x_c[i]]+=x[i];

    i_off-=8; // переход к следующему блоку
    j++;
  }

  // Этап окончен, первый буфер изменен. Меняем подстановку.

  memset(PiCnt,0,256);
  for(i=0;i<256;i++){
    while(PiCnt[Buff1[i]]==true)
    Buff1[i]++;
    Pi[i]=Buff1[i];
    PiCnt[Buff1[i]]=true;
  }

// Второй буфер переписываем на место первого
  memcpy(Buff1,Buff2,256);

// Процедура циклически повторяется
  if(ftell(fl)<FileLength)
    goto main_ckl;

// Первый проход завершен. Теперь считываем первый фрагмент файла во
// второй буфер и делаем еще один цикл

fseek(fl,0,SEEK_SET);

// Определяем количество байт, которые надо будет считать во второй буфер
  if(ftell(fl)+256<FileLength)
    j=256;
```

```
  else
    j=FileLength-ftell(fl);

// Считываем данные во второй буфер
  fread(Buff2,j,1,fl);
  fclose(fl); // файл больше не нужен

// Если в файле осталось менее 256 байт
  if(j<256)
    for(i=j;i<256;i++)
    Buff2[i]=(unsigned char)rand();

  j=0; // Счетчик числа блоков в первом буфере. Всего 64 блока
  i_off=248; // указатель блока в первом буфере

// Цикл по всем блокам в первом буфере
  while(j<64){
    memcpy(x,Buff1+i_off,8);
    for(i=0;i<KLENGTH;i++)
    k[i]+=Buff2[(unsigned char)((j<<2)+i)];
    CryptBlock(x,Pi,k,KLENGTH); // преобразовали считанный блок
    memcpy(x_c,x,8);            // запомнили преобразование
    CryptBlock(x,Pi,k,KLENGTH); // преобразовали еще раз

    // Меняем блок в первом буфере
    for(i=0;i<8;i++)
    Buff1[i_off+i]+=x_c[i];

    // Меняем еще 8 случайных точек в первом буфере
    for(i=0;i<8;i++)
    Buff1[x_c[i]]+=x[i];

    i_off-=8; // переход к следующему блоку
    j++;
  }

// Все этапы окончены, строим подстановку, зависящую от всего текста

  memset(PiCnt,0,256);
```

```
  for(i=0;i<256;i++){
    while(PiCnt[Buff1[i]]==true)
    Buff1[i]++;
    Pi[i]=Buff1[i];
    PiCnt[Buff1[i]]=true;
  }

// Строим итоговую хэш-функцию
// Она может быть практически любой разумной длины
// Здесь ее длина — 64 байта

  for(j=0;j<64;j+=8){
    memcpy(Hash+j,x,8);
    CryptBlock(x,Pi,k,100);
  }

// Запоминаем время в конце эксперимента
  TimeEnd = new TTime(Time->CurrentTime());
  delete Time;
  Time = new TTime(TimeEnd->Val — TimeStart->Val);
  Time->DecodeTime(&Hour,&Min,&Sec,&MSec);

  sprintf(Buff1,"\
  FileLength = %ld\n\
  Time: %d:%d:%d.%d",\
  FileLength,\
  Hour,Min,Sec,MSec);
  MessageBox(0,Buff1,"",MB_OK);
```

Результат работы этой процедуры показан на рис. 11.1.

Рис. 11.1. Результат работы функции хэширования

Условия эксперимента (тип процессора, операционная система и т. п.) те же, что и в аналогичном эксперименте из *главы 7*.

Таким образом, за почти 6 с вычислена хэш-функция для файла размером примерно 6,65 Мбайт. Скорость хэширования (с учетом операций считывания содержания файла с жесткого диска) составила около 1,13 Мбайт/с.

Для осуществления подписи, как мы видели в *главе 6*, нам нужны, кроме хэш-функции, еще секретный ключ и случайное число. Секретный ключ для подписи, как уже неоднократно отмечалось ранее, должен быть отдельный, не совпадающий с ключом для шифрования. Вырабатываться он может из единого базового ключа, его длина определяется тем способом осуществления операций возведения в степень, который мы выберем в криптографическом ядре. Остановимся на способе генерации случайного числа, необходимого для осуществления электронной подписи по методу Эль-Гамаля. Тут, естественно, вспомним маркант, использовавшийся при зашифровывании. Это почти готовое случайное число, необходимое для осуществления подписи. Его надо только несколько раз пропустить через процедуру `CryptBlock` с секретным ключом для осуществления подписи и получить таким способом случайное число требуемой длины.

Теперь рассмотрим некоторые вопросы, которые могут возникнуть в прикладных системах осуществления и проверки подписей.

Дата и время осуществления подписи

При осуществлении подписи, естественно, возникает желание проставить в ней дату и время ее осуществления, причем так, чтобы потенциальный злоумышленник, да и не только злоумышленник не смогли их потом подменить. При этом следует учитывать, что дата и время осуществления подписи могут не совпадать с датой и временем создания или изменения файла, а также то, что в одном файле может быть несколько различных подписей, осуществленных в различные моменты времени.

Одним из возможных вариантов решения этой проблемы является добавление даты и времени к хэш-функции после осуществления хэширования, но перед подписью:

```
getdate(&CurDay);
gettime(&CurTime);
Hash[0]^=CurDay.day;
Hash[1]^=CurDay.month;
Hash[2]^=CurDay.year;
Hash[3]^=CurTime.hour;
Hash[4]^=CurTime.min;
Hash[5]^=CurTime.sec;
Hash[6]^=CurTime.hund.
```

Сами значения даты и времени прописываются в подписи, а добавление к хэш-функции перед подписью является гарантией от их подделки.

Подпись, обычно, дописывается в конец файла. В прошлой главе мы уже рассматривали проблемы правильного размещения различной служебной криптографической информации в конце файла: обобщенного и открытого маркантов и подписей. На этапе проектирования криптографической системы необходимо определить удобные форматы и способы работы с ними. Необходим единый идентификатор начала подписи и открытого марканта. По опыту эксплуатации сложных автоматизированных систем, содержащих процедуры шифрования и подписи, удобно иметь возможность визуального контроля наличия открытого марканта и подписей. Следовательно, все входящие в подпись элементы должны отображаться на экране с помощью типового редактора. В подписи также должен присутствовать идентификатор ключа, на котором она была осуществлена, и, как уже упоминалось, дата и время осуществления подписи. В подписи, в качестве дополнительной защиты, может также присутствовать номер рабочего места, на котором она была осуществлена. Вот некоторые примеры подписей:

```
*****281101A01\001 132217031b2cfb643daca2ab4…186e0b542  000345
```

Подпись абонента A01\001, осуществлена 28.11.2001 в 13:22:17.03 на рабочем месте 000345.

```
*****211001A02C01 1352317798da46deee44b1aa7…054c84ee5ab 000231
```

Подпись абонента A02C01, осуществлена 21.10.2001 в 13:52:31.77 на рабочем месте 000231.

```
*****261101A01M01 15502271d7f4298fd590dd184…d4ca64a47e  000264
```

Подпись абонента A01M01, осуществлена 26.11.2001 в 15:50:22.71 на рабочем месте 000264.

Теперь обязательно нужно остановиться на проблеме нескольких подписей. А точнее, проблема сводится к тому, включать ли предыдущие подписи в хэш-функцию при хэшировании? Все дело в том, что в криптографической системе может быть предусмотрена функция *аннулирования* одной или нескольких подписей. В этом случае, если оставшиеся подписи защищали те, которые были аннулированы, то они при аннулировании автоматически становятся неверными. Если же предыдущие подписи не включать в хэш-функцию, то операция аннулирования любых предыдущих подписей не будет влиять на остальные подписи.

Следовательно, в реализации хэш-функции файла целесообразно предусмотреть возможность хэширования не всего файла, а до некоторого места в файле, например, до первой подписи. Признаком начала подписи, помимо идентификатора типа `*****`, может быть наличие на определенных позициях специфической информации, например, даты и времени подписи. Определение позиции в файле, до которой необходимо осуществлять хэширование, может осуществлять отдельная процедура, учитывающая различные

требуемые параметры. Иногда и сам пользователь в интерактивном режиме может определять, требуется ли ему защищать своей подписью все предыдущие подписи.

Основные особенности, связанные с процедурами проверки подписей, мы уже рассматривали ранее в *главе 6*, посвященной общим вопросам организации процедуры проверки подписей. Сейчас же немного порассуждаем о том, что нам потом делать с результатами проверки подписей.

Если процедура проверки подписей используется в интерактивной программе, когда пользователь сам визуально контролирует все происходящие криптографические процессы, то результат проверки подписей просто отображается на экране и пользователю, в случае, если проверка дала отрицательные результаты, предлагается самому отреагировать на них. Но очень часто процедура проверки подписи используется в различных автоматизированных системах, когда алгоритм реагирования на отрицательный результат проверки одной или нескольких подписей должен быть четко определен заранее. И в этом случае надо заранее описать все возможные ситуации, когда проверка подписи не дает положительного результата, и предложить те или иные действия во всех этих случаях. Дело в том, что, например, при организации пересылки платежных документов, подтвержденных электронными подписями, случайная ошибка при их проверке может вызвать задержку прохождения платежа и все связанные с этим негативные последствия. Итак, в каких случаях мы не получаем положительного результата проверки подписи и какие возможные варианты дальнейших действий можно предложить.

- Равенство $a^s = r^m \times (a^x)^r$, требуемое при проверке подписи по методу Эль-Гамаля, не выполняется. Это может быть вызвано или искажением текста (изменилась хэш-функция m), или неправильно выбранным открытым ключом a^x. Такая ситуация часто бывает при смене ключей, когда проверка подписи осуществляется с помощью базы данных открытых ключей, в которой по каким-то причинам замена старого ключа на новый произведена не была. Однозначно определить причину невыполнения равенства (хэш-функция или открытый ключ) нельзя, однако сообщение может быть отложено в некоторое хранилище "сомнительных" сообщений, с тем чтобы после проверки используемых открытых ключей его можно было бы проверить снова.
- Открытый ключ, требуемый для проверки подписи, не найден. Причина может быть вызвана или сбоем при обращении к базе данных с открытыми ключами, либо получением постороннего сообщения, предназначенного другому адресату. В последнем случае его, по-возможности, следует направить по назначению (вариант так называемой транзитной пересылки), эта ситуация может быть "штатной", заранее предполагаемой в системе электронного документооборота. Если данная ситуация вызвана

сбоем при обращении к базе данных с открытыми ключами, то сообщение в данном случае браковать не следует, необходимо найти причину сбоя и устранить ее. Это может быть сбой в локальной сети, одновременное использование одной и той же базы данных с открытыми ключами несколькими пользователями, ошибка при открытии файлов с базой данных и т. п.

- Проверяемая подпись в файле не найдена или искажена ее структура. Эта ошибка может быть вызвана сбоями при передаче файла по каналу связи или воздействием на файл компьютерных вирусов. Проверить подпись такого файла уже нельзя, его нужно или удалить, или скопировать в специальную папку для дальнейшего выяснения причин отсутствия или искажения структуры подписи.

В любом случае, при организации процедуры проверки подписей важных электронных документов в автоматизированной системе электронного документооборота при отсутствии положительного результата проверки одной или нескольких подписей проверку следует повторить несколько раз, чтобы убедиться в том, что отрицательный результат появляется постоянно. Дальнейшее разбирательство причин появления отрицательного результата осуществляется, как правило, в интерактивном режиме, при непосредственном участии оператора, обслуживающего систему электронного документооборота.

Часть III

Встраивание гарантированных алгоритмов в Microsoft Outlook

Глава 12. Основы MAPI

Глава 13. Шифрование и подпись объектов в хранилищах MAPI

Глава 14. Организация закрытой почты

Глава 15. Организация хранилища открытых ключей

В *части I* мы познакомились с основами криптографии. В *части II* рассмотрели общие вопросы, которые возникают у разработчика прикладной криптографической системы. В этой части мы рассмотрим способ построения конкретной криптографической системы на базе Messaging Application Programming Interface (MAPI) — интерфейса программирования приложений электронных сообщений.

Архитектура MAPI разрабатывалась так, чтобы обеспечить гибкость использующих его программных решений. Компания Microsoft заняла лидирующее положение в продвижении стандарта MAPI и всячески поощряет разработку продуктов MAPI независимыми производителями программного обеспечения. Microsoft сделала Windows универсальной платформой электронных сообщений, интегрировав подсистему MAPI в базовые компоненты операционных систем Windows. Эти компоненты представляют собой динамически подключаемые библиотеки (DLL), которые загружаются в память для того, чтобы можно было запускать и выполнять приложения MAPI. Хотя эти ключевые компоненты принадлежат Microsoft, большая часть MAPI общедоступна. Любой разработчик программ может создавать компоненты MAPI и свободно распространять их без каких-либо лицензий или отчислений.

Стандарт MAPI используется в широко распространенной системе электронной почты Microsoft Outlook. Общедоступность архитектуры MAPI, возможность встраивания в нее собственных приложений являются удобной платформой для разработчика прикладной криптографической системы при организации закрытой почты. Используя все сервисные возможности Microsoft Outlook, мы дополняем их возможностями осуществлять шифрование и электронную подпись отправляемых сообщений. У пользователя такой системы, привыкшего к Microsoft Outlook, добавление нескольких новых пунктов в меню не вызовет особенных затруднений и облегчит работу с остающимися для многих загадочными функциями шифрования и подписи.

Целью этой части книги является разработка специализированных криптографических компонент MAPI, в которых мы будем использовать упоминавшееся в *части II* криптографическое ядро и общие принципы построения криптографической системы с гарантированными свойствами.

Глава 12

Основы MAPI

Подробную информацию об основах MAPI можно почерпнуть из книги Ирвинга де ла Круза и Леса Талера "Основы MAPI" (Ирвинг де ла Круз, Лес Талер. Основы MAPI/Пер. с англ. — М.: Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd.", 1997). В этой книге авторы отмечают, что "в первую очередь, MAPI — это спецификация архитектуры, а не библиотека". Среди всего многообразия компонентов MAPI необходимо определить те из них, которые будут представлять для нас интерес при построении криптографических приложений.

Компонент доступа к хранилищу сообщений

Различные приложения, использующие MAPI, имеют дело с базой данных сообщений, которую компонент доступа к хранилищу сообщений представляет своим клиентам как иерархию папок и сообщений. Папки — это объекты, которые могут содержать другие папки и сообщения. Клиент может создать в хранилище сообщений сколько угодно папок и использовать их так, как считает нужным.

Хотя DLL компонента доступа к хранилищу выполняется локально на машине пользователя, само хранилище, к которому он обращается, может располагаться как локально, так и на сервере.

Каждое сообщение в хранилище MAPI можно рассматривать как некоторый набор свойств, для описания и использования которых в MAPI существует ряд специфических правил. MAPI определяет для свойств структуру данных `SPropValue`, в которой содержатся:

- `ulPropTag` — 32-разрядное беззнаковое целое число, известное под названием *бирка свойства (property tag)*;
- `Value` — данные свойства.

Каждая бирка свойств состоит из двух частей. Младшие 16 разрядов бирки — это *тип свойства*, указывающие тип данных свойства. Старшие 16 разрядов — *идентификатор свойства*, который позволяет отличать свойства друг от друга.

Криптографические операции шифрования и подписи мы будем осуществлять над сообщениями в хранилищах MAPI. Каждое сообщение состоит из набора свойств, поэтому в отличие от предыдущей части, где мы имели дело только с файлами, здесь мы уже будем оперировать с отдельными свойствами сообщений в хранилищах MAPI. Не всякое свойство имеет смысл шифровать и подписывать. Для шифрования и подписи мы будем использовать только свойства типа `PT_STRING8` (001E) — строка 8-разрядных символов, заканчивающихся символом с кодом 0, и `PT_BINARY` (0102) — массив байтов, хранящийся вместе со своей длиной.

Давайте внимательно посмотрим файл MAPITAGS.H, описывающий всевозможные бирки для стандартных свойств различных объектов MAPI, и постараемся выделить из них те, которые наиболее подходят для шифрования и подписи. Результаты приводятся ниже в табл. 12.1.

Таблица 12.1. *Свойства объектов в хранилищах MAPI*

Диапазон значений идентификатора свойств	Назначение свойств	Примечание
0001 — 0BFF	Определенные MAPI свойства конверта	Из всех этих свойств следует выделить свойство `PR_SUBJECT` (0037001E) — тема сообщения. Оно часто будет являться объектом для шифрования и подписи
0C00 — 0DFF	Определенные MAPI свойства для получателя информации	Отметим здесь свойство `PR_SENDER_EMAIL_ADDRESS` (0C1F001E) — адрес электронной почты отправителя. Иногда имеет смысл защитить его электронной подписью
0E00 — 0FFF	Определенные MAPI и не пересылаемые свойства	
1000 — 2FFF	Определенные MAPI свойства содержания сообщения	Здесь несколько свойств могут вызывать интерес с точки зрения шифрования и подписи. Это `PR_BODY` (1000001E) — тело сообщения, `PR_RTF_COMPRESSED` (10090102) — тело сообщения в RTF-формате

Таблица 12.1 *(окончание)*

Диапазон значений идентификатора свойств	Назначение свойств	Примечание
3000 — 3FFF	Определенные MAPI свойства	Представляет интерес для подписи PR_EMAIL_ADDRESS (3003001E) — список адресатов, для шифрования и подписи — PR_ATTACH_DATA_BIN (37010102) — файлы-вложения

Мы постараемся построить нашу систему шифрования и подписи сообщений в хранилищах MAPI таким образом, чтобы эти операции можно было бы совершать с любыми свойствами типа PT_STRING8 и PT_BINARY, в том числе и нестандартными, которые мог назначить кто угодно. В MAPI предусмотрена и такая возможность, специально зарезервирован для этого диапазон значений идентификаторов от 6000 до 65FF. Необходимо отдельно запомнить перечень свойств, подлежащих шифрованию, отдельно — перечень свойств, подлежащих подписи, поскольку они могут не совпадать, а также дать возможность пользователю осуществлять настройку этих перечней в интерактивном режиме.

К входящим в состав сообщений MAPI свойствам мы еще неоднократно вернемся в этой части книги, а сейчас рассмотрим более подробно механизм подключения нашего криптографического приложения к MAPI. Для этого мы будем использовать технологию *Exchange Client Extensions*, подробно описанную в книге Michael C. Amundsen "MAPI, SAPI, TAPI Developer's Guide" (Sams; ISBN: 0672309289; 1st edition, October 1996).

Постараемся сформулировать те требования, которые мы хотели бы предъявить к нашему криптографическому приложению, встроенному в Microsoft Outlook. Это приложение всюду в дальнейшем мы будем называть *Криптоцентром*.

- При открытии Microsoft Outlook система должна запрашивать базовый секретный ключ пользователя. При правильном вводе базового секретного ключа в оперативной памяти должны сформироваться секретные ключи для шифрования с использованием и без использования системы с открытым распределением ключей, а также отдельный секретный ключ для подписи.
- В основное меню Microsoft Outlook должен быть добавлен новый пункт **Криптоцентр**, с помощью которого в интерактивном режиме можно было бы осуществлять криптографические операции.
- С помощью системы Криптоцентр хотелось бы иметь возможность осуществлять шифрование как для хранения сообщений или отдельных их

свойств в зашифрованном виде, так и для их передачи по каналу связи. В первом случае целесообразно использовать систему шифрования без открытых ключей, а во втором — с открытым распределением ключей.

- В интерфейс **Параметры** должен быть добавлен пункт **Криптоцентр**, с помощью которого можно было бы осуществлять упоминавшуюся ранее настройку параметров для шифрования и подписи.
- В интерфейс **Справка** должна быть добавлена справочная информация по системе Криптоцентр.

Криптографическое приложение мы разобьем на две отдельные динамические библиотеки. Первая (CENTRE.DLL), являющаяся Exchange Client Extension, будет отвечать за подключение к MAPI и загрузку в оперативную память индивидуального секретного ключа пользователя. Вторая библиотека (MCS_MAPI.DLL) будет автоматически подгружаться при загрузке первой библиотеки и содержать основные криптографические процедуры, выполняемые над различными свойствами сообщений MAPI.

В первой библиотеке мы будем строго соблюдать все правила построения Exchange Client Extension, изложенные в книге Michael C. Amundsen "MAPI, SAPI, TAPI Developer's Guide". Нет смысла пересказывать их здесь достаточно подробно, однако на некоторых чисто криптографических особенностях следует остановиться.

Ввод секретного ключа. Ввод в оперативную память базового секретного ключа осуществляется в точке входа в динамическую библиотеку CENTRE.DLL. Весь сеанс работы секретный ключ находится в памяти и готов к использованию. Пользователь имеет возможность в интерактивном режиме выгрузить секретный ключ из памяти, загрузить его вновь, выбрать другой секретный ключ и загрузить его. В системе "Криптоцентр" предусмотрен *секретный ключ по умолчанию.* Имя файла с таким ключом прописывается в реестре Windows, сам файл может находиться на дискете. Пользователю предоставляется возможность ввода как ключа по умолчанию, так и выбора и ввода произвольного другого секретного ключа. При запуске Microsoft Outlook запрашивается секретный ключ по умолчанию, как это показано на рис. 12.1.

Вот фрагмент библиотеки CENTRE.DLL, описывающей точку входа в нее.

Листинг 12.1. Точка входа в Exchange Client Extension

```
int WINAPI DllEntryPoint(HINSTANCE hinst, unsigned long reason, void*)
{

  // При выходе освобождается и вторая библиотека
  if(reason==DLL_PROCESS_DETACH&&hinstLib!=NULL)
```

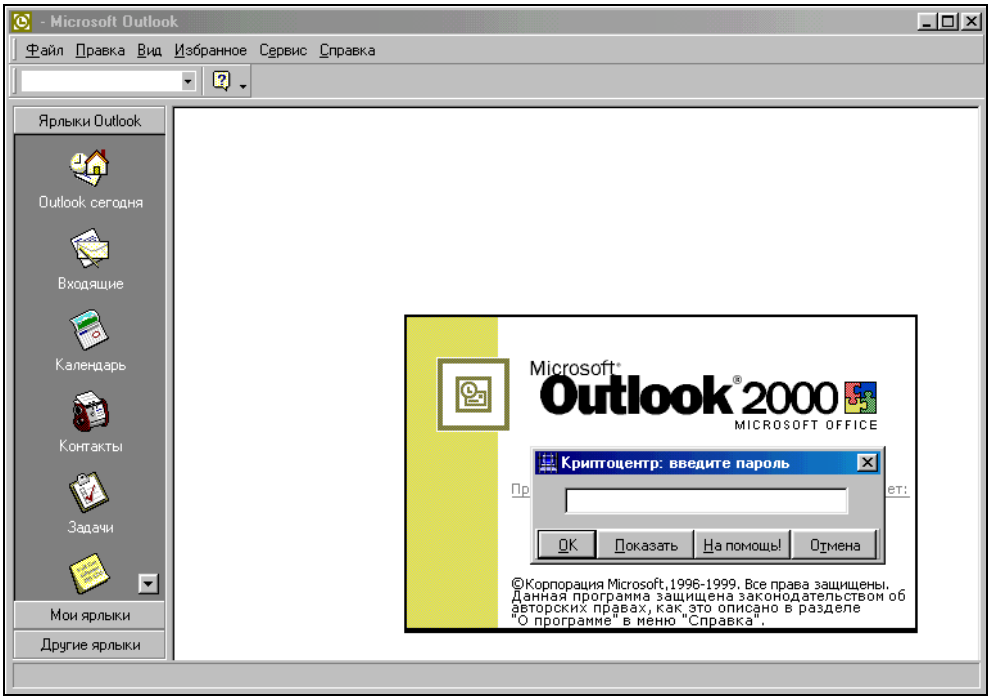

Рис. 12.1. Запуск Microsoft Outlook с системой Криптоцентр

```
FreeLibrary(hinstLib);

  // При входе автоматически загружается библиотека MCS_MAPI.DLL,

  // содержащая криптографические процедуры

   if (DLL_PROCESS_ATTACH == reason){
     ghInstDLL = hinst;

      hinstLib = LoadLibrary("MCS_MAPI.DLL");
      if (hinstLib == NULL){
        MessageBox(0,"Не найдена библиотека MCS_MAPI.DLL. Криптоцентр не
        установлен","ОШИБКА",MB_ICONERROR | MB_OK);
        Correct = false;
        return 1;
      }
```

```
    // Процедура ввода секретного ключа в оперативную память
    InputSecretKey();

    // Процедура ввода в оперативную память открытого ключа
    // менеджера СБ
    InputKeyMan();
  }
  return 1;
}
```

Некоторые пояснения к приведенному листингу. В нем описываются действия, выполняемые на первом этапе обращения к динамической библиотеке CENTRE.DLL. Таких действий здесь три.

1. Загрузка динамической библиотеки MCS_MAPI.DLL, содержащей криптографические процедуры.
2. Загрузка в оперативную память индивидуального секретного ключа пользователя (ее листинг будет приведен чуть позже).
3. Загрузка в оперативную память открытого ключа менеджера системы безопасности, необходимого для проверки сертификатов открытых ключей.

По результатам выполнения первых двух действий формируется значение глобальной переменной `Correct`, которая будет определять дальнейшую процедуру подключения Exchange Extension: если `Correct==false`, то подключается пустое расширение, не содержащее никаких новых функций.

Листинг 12.2. Подключение расширения "Криптоцентр"

```
extern "C" __declspec(dllexport) LPEXCHEXT CALLBACK ExchEntryPoint(void)
{
    if(Correct == true)
      return new CentreExt;
    else
      return new CentreExtNull;
}
```

Отметим, что значение `false` формируется в том случае, если вторая библиотека MCS_MAPI.DLL либо не найдена, либо не содержит требуемых процедур по вводу в оперативную память секретного ключа. Если же все необходимые процедуры в MCS_MAPI.DLL присутствуют, то независимо от

результатов ввода секретного ключа, значение `Correct` формируется как `true`. Пользователь сможет в этом случае ввести секретный ключ позднее.

В сам класс `CentreExt` мы включим 3 подкласса:

- `CentreExtCommands` — для формирования и выполнения криптографических команд;
- `CentreExtMessageEvents` — для обработки событий, связанных с сообщениями;
- `CentreExtPropertySheets` — для включения в окне **Параметры** дополнительной страницы **Криптоцентр**.

Назначение подкласса `CentreExtCommands` достаточно понятное: интерактивное выполнение различных криптографических процедур. С помощью процедуры `CentreExtCommands::InstallCommands` мы устанавливаем необходимое нам количество криптографических команд и вносим изменения в меню Microsoft Outlook. С помощью процедуры `CentreExtCommands::DoCommand` обеспечиваем выполнение установленных криптографических команд. С помощью `CentreExtCommands::Help` обеспечиваем вызов необходимых разделов файла помощи. Подкласс `CentreExtCommands` удобен в основном окне Microsoft Outlook, когда пользователь в интерактивном режиме выбирает сообщения и осуществляет с ними те или иные криптографические операции.

Подкласс `CentreExtMessageEvents` удобен в окне редактора сообщений. Например, с помощью процедуры `CentreExtMessageEvents::OnRead` (при наступлении события чтения сообщения) можно автоматически определять зашифрованные сообщения (по обобщенному марканту) и автоматически их расшифровывать при загрузке в редактор. С помощью процедуры `CentreExtMessageEvents::OnWriteComplete` (запись сообщения завершена) можно при необходимости зашифровать сообщение для его хранения в зашифрованном виде.

С помощью подкласса `CentreExtPropertySheets` к интерфейсу **Параметры** основного меню Microsoft Outlook подключается дополнительная страница **Криптоцентр**. На ней пользователь имеет возможность осуществить настройку перечня свойств, подлежащих зашифровыванию и подписи, установить файл с открытым ключом менеджера СБ и другие параметры криптографической системы.

Давайте обсудим, какие криптографические команды целесообразно установить и как затем их использовать.

Команды, связанные с секретными ключами

Приведем листинг упоминавшейся ранее процедуры ввода базового секретного ключа в оперативную память.

Листинг 12.3. Загрузка в оперативную память секретных ключей

```
void InputSecretKey()
{
  int i;
  char buff[128];
  GETKEYNAMEFROMREESTR ProcAdd1;
  KEYINP ProcAdd2;

 // При входе в программу ключи и имя файла с ключом обнуляются
  memset(keyname,0,128);
  memset(KeyLabel,0,8);
  memset((char *)sign_key,0,80);
  memset((char *)crypt_key,0,80);
  memset(pi,0,256);
  memset(k,0,80);

  // Имя файла с ключом считывается из реестра Windows
  ProcAdd1 = (GETKEYNAMEFROMREESTR)GetProcAddress(hinstLib,
  "_GetKeyNameFromReestr");

  if (ProcAdd1 == NULL)
    {
      Correct = false;return;
    }
    else
    {
      (ProcAdd1)(keyname);
      if(keyname[0]!=0)
    {
  // Если файл с ключом считывается с дискеты, то выдается приглашение
  // вставить в дисковод дискету
  if(keyname[0]=='A'||keyname[0]=='a'||keyname[0]=='B'||keyname[0]=='b')
  {
    sprintf(buff,"Вставьте в дисковод %c дискету с ключом",keyname[0]);
    MessageBox(0,buff,"",MB_OK);
  }
  ProcAdd2 = (KEYINP)GetProcAddress(hinstLib,"_PrepearSecretKeyL");
  if (ProcAdd2 == NULL)
```

```
  {
    Correct = false;return;
  }
  Correct = true;
  // Процедура считывания базового секретного ключа и формирование
  // секретных ключей для подписи и шифрования
  (ProcAdd2)(keyname,sign_key,crypt_key,KeyLabel,pi,k);
}
```

В результате выполнения этой процедуры имеем:

- секретный ключ для шифрования сообщений для их хранения в зашифрованном виде: пара (`pi,k`) — подстановка из S_{256} и вектор;
- секретный ключ для шифрования с использованием системы с открытым распределением ключей: `crypt_key`;
- секретный ключ для осуществления подписи: `sign_key`.

Имя файла с секретным ключом в этой процедуре считывается из реестра Windows, т. е. эта процедура осуществляет ввод секретного ключа по умолчанию. Но поскольку различных пользователей может быть несколько и у каждого есть свой секретный ключ, то целесообразно было бы также добавить аналогичную процедуру ввода произвольного секретного ключа. Ее отличие будет в том, что в ней имя файла с секретным ключом пользователю предлагается выбрать в интерактивном режиме самостоятельно.

Чтобы сделать систему Криптоцентр независимой, в перечень команд, связанных с секретными ключами, целесообразно также добавить команду выработки новых секретных и открытых ключей. Модуль выработки (здесь мы называем его *MCS-key*) может быть аналогичным тому, о котором шла речь в *главе 8*, он либо встраивается в библиотеку MCS_MAPI.DLL, либо придается в качестве независимого модуля.

В целях обеспечения безопасности, пользователь должен в любой момент иметь возможность *выгрузки* секретного ключа из оперативной памяти. В *главе 7* мы уже упоминали о том, что наличие подключенных коммуникационных устройств в момент нахождения секретного ключа в оперативной памяти представляет потенциальную угрозу сканирования памяти и компрометации ключа. Поэтому, осуществив в *предварительном режиме* все криптографические операции, требующие наличия в памяти секретных ключей, пользователь может затем выгрузить их из памяти и пересылать зашифрованные сообщения.

В зависимости от потребностей пользователя, можно установить и другие команды, связанные с секретными ключами: ввод секретного ключа из специализированных устройств типа *Touch Memory*, *Finger ID*. Для контроля можно задать команду на просмотр метки текущего ключа, находящегося

в оперативной памяти. На рис. 12.2 приведен пример организации команд, связанных с секретными ключами, в основном меню Microsoft Outlook.

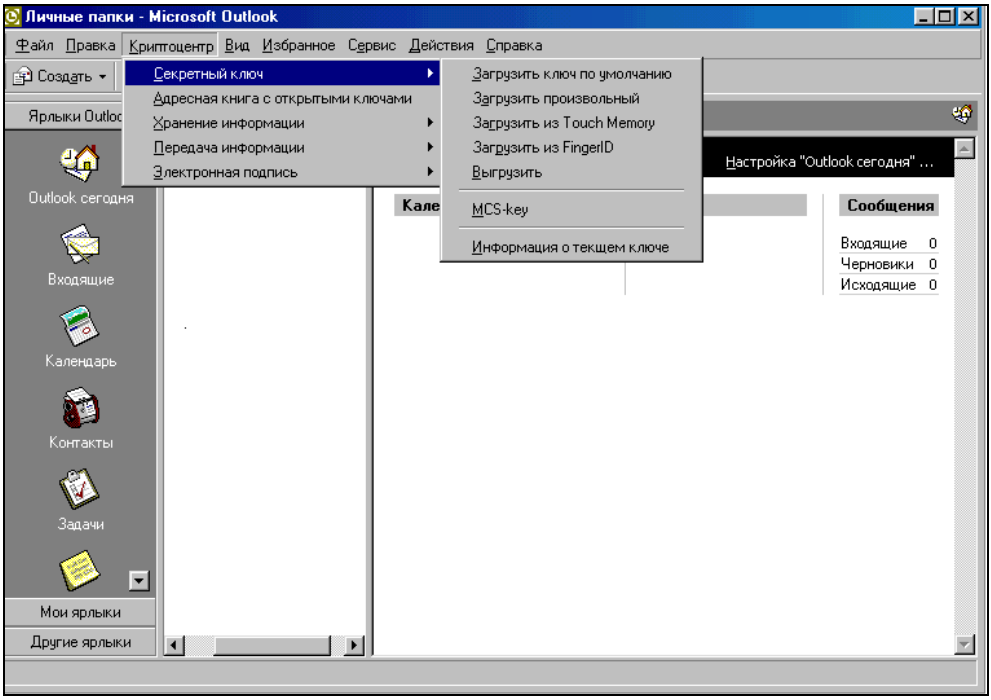

Рис. 12.2. Команды, связанные с секретными ключами системы Криптоцентр, в основном меню Microsoft Outlook

Команды, связанные с шифрованием без использования системы с открытым распределением ключей

Более подробно о таком способе шифрования мы поговорим в следующей главе, сейчас же отметим некоторые связанные с этими операциями особенности.

Как уже отмечалось раньше, мы планируем шифровать только свойства типа `PT_STRING8` и `PT_BINARY`. При этом предполагается, что любое значение свойства типа `PT_STRING8` может быть отображено на экране. Следовательно, зашифрованное свойство этого типа также должно быть отображаемо на экране. Таким образом, шифртекст должен состоять только из отображаемых на экране символов. Применим для решения этой проблемы тот же метод, что и при представлении подписи и открытого марканта:

каждый байт будем отображать парой HEX-символов от 00 до FF. Произойдет более чем 2-кратное увеличение длины шифруемого текста, однако при хранении информации это часто не является непреодолимой преградой.

Для свойства типа `PT_BINARY` отображения на экране, как правило, не требуется, поэтому шифртекст здесь не нуждается в особом представлении.

Перечень свойств сообщения, подлежащих шифрованию, мы задаем в диалоговом окне **Параметры**. Однако может возникнуть ситуация, когда нам потребуется зашифровать свойство, идентификатора которого мы не знаем. Ведь в хранилищах MAPI могут храниться не только сообщения, пересылаемые по электронной почте (сообщения класса `IPM.Note`), но и контакты (класс `IPM.Contact`), заметки (класс `IPM.StickyNote`) и другие классы. Например, в контактах нам потребуется зашифровать номер телефона указанного лица, в дневнике может потребоваться зашифровать определенную запись и т. п. Поэтому при проектировании криптографических команд имеет смысл сделать процедуру интерактивного выбора свойства и его зашифровывания. На рис. 12.3 приведен пример реализации такой процедуры.

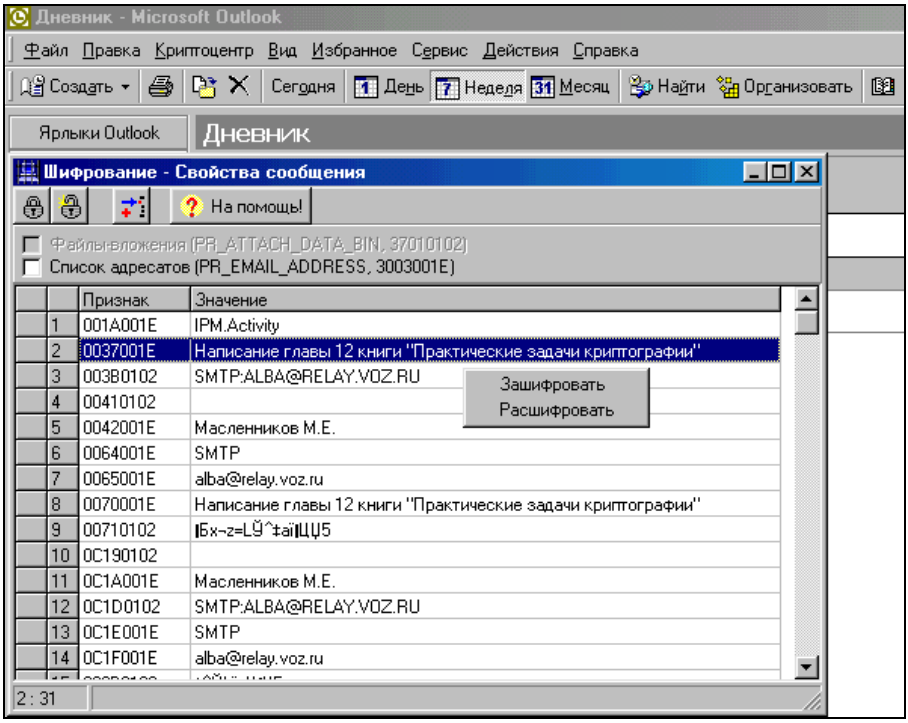

Рис. 12.3. Выбор и зашифровывание свойства в интерактивном режиме

В "штатном" режиме зашифровывания перечень свойств, подлежащих зашифровыванию, выбирается, как уже отмечалось выше, из окна **Параметры**.

Идентификаторы этих свойств могут быть прочитаны в момент загрузки системы Криптоцентр, например, из реестра Windows. Чтение этих свойств, а также свойств, защищаемых подписью, из реестра Windows может быть осуществлено одновременно с процедурой `InputKeyMan` загрузки открытого ключа менеджера СБ.

Команды, связанные с пересылкой сообщений в защищенном режиме

Под пересылкой сообщений в защищенном режиме мы будем понимать такую пересылку, когда потенциальный злоумышленник не имеет возможности, в случае перехвата сообщения, восстановить его текст. По желанию пользователя, пересылаемое в защищенном режиме сообщение также может быть подписано.

Особенностью пересылки сообщений в защищенном режиме является то, что не все свойства сообщения являются пересылаемыми. Если же непересылаемое свойство защитить подписью, то после пересылки такого сообщения подпись становится неверной.

Для решения подобных проблем Microsoft придумала и опубликовала *TNEF — Transport Neutral Encapsulation Format* — не зависящий от транспорта формат. Это протокол инкапсуляции свойств сообщений MAPI в двоичный файл. Прежде чем сообщение прибудет в точку назначения, оно, по всей вероятности, пройдет один или несколько шлюзов, и каждый из них будет копировать сообщение, определять путь до следующей точки и затем снова пересылать сообщение. Весьма велика опасность, что один или несколько промежуточных шлюзов не понимают MAPI, что может привести к искажению содержания сообщения.

Если же все подписываемые свойства сообщения собраны в один файл, а этот файл, по аналогии с TNEF, пересылается в качестве файла-вложения, то в этом случае мы можем быть уверены в правильности восстановления сообщения из файла. Сам файл также может быть подписан, чтобы была гарантия правильности его приема. По некоторым вспомогательным признакам (например, по специфической теме сообщения, в котором в качестве зашифрованного файла-вложения присутствует сообщение, пересылаемое в защищенном режиме) можно создать систему автоматизированного распознавания и раскрытия сообщений, пересылаемых в защищенном режиме.

Более подробно о пересылке сообщений в защищенном режиме еще пойдет речь в *главе 14*.

Команды, связанные с подписью

Для этих команд в MAPI также возникает ряд специфических требований. Под саму подпись целесообразно выделить отдельное свойство из упомянутого ранее перечня свойств, зарезервированных в качестве нестандартных и предназначенных для независимых разработчиков приложений MAPI. Но здесь мы сразу же сталкиваемся с проблемой нескольких подписей. Определенное нами свойство **Подпись** должно допускать возможность наличия в нем нескольких записей, соответствующих различным подписям. Но и это еще не все. Вместе с подписью в каждой такой записи должен храниться *хэш-вектор* с перечнем свойств, на котором была осуществлена подпись. Здесь проявляется особенность процедур осуществления и проверки подписей, их отличие от процедур зашифровывания/расшифрования сообщений в хранилищах MAPI. При осуществлении расшифрования информации, хранящейся в зашифрованном виде, мы могли перебрать все свойства сообщения и по обобщенному марканту определить для каждого свойства, было оно зашифровано или нет. Мы могли шифровать различные свойства в разные моменты времени, при расшифровании все они могут быть успешно расшифрованы за один раз. При осуществлении подписи, в момент вычисления хэш-функции мы должны четко знать, какие свойства подлежат хэшированию. Перечень свойств, защищенных подписью, должен храниться вместе с самой подписью, иначе потом нельзя будет проверить подпись. Но свойств, защищаемых подписью, может быть, вообще говоря, очень много, поэтому их перечень может занимать достаточно много места. Поэтому или при разработке криптографической системы подписи сообщений MAPI необходимо предусмотреть возможность наличия в свойстве **Подпись** записей различной длины, или же ввести ограничения на максимальное число свойств, защищаемых за один раз подписью, и тогда появляется возможность сделать все записи в свойстве **Подпись** одинаковой длины. С точки зрения удобства и скорости работы криптографической системы одинаковая длина записей в свойстве **Подпись** предпочтительнее. Если же возникнет потребность защитить очень большое число свойств подписью, то их в этом случае можно будет защитить не одной, а несколькими подписями одного и того же абонента.

Аналогично со случаем шифрования для хранения в зашифрованном виде, при осуществлении подписи может оказаться удобной процедура интерактивного выбора свойств, защищаемых подписью. Однако здесь отличие от случая шифрования состоит в том, что надо выбрать сразу набор свойств, защищаемых подписью. На рис. 12.4 приведен пример реализации процедуры осуществления подписи в интерактивном режиме.

Более подробно об операциях осуществления и проверки подписи свойств сообщений MAPI пойдет речь в *главе 13*.

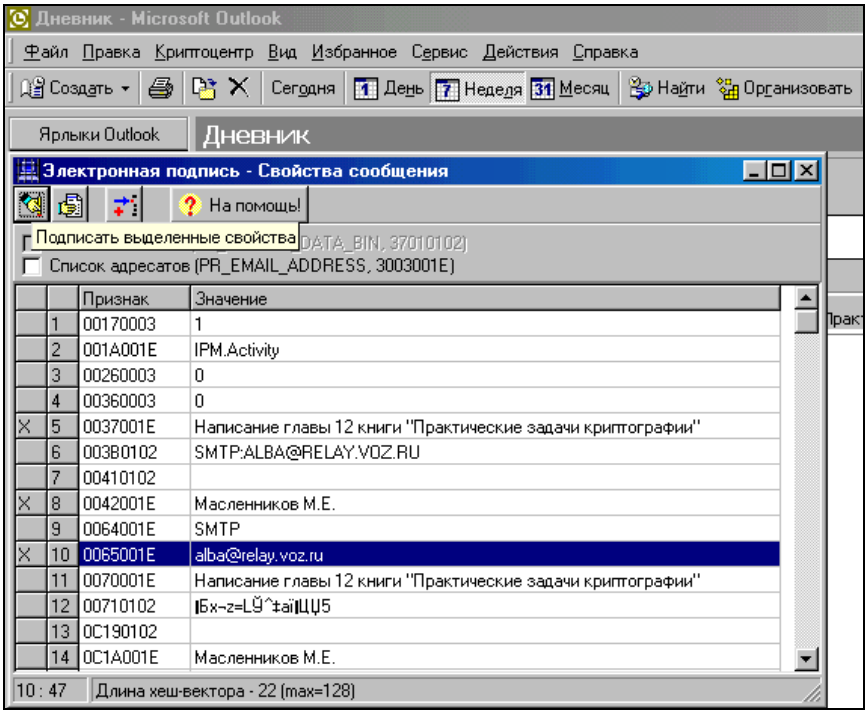

Рис. 12.4. Процедура осуществления подписи в интерактивном режиме

Теперь необходимо упомянуть еще об *адресной книге* как о потенциальном месте хранения открытых ключей, хотя возможно также построение независимого хранилища открытых ключей. Удобство использования адресной книги для хранения открытых ключей заключается в том, что при разработке интерфейса для работы с открытыми ключами в этом случае мы можем использовать существующие в MAPI процедуры обращения к адресной книге и готовый интерфейс для работы с ней. Задача разработчика криптографической системы сводится к тому, чтобы выбрать в адресной книге подходящее место для хранения открытого ключа и организовать процедуры обращения к нему. Примером такого места может служить свойство `PR_COMMENT` (3004001E), соответствующее разделу **Примечания** сервиса **Контакты**. На рис. 12.5 приведен пример организации интерфейса и записи открытого ключа в адресную книгу.

Формат открытого ключа здесь тот же, что и для упоминавшейся в *главе 9* базы данных с открытыми ключами при работе с файлами.

Однако запись открытых ключей в адресную книгу имеет и некоторые недостатки. Во-первых, свойство `PR_COMMENT` записи в адресной книге будет использоваться не по основному назначению, и это может вызвать неудобства у пользователя. Во-вторых, в некоторых версиях Microsoft Outlook

изменения в адресной книге могут быть ограничены и пользователь не сможет внести в нее требуемый открытый ключ и, тем самым, не сможет воспользоваться всеми возможностями криптографической системы. В таких случаях более целесообразным может оказаться создание независимой базы данных с открытыми ключами подобно той, которая была описана в *главе 9* для работы с файлами.

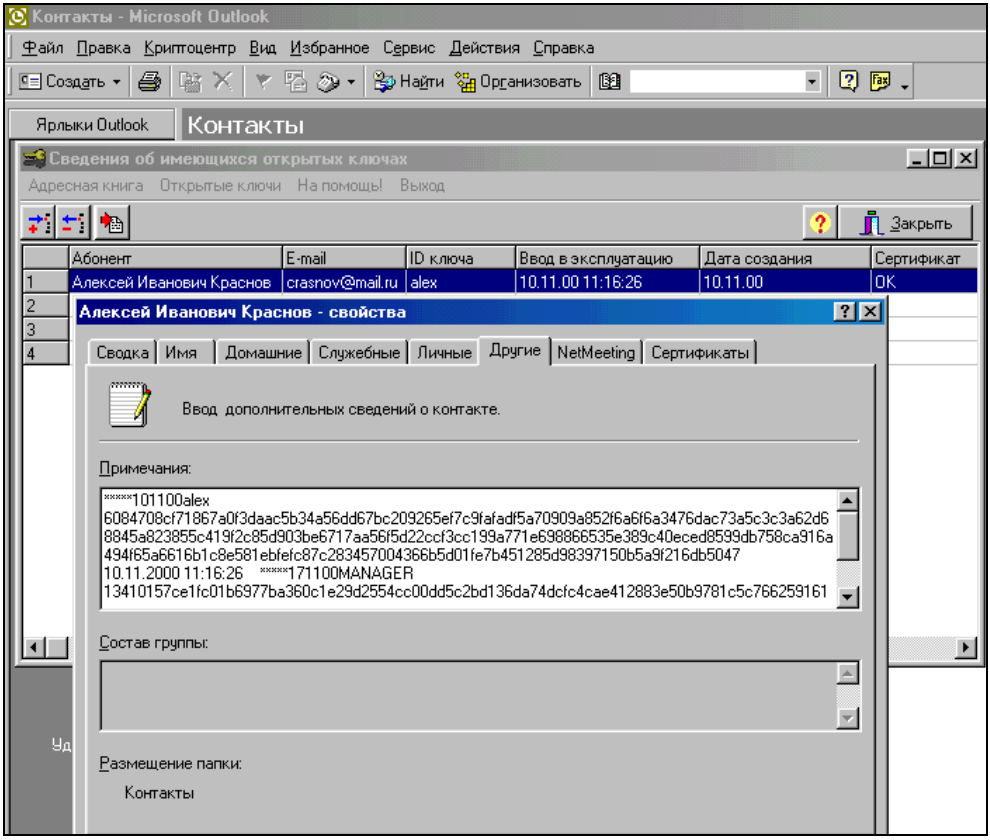

Рис. 12.5. Открытый ключ, записанный в адресную книгу

Суммируя все необходимые нам в дальнейшем сведения об основах MAPI, вкратце перечислим еще раз наиболее главные из них.

- Для подключения к MAPI мы используем механизм Exchange Client Extension, определяя в нем собственный класс `CentreExt`, содержащий подклассы, предназначенные для подключения и выполнения различных криптографических процедур.
- При работе с MAPI основными объектами являются различные свойства сообщений в хранилищах MAPI. Эти свойства могут быть различных

типов, содержать как полезную для пользователя информацию, так и некоторые вспомогательные служебные данные, которые пользователю незаметны.

- Для пересылки сообщений в защищенном режиме все его свойства собираются в один файл, который затем шифруется с использованием системы с открытым распределением ключей. При необходимости этот файл также может быть подписан. Для пересылки формируется новое сообщение, в котором сформированный файл является файлом-вложением.
- Для хранения информации в зашифрованном виде шифрованию без использования системы с открытым распределением ключей подвергается каждое свойство в отдельности. Зашифровывать мы будем только свойства определенных типов. При зашифровывании будем соблюдать правила, изложенные в *главе 10*, посвященной шифрованию файлов. В частности, в шифрованном тексте обязательно будет присутствовать обобщенный маркант, по которому программа определяет, являлось ли свойство зашифрованным.
- Перед осуществлением операции подписи любого сообщения в хранилищах MAPI необходимо установить перечень свойств, подлежащих подписи. Этот перечень свойств добавляется вместе с самой подписью в специально определенное для подписей свойство сообщения. Одно сообщение может иметь несколько подписей. Длины всех подписей одинаковы.
- В качестве базы данных с открытыми ключами в MAPI может выступать адресная книга, хотя в некоторых случаях целесообразно использовать независимую базу данных.

Глава 13

Шифрование и подпись объектов в хранилищах MAPI

В этой главе мы специально объединили вместе процедуры шифрования и подписи объектов в хранилищах MAPI. Под объектами в хранилищах MAPI будем подразумевать все доступные в основном меню Microsoft Outlook объекты: сообщения электронной почты, записи в сервисах **Контакты**, **Календарь**, **Дневник**, **Заметки** и т. п. Все они имеют один и тот же принцип построения и состоят из набора различных свойств. Под каждый сервис в Microsoft Outlook имеется собственная интерфейсная оболочка, которую мы хотим дополнить криптографическими функциями шифрования и подписи. И в первую очередь, хотелось бы сделать универсальную реализацию процедур шифрования и подписи, встраиваемую в основное меню Microsoft Outlook и не зависящую от выбранного пользователем конкретного сервиса, т. е. сделать такую криптографическую систему, чтобы мы могли шифровать и подписывать не только сообщения электронной почты, но и записи в других сервисах по единым правилам работы со свойствами объектов в хранилищах MAPI.

Немного углубимся в механизм Exchange Client Extension. В нем очень важную роль играет интерфейс `IExchExtCallback`. Он позволяет расширению, в частности, получать информацию о выделенных объектах в основном окне Microsoft Outlook. Указатель на этот интерфейс передается в качестве параметра в упоминавшейся в прошлой главе процедуре `CentreExtCommands::DoCommand` выполнения криптографических команд:

```
STDMETHODIMP CentreExtCommands::DoCommand (LPEXCHEXTCALLBACK lpeecb,
UINT cmdid)
```

Здесь `LPEXCHEXTCALLBACK` — указатель на интерфейс, а `cmdid` — идентификатор криптографической команды.

Остается с помощью этого интерфейса научиться перечислять выделенные объекты и осуществлять с ними операции шифрования и подписи.

Листинг 13.1. Алгоритм перечисления выделенных объектов MAPI

```
STDMETHODIMP CentreExtCommands::DoCommand (LPEXCHEXTCALLBACK lpeecb, UINT
cmdid){
LPMESSAGE pMsg;
LPMDB FAR pmdb;
HRESULT hRes;
ULONG FAR i_l;
ULONG FAR lpcbeid;
LPENTRYID FAR lppeid;
ULONG FAR lpulType;
ULONG cbMsgClass;
ULONG FAR lpulMsgFlags;

...

hRes=lpeecb->GetObject(&pmdb,NULL);
if(hRes!=S_OK)
{
MessageBox(0,"Ошибка при определении объекта","Криптоцентр",MB_ICONERROR
| MB_OK);
return(-1);
}

// Определяем количество выделенных объектов
lpeecb->GetSelectionCount(&i_l);
if(i_l<=0)
{
MessageBox(0,"Не выделены сообщения для
зашифровывания","Криптоцентр",MB_ICONERROR | MB_OK);
return(-1);
}

// Цикл по всем выделенным объектам
for(i=i_l-1;i>=0;i--)
{
hRes=lpeecb->GetSelectionItem((ULONG)i,&lpcbeid,&lppeid,\
&lpulType,NULL,cbMsgClass,&lpulMsgFlags,0);
if(hRes!=S_OK)
```

```
{
err_msg:
err_num++;
sprintf(buff,"Ошибка при определении сообщения № %d",i+1);
MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
continue;
}

// Открываем выделенный объект
hRes = pmdb -> OpenEntry(lpcbeid,lppeid,&IID_IMessage,\
MAPI_BEST_ACCESS,&lpulType,(LPUNKNOWN *)&pMsg);

if(hRes!=S_OK)goto err_msg;

// Теперь с открытым выделенным объектом можно осуществлять
// операции шифрования и подписи
   ...

}
```

Таким образом, основным объектом, над которым в дальнейшем мы будем осуществлять криптографические операции шифрования и подписи, является объект `MESSAGE`, указатель которого `LPMESSAGE` в конечном итоге выдает нам алгоритм перечисления выделенных объектов.

Отметим, что объект `MESSAGE` можно получить и при различных процедурах из подкласса `CentreExtMessageEvents`, связанных с обработкой определенных событий, упоминавшихся в прошлой главе. Например, при событии `CentreExtMessageEvents::OnRead`.

Листинг 13.2. Определение объекта MESSAGE по событию чтения сообщения

```
STDMETHODIMP CentreExtMessageEvents::OnRead(LPEXCHEXTCALLBACK lpeecb)
{
LPMESSAGE pMsg = NULL;
HRESULT hRes;

hRes=lpeecb->GetObject(NULL,(LPMAPIPROP *)&pMsg);
if(hRes!=S_OK)
{
MessageBox(0,"Ошибка при определении объекта","Криптоцентр",MB_ICONERROR | MB_OK);
```

```
return(-1);
}
...
}
```

Следующей операцией, которую необходимо выполнить, будет определение перечня свойств полученного объекта. Здесь можно использовать процедуру `GetProps`, входящую в состав класса `MESSAGE`.

Листинг 13.3. Определение перечня свойств объекта

```
LPSPropValue pspv = NULL;
ULONG cVals;
...
hRes = pMsg -> GetProps(NULL,0,&cVals,&pspv);
if(hRes<0)
{
MessageBox(0,"Ошибка!","Криптоцентр",MB_OK);
return(-1);
}
```

В результате выполнения процедуры `GetProps` мы получили количество свойств объекта (переменная `cVals`) и упоминавшуюся в прошлой главе структуру `SPropValue`, содержащую бирки свойств и их значения. Теперь можно заняться их шифрованием и подписью.

Для выполнения операции зашифровывания или подписи необходимо иметь вектор свойств, подлежащих зашифровыванию или подписи. Этот вектор может формироваться или из диалогового окна **Параметры**, или в интерактивном режиме. На рис. 13.1 приведен пример формирования вектора в окне **Параметры**.

Из этого рисунка видно, что, как для шифрования, так и для подписи выделены 6 наиболее часто встречающихся свойств, для которых в векторе предусмотрен только индикатор, т. е. значения 0 или 1. Таким образом, первые 6 значений в векторе могут быть только 0 или 1. Для всех остальных свойств их бирка включается в вектор полностью, т. е. каждое свойство добавляет в вектор 6-значное число в HEX-формате. Эти свойства высвечиваются при нажатии кнопки **Дополнительно**, как показано на рис 13.2.

Общая длина вектора не более 128. Таким образом, количество дополнительных свойств не превосходит 20.

Принцип построения вектора одинаковый как для шифрования, так и для подписи, хотя сами значения векторов для шифрования и подписи могут

быть различными. При подписи, как уже отмечалось в прошлой главе, вектор определяет перечень свойств, подлежащих хэшированию, и включается в саму подпись.

Подготовительные этапы к осуществлению процедур шифрования и подписи начинаются с анализа этих векторов, т. е. определения свойств, которые необходимо будет зашифровать или подписать.

Процедуры шифрования некоторых свойств имеют свою специфику, поэтому в программной реализации процедуры шифрования объекта их имеет смысл выделить особо. Отметим, что в памяти уже находятся секретные ключи для шифрования — это упоминавшаяся в прошлой главе пара (`pi,k`).

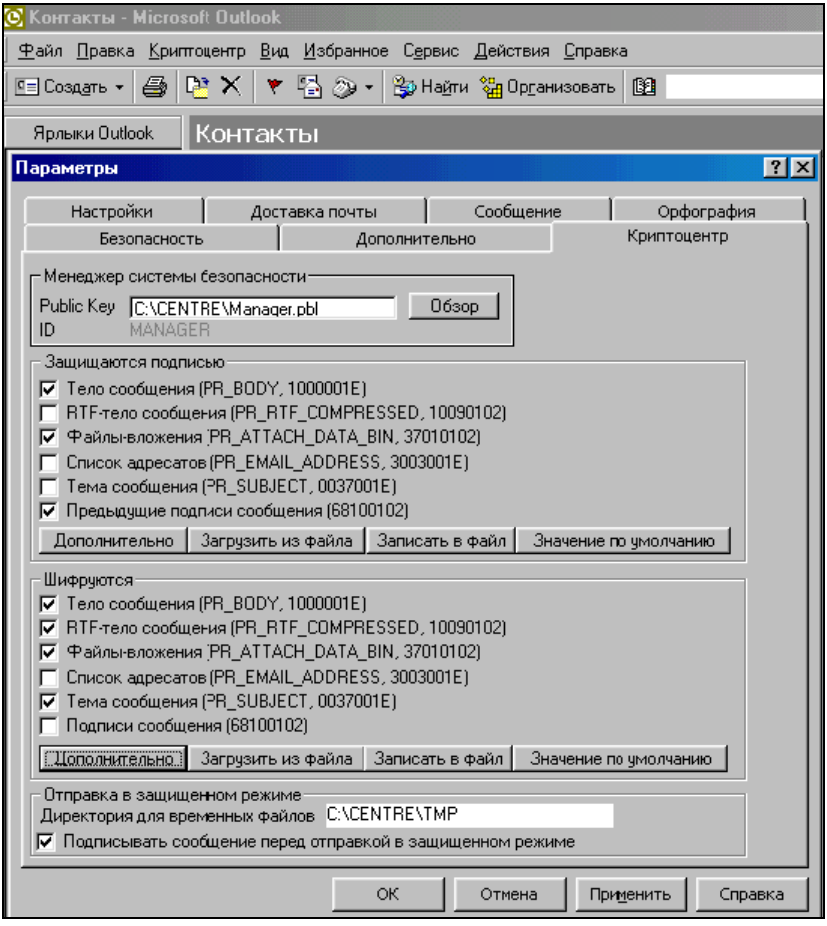

Рис. 13.1. Диалоговое окно **Параметры** системы Криптоцентр

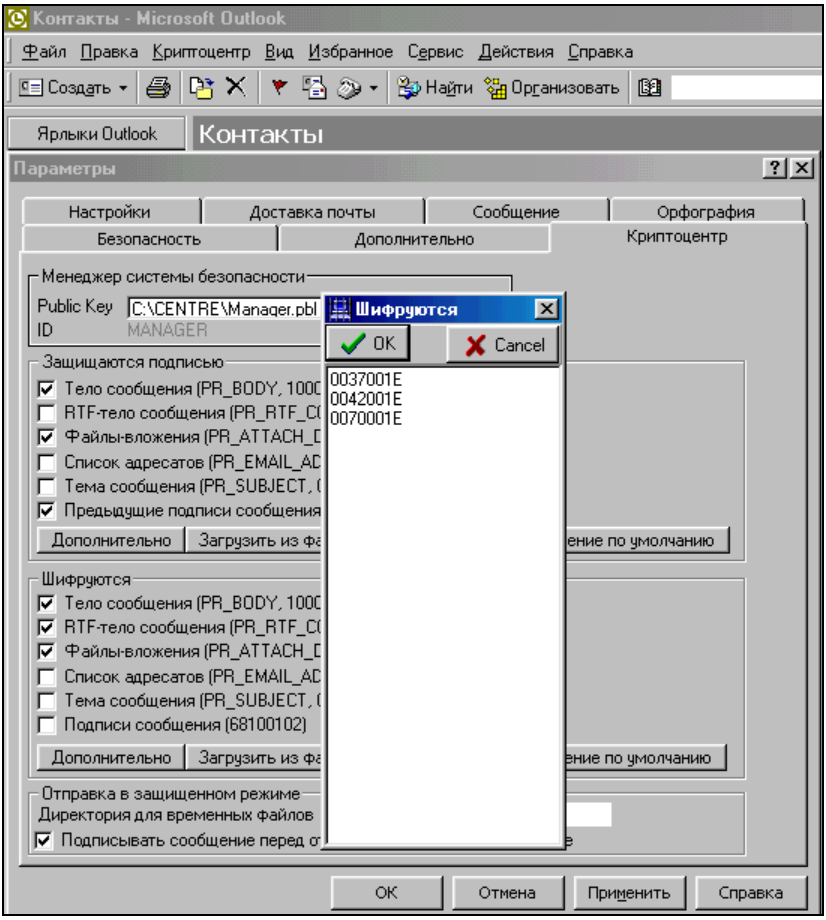

Рис. 13.2. Дополнительные шифруемые свойства

Шифрование вложений. Свойство `PR_ATTACH_DATA_BIN` присутствует в нашем векторе в качестве основного на третьем месте. В то же время в объекте MAPI предусмотрено булево свойство `PR_HASATTACH` (0E1B000B), принимающее значение `true` тогда и только тогда, когда объект имеет вложение. Определяем значение свойства `PR_HASATTACH` или с помощью перебора всех полученных ранее свойств, или с помощью функции `GetProps`, задавая в ней в качестве первого параметра структуру `SPropTagArray` вида `{1,PR_HASATTACH}`.

Листинг 13.4. Шифрование вложений

```
bool HasAttach = false;
for(i=0;i<(int)cVals;i++)
```

```
if(pspv[i].ulPropTag==PR_HASATTACH)
  HasAttach=pspv[i].Value.b;

//Теперь определяем условия, при которых вложения будут шифроваться

if(HasAttach==true&&CryptVect[2]=='1')
{
  // Получаем таблицу вложений
  hRes = pMsg->GetAttachmentTable(0,&pAttTbl);
  if(hRes!=0)
    goto err_msg;

  // Число строк в таблице вложений
  pAttTbl->GetRowCount(0,&i_l);
  pRow=NULL;
  if(i_l==0)
    return(-1);
  // Опрос строк в таблице вложений
  pAttTbl->QueryRows(i_l,0,&pRow);
  if(pRow==NULL)
    return(-1);

  // Цикл по всем Attachmentам
  for(i=0;i<(int)pRow->cRows;i++)
  {

    // Цикл по свойствам конкретного вложения
    for(j=0;j<(int)pRow->aRow[i].cValues;j++)
    {

      if(pRow->aRow[i].lpProps[j].ulPropTag==PR_ATTACH_NUM)
      {
        AttachNum=pRow->aRow[i].lpProps[j].Value.ul;
        // Открываем вложение
        hRes = pMsg->OpenAttach(AttachNum,&IID_IAttachment,
        MAPI_DEFERRED_ERRORS | MAPI_MODIFY,&lppAttach);
        if(hRes!=S_OK)
          goto err_msg;
```

```
        // Получаем свойства конкретного вложения
        hRes = lppAttach -> GetProps(NULL,0,&cVals1,&pspv1);

        // Запоминаем имя файла-вложения
        for(i1=0;i1<(int)cVals1;i1++)
          if(pspv1[i1].ulPropTag==PR_ATTACH_LONG_FILENAME)
            strcpy(LFilename,pspv1[i1].Value.lpszA);

        if(lppAttach->OpenProperty(PR_ATTACH_DATA_BIN,
        &IID_IStream, 0,MAPI_DEFERRED_ERRORS | MAPI_MODIFY,
        (LPUNKNOWN *) &pStream)==S_OK)goto read_att;
          if(lppAttach->OpenProperty(PR_ATTACH_DATA_OBJ,
          &IID_IStreamDocfile, 0,MAPI_DEFERRED_ERRORS | MAPI_MODIFY,
          (LPUNKNOWN *) &pStream)==S_OK)
            goto read_att;

        // Не удалось открыть вложение
        lppAttach->Release();
        // Переход к шифрованию тела сообщения
        goto body;

read_att:
        // Шифруем поток, соответствующий вложению
        i1=EncryptStream(LFilename,pi,k);
        if(i1==-1)
        {
          sprintf(buff,"Ошибка при зашифровывании вложения %s",LFilename);
          MessageBox(0,buff,"Криптоцентр",\
          MB_ICONERROR | MB_OK);
          pStream->Release();
          lppAttach->Release();
          break;
        }

        // Если зашифровывание потока было осуществлено успешно,
        // то в длинном имени файла меняем расширение
        if(i1==0)
        {
```

```
        if(strrchr(LFilename,'.')!=NULL)
          strcpy(strrchr(LFilename,'.')+1,"C_C");
        else
          strcat(LFilename,".C_C");

        CurProp.cValues = 1;
        CurProp.aulPropTag[0]=PR_ATTACH_LONG_FILENAME;
        pMsg->DeleteProps((LPSPropTagArray)&CurProp,NULL);

        pspv1[0].ulPropTag = PR_ATTACH_LONG_FILENAME;
        pspv1[0].Value.lpszA=LFilename;
        hRes=lppAttach->SetProps(1,pspvSaved,NULL);
        memset(buff,0,256);

        // Меняем значок вложения
        memset(buff,0,256);
        CurProp.cValues = 1;
        CurProp.aulPropTag[0]=PR_ATTACH_RENDERING;
        pMsg->DeleteProps((LPSPropTagArray)&CurProp,NULL);

        pspvSaved[0].ulPropTag = PR_ATTACH_RENDERING;
        pspvSaved[0].Value.bin.cb=256;
        pspvSaved[0].Value.bin.lpb=buff;
        lppAttach->SetProps(1,pspvSaved,NULL);
      }

      hRes=lppAttach->SaveChanges(0);
      if(hRes!=S_OK)return(-1);
        lppAttach->Release();
      break;
    }// if(...==PR_ATTACH_NUM)
  }// Цикл по свойствам конкретного вложения
 } // Цикл по вложениям
}
```

Каждое вложение мы открываем в виде потока данных, к которому применяем операцию зашифровывания потока (`EncryptStream`). Процедура `EncryptStream` строится так же, что и процедура зашифровывания файла, только вместо операций считывания/записи из файла в ней используются операции считывания/записи из потока. В конец зашифрованного потока

записывается обобщенный маркант. Шифртекст представляется в виде произвольных ASCII-символов.

В зашифрованном файле-вложении расширение изменяется на C_C. Это позволяет изменить иконку отображаемого файла на специальное изображение, соответствующее зашифрованному файлу. Отметим, что в объектах MAPI предусмотрено несколько свойств, связанных с именем файла и его расширением:

- PR_ATTACH_LONGFILENAME — длинное имя файла-вложения;
- PR_ATTACH_FILENAME — обычное имя файла-вложения;
- PR_ATTACH_EXTENSION — расширение в имени файла-вложения.

Мы изменили расширение только в PR_ATTACH_LONGFILENAME, сохранив старое расширение в остальных свойствах для того, чтобы восстановить его при расшифровании.

Шифрование тела сообщения. Тело сообщения может быть представлено или в обычном (PR_BODY) или в RTF-формате (PR_RTF_COMPRESSED).

Листинг 13.5. Шифрование тела сообщения в обычном формате

```
crypt_body:

// Размер тела увеличивается в два раза + обобщенный маркант (24x2)
mem=(char *)malloc(strlen(pspv[i].Value.lpszA)*2+48);

memset(mem,0,strlen(pspv[i].Value.lpszA)*2+48);
memcpy(mem,pspv[i].Value.lpszA,strlen(pspv[i].Value.lpszA));


i1=EncryptMem("Тело сообщения",mem,strlen(pspv[i].Value.lpszA),pi,k);

// Если тело ранее уже было зашифровано
if(i1==1)
  goto f_m;

if(i1==0)
{
  CurProp.cValues = 1;
  CurProp.aulPropTag[0]=PR_BODY;
  pMsg->DeleteProps((LPSPropTagArray)&CurProp,NULL);

  pspvSaved[0].ulPropTag = PR_BODY;
```

```
  pspvSaved[0].Value.lpszA=mem;
  hRes=pMsg->SetProps(1,pspvSaved,NULL);
  if(hRes!=S_OK) MessageBox(0,"Ошибка при записи зашифрованного тела\
  сообщения","Криптоцентр",MB_ICONERROR | MB_OK);
}
else
  MessageBox(0,"Ошибка при зашифровывании тела\
  сообщения","Криптоцентр",MB_ICONERROR | MB_OK);

f_m:
free(mem);
```

Здесь уже для зашифровывания используется процедура `EncryptMem`. Ее отличие от `EncryptStream` в том, что шифртекст (вместе с обобщенным маркантом) она представляет в "читаемом" виде — в виде последовательности HEX-символов от 0 до F. А для шифрования тела в RTF-формате этого не требуется.

Листинг 13.6. Шифрование тела сообщения в RTF-формате

```
if(pMsg->OpenProperty(PR_RTF_COMPRESSED, &IID_IStream,
0,MAPI_DEFERRED_ERRORS | MAPI_MODIFY, (LPUNKNOWN *) &pStream)==S_OK)
{
  i1=EncryptStream("Тело сообщения (RTF COMPRESSED)",pi,k);
  if(i1==-1)
  {
    MessageBox(0, "Ошибка при зашифровывании тела сообщения",\
    "Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }
}
```

Шифрование списка адресатов. Здесь, по аналогии с файлами-вложениями, приходится использовать специфические процедуры MAPI.

Листинг 13.7. Шифрование списка адресатов

```
// Получение таблицы адресатов
hRes = pMsg->GetRecipientTable(0,&pAttTbl);
if(hRes!=0)goto err_msg;
```

```
// Процедуры, аналогичные тем, которые использовались в таблице вложений
pAttTbl->GetRowCount(0,&i_l);
pRow=NULL;
if(i_l==0)goto subj; // адресатов нет
pAttTbl->QueryRows(i_l,0,&pRow);
if(pRow==NULL)return(-1);

// Цикл по всем адресам
for(i=0;i<(int)pRow->cRows;i++)
{
for(j=0;j<(int)pRow->aRow[i].cValues;j++)
pspvSaved[j]= pRow->aRow[i].lpProps[j];

// Цикл по свойствам конкретного вложения
for(j=0;j<(int)pRow->aRow[i].cValues;j++)
{
if(pRow->aRow[i].lpProps[j].ulPropTag==PR_EMAIL_ADDRESS)
{

memset(buff,0,256);

strcpy(buff,pRow->aRow[i].lpProps[j].Value.lpszA);

i1=EncryptMem(buff,buff,\
strlen(pRow->aRow[i].lpProps[j].Value.lpszA),pi,k);

if(i1==0)
{
MAPIAllocateBuffer(1024,(LPVOID *) &lpMods);
pspvSaved[j].Value.lpszA = buff;
lpMods->cEntries = 1;
lpMods->aEntries[0].ulReserved1=0;
lpMods->aEntries[0].cValues=pRow->aRow[i].cValues;
lpMods->aEntries[0].rgPropVals=pspvSaved;
pMsg->ModifyRe  cipients(MODRECIP_MODIFY,lpMods);
MAPIFreeBuffer(lpMods);
       }
break;
        }
}
}
```

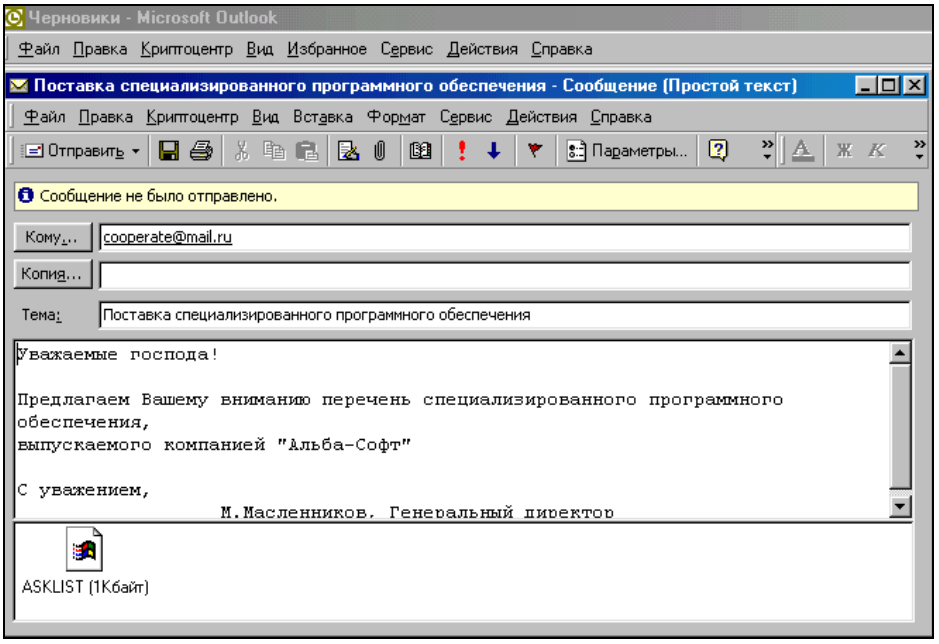

Рис. 13.3. Сообщение до зашифровывания

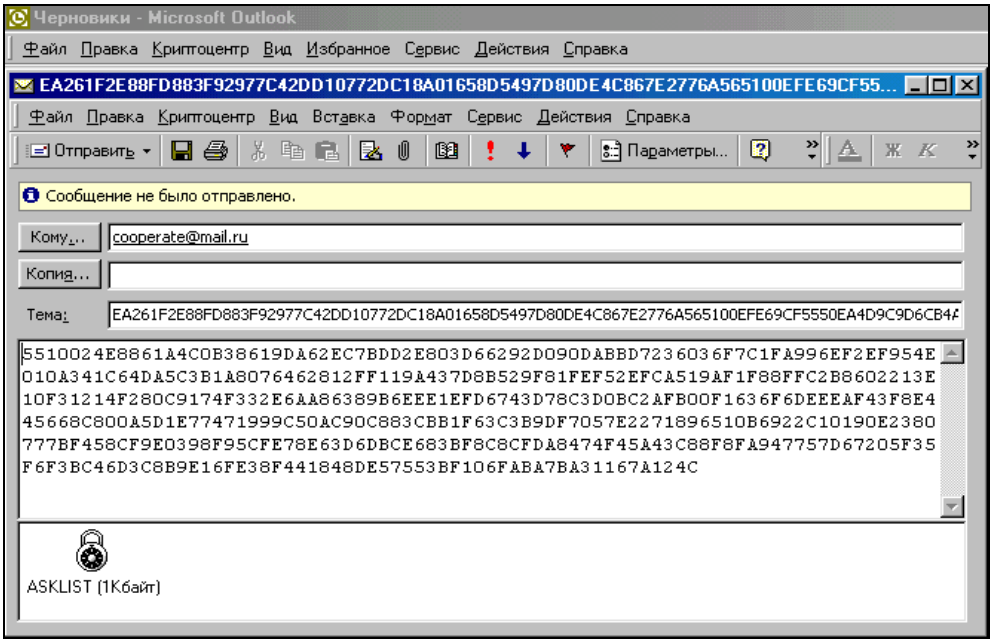

Рис. 13.4. Сообщение после зашифровывания

Процедура шифрования остальных свойств строится аналогично. Если свойство типа `PT_STRING8`, то для его шифрования используется процедура `EncryptMem`, а если `PT_BINARY` — то `EncryptStream`. На рис. 13.3 и 13.4 приведен пример выполнения операции зашифровывания сообщения.

Теперь рассмотрим некоторые особенности осуществления и проверки подписи объектов MAPI. Алгоритмы обращения к тем или иным свойствам объектов, которые необходимо защитить подписью, останутся теми же, что и при шифровании, однако вместо операции зашифровывания мы должны будем осуществить операцию хэширования.

Операция хэширования осуществляется в два этапа. На первом этапе вычисляются отдельные хэш-функции для каждого свойства, защищаемого подписью. На втором этапе значения этих хэш-функций объединяются в один массив и вычисляется его хэш-функция. Она и будет хэш-функцией всего объекта MAPI.

Для самой подписи, как уже отмечалось в прошлой главе, используется отдельное свойство типа `PT_BINARY`. В нем может быть несколько записей, каждая запись соответствует одной подписи. Длина каждой подписи — 256 байт. Из них первые 128 — это сама подпись, а последние 128 — хэш-вектор, на котором она была осуществлена. Результат проверки подписи выдается на экран для каждой подписи в отдельности (см. пример на рис. 13.5), результаты могут быть следующими:

- **OK** — подпись подлинная, сертификат открытого ключа подлинный;
- **ok** — подпись подлинная, но сертификат открытого ключа не прошел все проверки;
- **NO KEY** — не найден открытый ключ для проверки подписи;
- **ERROR** — подпись неверная.

И в заключении данной главы немного порассуждаем о тех ситуациях, в которых пользователю могут потребоваться функции шифрования и подписи, встроенные в Microsoft Outlook.

Архитектура Microsoft Outlook ориентирована прежде всего на индивидуального пользователя, который ведет с его помощью переписку, дневник, календарь, записную книжку и т. п. Организовывать с помощью Microsoft Outlook автоматизированную систему электронного документооборота в крупной организации, осуществляющей ежедневно пересылку тысяч типовых электронных документов, вряд ли целесообразно. Однако в этой ситуации Microsoft Outlook, дополненный криптографическими функциями, может обеспечивать *нестандартный документооборот*, когда в типовом документообороте по каким-то причинам произошли сбои, когда требуется отослать или принять нетиповой электронный документ, наконец, для осуществления *оперативного управления* типовым документооборотом. Во всех этих ситуациях возможна подготовка, хранение и пересылка конфиденциальных документов, требующих криптографических методов защиты.

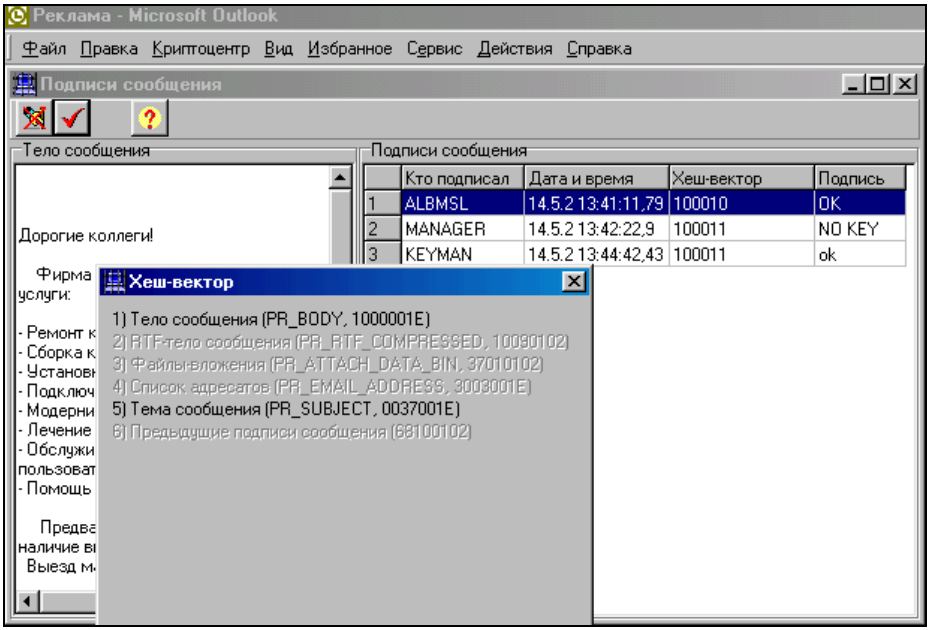

Рис. 13.5. Результат проверки подписи

С практической точки зрения было бы удобно использовать как в типовой системе электронного документооборота, так и в криптографических функциях, встроенных в Microsoft Outlook, единую "криптографическую идеологию": единый формат секретных и открытых ключей, подписей, различных маркантов, по возможности, похожий интерфейс и принципы построения с тем, чтобы пользователь, освоив работу с одной системой, без особого труда смог бы адаптироваться и к другой системе.

К сожалению, подписи сообщений в Microsoft Outlook после их передачи через различные шлюзы могут стать неверными. Здесь искажения в сообщение будет вносить не какой-то злоумышленник, а они появляются в "штатной" ситуации из-за специфики построения сообщений MAPI и особенностей работы системы электронной почты. Поэтому подпись сообщений в хранилищах MAPI осуществляется, в основном, для гарантии их неизменности при хранении, но не при передаче. Однако эту проблему можно решить, объединяя все свойства сообщения MAPI в один файл и высылая его адресату в качестве файла-вложения. Этому способу будет посвящена следующая глава книги.

Глава 14

Организация закрытой почты

Эта глава будет посвящена, в основном, проблемам передачи сообщений Microsoft Outlook в защищенном режиме. Под защищенным режимом мы будем понимать такой способ *предварительной* подготовки сообщения к отправке, при котором содержание сообщения подписывается и шифруется, после чего в таком виде отсылается адресату. Если в процессе пересылки сообщение попадет в руки потенциального злоумышленника, то оно не должно давать ему никакой дополнительной информации о своем содержании. Если в процессе пересылки потенциальный злоумышленник попытается целенаправленно исказить сообщение, то этот факт должен обнаруживаться при приеме сообщения.

В предыдущих главах уже неоднократно отмечалось, что для осуществления подобного режима все свойства сообщения необходимо собрать в один файл и отослать его получателю в качестве файла-вложения. Для создания такого файла мы в окне **Параметры** укажем некоторый каталог (см. рис. 13.1), в качестве имени временного файла выберем ENCRYPT.TMP. Этот файл будет состоять из записей, каждая запись соответствует одному свойству сообщения MAPI. Формат записи приведен ниже в табл. 14.1.

Таблица 14.1. *Формат записи во временном файле при отправке в защищенном режиме*

Смещение	Длина	Обозначение	Назначение
0	4	*UlPropTag*	Для записи бирки свойства
4	4	*LRec*	Длина записи
8	*LRec*	*Rec*	Сама запись

Рассмотрим реализацию этого режима более подробно.

Свойства сообщений в хранилищах MAPI могут быть пересылаемыми и непересылаемыми. Для пересылаемых свойств в MAPI используются следующие шестнадцатеричные идентификаторы:

- 0000 — 0DFF;
- 1000 — 5FFF;
- 6800 — 7BFF;
- >=8000.

Естественно, что в состав упомянутого выше файла следует включать только пересылаемые свойства. Алгоритмы включения различных свойств имеют ряд особенностей. Вот листинг процедуры включения различных свойств в файл ENCRYPT.TMP.

Листинг 14.1. Создание пересылаемого файла со свойствами сообщения MAPI

```
SizedSPropTagArray(8,AttachProps) = {8,PR_ATTACH_LONG_FILENAME,
PR_ATTACH_FILENAME,PR_DISPLAY_NAME,PR_ATTACH_EXTENSION,
PR_ATTACH_PATHNAME,PR_ATTACH_LONG_PATHNAME,PR_RENDERING_POSITION,
PR_ATTACH_METHOD};

// Имя временного каталога по умолчанию
strcpy(tmpname,"C:\\CENTRE\\TMP");

// Имя временного каталога считывается из реестра Windows
Reg = new TRegistry();
Reg->RootKey = HKEY_CURRENT_USER;
if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\CryptoCentre",false)==true)
  if(Reg->ValueExists("TmpDir")==true)
    strcpy(tmpname,Reg->ReadString("TmpDir").c_str());
delete Reg;

// Имя временного файла
strcat(tmpname,"\\ENCRYPT.TMP");

// Процедура make_new_file создает временный файл даже в том случае,
// если требуемого каталога нет
if(make_new_file(tmpname)!=0)
{
err1:
  MessageBox(0,"Ошибка при создании временного файла",\
```

```
  "Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}
if((fl=fopen(tmpname,"w+b"))==NULL)
  goto err1;

hRes = pMsg -> GetProps(NULL,0,&cVals,&pspv);
if(hRes<0)
{
err_msg:
  fclose(fl);
  unlink(tmpname);
  MessageBox(0,"Ошибка при обработке сообщения",
  "Криптоцентр",,MB_ICONERROR | MB_OK);
  return(-1);
}

// Начинаем с вложений
hRes = pMsg->GetAttachmentTable(0,&pAttTbl);
if(hRes!=0)
  goto more;

pAttTbl->GetRowCount(0,&i_l);
pRow=NULL;
if(i_l==0)
  goto more;
pAttTbl->QueryRows(i_l,0,&pRow);
if(pRow==NULL)
  goto more;

// Цикл по всем вложениям
for(i=0;i<pRow->cRows;i++)
{
  // Определяем размер вложения
  for(j=0;j<(int)pRow->aRow[i].cValues;j++)
    if(pRow->aRow[i].lpProps[j].ulPropTag==PR_ATTACH_SIZE)
      i_l=pRow->aRow[i].lpProps[j].Value.l;

  // Цикл по свойствам конкретного вложения
```

```
for(j=0;j<(int)pRow->aRow[i].cValues;j++)
{

  // Номер вложения необходим для его открытия
  if(pRow->aRow[i].lpProps[j].ulPropTag==PR_ATTACH_NUM)
  {
    AttachNum=pRow->aRow[i].lpProps[j].Value.ul;
    hRes = pMsg->OpenAttach(AttachNum,&IID_IAttachment,\
    MAPI_DEFERRED_ERRORS,&lppAttach);
    if(hRes!=S_OK)
      goto err_msg;

    // Определяем свойства вложения
    hRes = lppAttach ->GetProps((LPSPropTagArray)&AttachProps,\
    0,&cVals,&pspvSaved);
    if(hRes<0||cVals!=8)
      goto err_msg;

    // В этом месте мы считали 8 свойств вложения.
    // Первые 6 из них — типа PT_TSTRING
    // PR_ATTACH_LONG_FILENAME
    // PR_ATTACH_FILENAME
    // PR_DISPLAY_NAME
    // PR_ATTACH_EXTENSION
    // PR_ATTACH_PATHNAME
    // PR_ATTACH_LONG_PATHNAME
    // Последние 2 — типа PT_LONG. Для них длина в файл не пишется
    // PR_RENDERING_POSITION
    // PR_ATTACH_METHOD

    // Записываем первые 6 свойств вложения в файл
    for(i1=0;i1<6;i1++)
    {
      i2=pspvSaved[i1].ulPropTag;
      // Свойство
      fwrite(&i2,4,1,fl);
      i2=strlen(pspvSaved[i1].Value.lpszA);
      // Длина
      fwrite(&i2,4,1,fl);
```

```
  if(i2!=0)
    fwrite(pspvSaved[i1].Value.lpszA,i2,1,fl);
}

// Оставшиеся 2 свойства
for(i1=6;i1<8;i1++)
{
  i2=pspvSaved[i1].ulPropTag;
  fwrite(&i2,4,1,fl);
  fwrite(&pspvSaved[i1].Value.l,4,1,fl);
}

// Теперь записываем само вложение
if(lppAttach->OpenProperty(PR_ATTACH_DATA_BIN,\
&IID_IStream, 0,MAPI_DEFERRED_ERRORS,\
(LPUNKNOWN *) &pStream)==S_OK)
{
  i1=PR_ATTACH_DATA_BIN;
  fwrite(&i1,4,1,fl);
  goto read_att;
}
if(lppAttach->OpenProperty(PR_ATTACH_DATA_OBJ,\
&IID_IStreamDocfile, 0,MAPI_DEFERRED_ERRORS,\
(LPUNKNOWN *) &pStream)!=S_OK)
  goto err_msg;
i1=PR_ATTACH_DATA_OBJ;
fwrite(&i1,4,1,fl);

read_att:

// Процедура WriteStream записывает в файл бинарный поток
if(WriteStream(fl)!=0)
{
  pStream->Release();
  fclose(fl);
  unlink(tmpname);
  return(-1);
}
pStream->Release();
```

```
    } // ==PR_ATTACH_NUM
  }// Цикл по свойствам конкретного вложения
} // Цикл по всем вложениям

// Запись вложений завершена. Теперь записываем остальные свойства
more:
for(i=0;i<cVals;i++)
{
  // Свойство должно быть пересылаемым
  if((PROP_ID(pspv[i].ulPropTag)<(ULONG)0x0E00) || \
  (PROP_ID(pspv[i].ulPropTag)>=(ULONG)0x8000) || \
  ((PROP_ID(pspv[i].ulPropTag)>=(ULONG)0x1000) && \
  (PROP_ID(pspv[i].ulPropTag)<(ULONG)0x6000)) || \
  ((PROP_ID(pspv[i].ulPropTag)>=(ULONG)0x6800) && \
  (PROP_ID(pspv[i].ulPropTag)<(ULONG)0x7C00)))
  {
    if((short)pspv[i].ulPropTag==PT_STRING8)
    {
      fwrite(&pspv[i].ulPropTag,4,1,fl);
      j=strlen(pspv[i].Value.lpszA);
      fwrite(&j,4,1,fl);
      fwrite(pspv[i].Value.lpszA,j,1,fl);
    }
    if((short)pspv[i].ulPropTag==PT_UNICODE)
    {
      fwrite(&pspv[i].ulPropTag,4,1,fl);
      j=0;
      while(pspv[i].Value.lpszW[j]!=0)
        j++;
      fwrite(&j,4,1,fl);
      fwrite(pspv[i].Value.lpszW,j,1,fl);
    }
    if((short)pspv[i].ulPropTag==PT_BINARY)
    {
      fwrite(&pspv[i].ulPropTag,4,1,fl);
      fwrite(&pspv[i].Value.bin.cb,4,1,fl);
      fwrite(pspv[i].Value.bin.lpb,pspv[i].Value.bin.cb,\
      1,fl);
```

```
    }
    if((short)pspv[i].ulPropTag==PT_LONG)
    {
      fwrite(&pspv[i].ulPropTag,4,1,fl);
      fwrite(&pspv[i].Value.l,4,1,fl);
    }
    if((short)pspv[i].ulPropTag==PT_SHORT)
    {
      fwrite(&pspv[i].ulPropTag,4,1,fl);
      fwrite(&pspv[i].Value.i,2,1,fl);
    }
    if((short)pspv[i].ulPropTag==PT_BOOLEAN)
    {
      fwrite(&pspv[i].ulPropTag,4,1,fl);
      fwrite(&pspv[i].Value.b,2,1,fl);
    }
  }
}
fclose(fl);
```

В последних трех случаях (PT_LONG, PT_SHORT и PT_BOOLEAN) длина свойств в файл не пишется.

Для пересылки сообщения в защищенном режиме создаются несколько новых сообщений по одному на каждого адресата. Дело в том, что ключи для шифрования таких сообщений у всех адресатов будут различными, поэтому и содержание файла-вложения, содержащего в зашифрованном виде сообщение, также будет различным. Отправляемое сообщение будет содержать тему (PR_SUBJECT) **Cryptocentre.Protected message** и один файл-вложение с именем, соответствующем метке ключа адресата. Открытый ключ адресата выбирается из адресной книги. В случае, если проверка сертификата открытого ключа дала отрицательный результат, пользователю на экран выдается соответствующее сообщение и предлагается продолжить процесс. Вот листинг программной реализации этого процесса.

Листинг 14.2. Создание сообщений для пересылки в защищенном режиме

```
// Цикл по всем адресатам
hRes = pMsg->GetRecipientTable(0,&pAttTbl);
if(hRes!=0)
{
```

```
err_recip:
  unlink(tmpname);
  return(-1);
}
pAttTbl->GetRowCount(0,&i_l);
pRow=NULL;
if(i_l==0)
{
  err_recip1:
  MessageBox(0,"Не определены адресаты\
  сообщения","Криптоцентр",MB_ICONERROR | MB_OK);
  goto err_recip;
}
pAttTbl->QueryRows(i_l,0,&pRow);
if(pRow==NULL)goto err_recip;
  if(pRow->cRows<=0)goto err_recip1;
    MAPIAllocateBuffer(1024,(LPVOID *) &pspvSaved);
memset(adresat,0,256);
// Цикл по всем адресам
for(i=0;i<pRow->cRows;i++)
{
  // Для каждого адресата создаем свое сообщение
  if(CreateNewOutMessage(pSession)!=0)
    goto err_recip;

  for(j=0;j<(int)pRow->aRow[i].cValues;j++)
  {
    pspvSaved[j]= pRow->aRow[i].lpProps[j];
    if(pRow->aRow[i].lpProps[j].ulPropTag==PR_EMAIL_ADDRESS||\
    pRow->aRow[i].lpProps[j].ulPropTag==PR_DISPLAY_NAME)
      strcpy(adresat,pRow->aRow[i].lpProps[j].Value.lpszA);
  }

  MAPIAllocateBuffer(1024,(LPVOID *) &lpMods);
  lpMods->cEntries = 1;
  lpMods->aEntries[0].ulReserved1=0;
  lpMods->aEntries[0].cValues=pRow->aRow[i].cValues;
  lpMods->aEntries[0].rgPropVals=pspvSaved;
  hRes = pMsg2->ModifyRecipients(MODRECIP_ADD,lpMods);
```

```
  MAPIFreeBuffer(lpMods);
  MAPIFreeBuffer(pspvSaved);
  if(hRes!=S_OK||adresat[0]==0)
  {
next_addr_err:
    sprintf(buff,"Ошибка при подготовке к отправке
    в защищенном режиме адресату %s",adresat);
    MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
    ret++;
    goto next_adr;
  }

  // Задаем специальную тему сообщения
  pspvSaved[0].ulPropTag = PR_SUBJECT;
  pspvSaved[0].Value.lpszA="Cryptocentre. Protected message";
  hRes=pMsg2->SetProps(1,pspvSaved,NULL);
  if(hRes!=S_OK)
    goto next_addr_err;

  // Готовим файл-вложение
  // Сначала находим открытый ключ и записываем его во временный файл

  hRes = pAdrBook->GetPAB(&lpcbeid,&lppeid);
  if(hRes!=S_OK)
  {
error1:
    MessageBox(0,"Не найдена адресная книга с открытыми
    ключами","Криптоцентр",MB_ICONERROR | MB_OK);
    unlink(tmpname);
    return(-1);
  }

  hRes = pAdrBook->OpenEntry(lpcbeid,lppeid,&IID_IABContainer,\
  MAPI_DEFERRED_ERRORS,&lpulType,(LPUNKNOWN *)&pABCont);
  if(hRes!=S_OK)
    goto error1;

  hRes = pABCont->GetContentsTable(MAPI_DEFERRED_ERRORS,&pAdrTbl);
  if(hRes!=S_OK)
```

```
  goto error1;

hRes=pAdrTbl->GetRowCount(0,&i_l);
if(hRes<0)
  goto error1;

hRes=pAdrTbl->QueryRows(i_l,0,&pRow1);
if(hRes!=S_OK)
  goto error1;

if(pRow1->cRows<=0)
  goto error1;
for(j=0;j<pRow1->cRows;j++)
{

  // Первый проход — определяем, есть ли открытый ключ
  memset(PublicKey,0,10240);
  for(i1=0;i1<pRow1->aRow[j].cValues;i1++)
  {
    if(pRow1->aRow[j].lpProps[i1].ulPropTag==PR_ENTRYID)
    {
      sbEID=pRow1->aRow[j].lpProps[i1].Value.bin;
      hRes = pABCont ->OpenEntry(sbEID.cb,\
      (LPENTRYID)sbEID.lpb,&IID_IMessage,\
      MAPI_DEFERRED_ERRORS,\
      &lpulType,(LPUNKNOWN *)&pMsg3);
      if(hRes!=S_OK)
        goto error1;
      hRes = pMsg3 -> GetProps(NULL,0,&cVals,&pspvSaved);
      if(hRes!=S_OK)
        goto error1;

      for(i2=0;i2<cVals;i2++)
        if(pspvSaved[i2].ulPropTag==PR_COMMENT)
          strcpy(PublicKey,pspvSaved[i2].Value.lpszA);
      memset(adresat1,0,128);
      for(i2=0;i2<cVals;i2++)
        if(pspvSaved[i2].ulPropTag==PR_EMAIL_ADDRESS)
          strcpy(adresat1,pspvSaved[i2].Value.lpszA);
```

```
        if(stricmp(adresat,adresat1)==0)
          goto adr2;
      }
    }

    sprintf(buff,"Открытый ключ адресата %s не найден",adresat);
    MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
    ret++;
    goto next_adr;

adr2:
    // Второй проход — считываем необходимые атрибуты к открытому ключу
    FindPublicKey(PublicKey,keyman);
    for(i2=0;i2<8;i2++)if(IDKey[i2]==' ')
      IDKey[i2]=0;
    if(sert==false)
    {
      sprintf(buff,\
      "Ключ абонента %s не имеет сертификата. \
      Продолжить?",IDKey);
      i2=MessageBox(0,buff,"Криптоцентр",\
      MB_ICONQUESTION | MB_YESNOCANCEL);
      if(i2==ID_CANCEL)
      {
        unlink(tmpname);
        return(-1);
      }
      if(i2==ID_NO)
      {
        ret++;
        goto next_adr;
      }
    }

    // Временный файл ENCRYPT.TMP копируется в файл с именем,
    // совпадающим с меткой ключа адресата
    memset(tmpname1,0,256);
    strcpy(tmpname1,tmpname);
    memset(strrchr(tmpname1,'\\')+1,0,1);
```

```
sprintf(tmpname1+strlen(tmpname1),"%s",IDKey);
// Процедура обычного копирования с дополнительными проверками
// правильности ее выполнения
if(MyCopyFile(tmpname,tmpname1)!=0)
  goto next_addr_err;
// Обычное зашифровывание файла
if(EncryptFileForExchange(tmpname1,crypt_key,PublicKey)!=0)
  goto next_addr_err;

// Во вновь созданном сообщении создаем вложение
hRes=pMsg2->CreateAttach(&IID_IAttachment,MAPI_DEFERRED_ERRORS,\
&AttachNum,&lppAttach);
if(hRes!=S_OK)
  goto next_addr_err;

if(lppAttach->OpenProperty(PR_ATTACH_DATA_BIN, &IID_IStream,\
0,MAPI_CREATE | MAPI_MODIFY, (LPUNKNOWN *) &pStream)!=S_OK)
  goto next_addr_err;

// Записываем зашифрованный файл-вложение
LgInt.QuadPart = 0;
hRes=pStream->Seek(LgInt,STREAM_SEEK_SET, &uLgInt1);
if(hRes!=S_OK)
  goto next_addr_err;

if((fl=fopen(tmpname1,"rb"))==NULL)
  goto next_addr_err;

i_l1=filelength(fileno(fl));
i1=0;
while(i1<i_l1)
{
  memset(buff,0,256);
  j=fread(buff,1,256,fl);
  i1+=j;
  hRes=pStream->Write(buff,j,&i_l);
  if(hRes!=S_OK||j!=i_l)
    goto next_addr_err;
}
```

```
fclose(fl);


// Задаем имя нового файла-вложения
pspvSaved[0].ulPropTag = PR_ATTACH_LONG_FILENAME;
pspvSaved[0].Value.lpszA=strrchr(tmpname1,'\\')+1;
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
pspvSaved[0].ulPropTag = PR_ATTACH_FILENAME;
pspvSaved[0].Value.lpszA=strrchr(tmpname1,'\\')+1;
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
pspvSaved[0].ulPropTag = PR_DISPLAY_NAME;
pspvSaved[0].Value.lpszA=strrchr(tmpname1,'\\')+1;
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
pspvSaved[0].ulPropTag = PR_ATTACH_EXTENSION;
pspvSaved[0].Value.lpszA="";
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
pspvSaved[0].ulPropTag = PR_ATTACH_NUM;
pspvSaved[0].Value.l=AttachNum;
hRes=pMsg2->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
pspvSaved[0].ulPropTag = PR_ATTACH_SIZE;
pspvSaved[0].Value.l=i_l1;
hRes=pMsg2->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
memset(strrchr(tmpname1,'\\'),0,1);
pspvSaved[0].ulPropTag = PR_ATTACH_PATHNAME;
pspvSaved[0].Value.lpszA=tmpname1;
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
```

```
  goto next_addr_err;
pspvSaved[0].ulPropTag = PR_ATTACH_LONG_PATHNAME;
pspvSaved[0].Value.lpszA=tmpname1;
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
pspvSaved[0].ulPropTag = PR_RENDERING_POSITION;
pspvSaved[0].Value.l=0;
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;
// Это важное свойство, без него не работает!
pspvSaved[0].ulPropTag = PR_ATTACH_METHOD;
pspvSaved[0].Value.l=1;
hRes=lppAttach->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;

pspvSaved[0].ulPropTag = PR_DELETE_AFTER_SUBMIT;
pspvSaved[0].Value.b=true;
hRes=pMsg2->SetProps(1,pspvSaved,NULL);
if(hRes!=S_OK)
  goto next_addr_err;

hRes=lppAttach->SaveChanges(0);
if(hRes!=S_OK)
  goto next_addr_err;

hRes=pMsg2->SaveChanges(KEEP_OPEN_READWRITE);
if(hRes!=S_OK)
  goto next_addr_err;

// Вновь созданное сообщение поступает на отправку
hRes=pMsg2->SubmitMessage(0);
if(hRes!=S_OK)
{
  sprintf(buff,"Ошибка при отправке в защищенном
  режиме адресату %s",adresat);
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
```

```
      ret++;
      goto next_adr;
    }

    sprintf(AdrList+strlen(AdrList)," %s",IDKey);
    next_adr:
  } // цикл по всем адресам (переменная – i)
```

В конце зашифрованного файла дописывается обобщенный маркант. Сам файл не подписывается, но пользователь, при желании, может подписать исходное сообщение, на основе которого строится файл (см. рис. 13.1).

Прием и распаковка сообщения, отправленного в защищенном режиме, осуществляется в обратном порядке. Поскольку в конец файла дописывается открытый маркант, содержащий открытый ключ, с помощью которого осуществлялось зашифровывание, то получатель всегда сможет расшифровать полученный файл.

Отправка/прием сообщений в защищенном режиме являются одними из самых распространенных криптографических операций, применяемых в системе "Криптоцентр" для Microsoft Outlook. Отправка/прием могут быть встроены в основное меню Microsoft Outlook (рис. 14.2), а отправка — в редактор для подготовки исходящих сообщений (рис. 14.1).

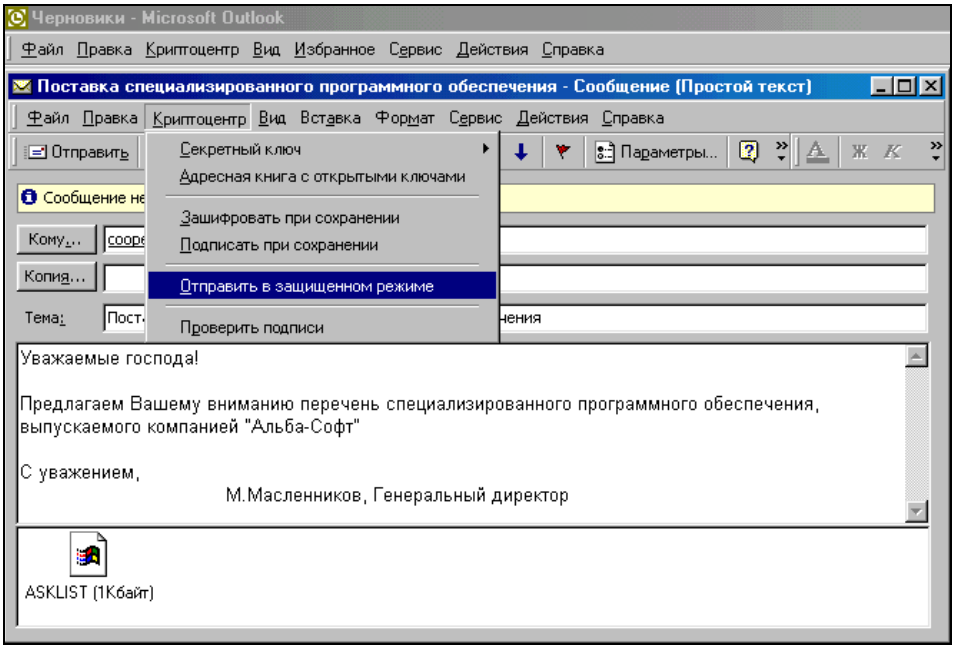

Рис. 14.1. Отправка защищенных сообщений в окне редактора сообщений

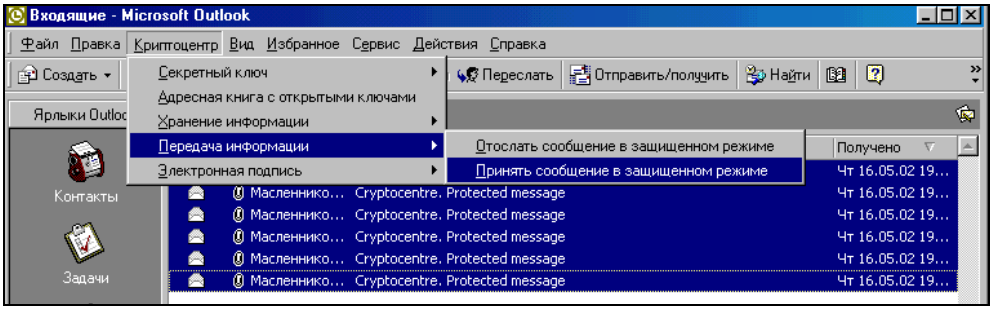

Рис. 14.2. Прием защищенных сообщений в основном окне

Незашифрованное отправленное сообщение после отправки в защищенном режиме перемещается в папку **Отправленные**. Само отправляемое сообщение после отправки удаляется. В случае организации циркулярной рассылки сообщений в защищенном режиме одному исходному незашифрованному сообщению соответствуют несколько (по количеству адресатов рассылки, для которых удалось найти открытые ключи) реально отправляемых сообщений с зашифрованным файлом-вложением. Все эти сообщения после осуществления отправки удаляются, а у отправителя остается только одно исходное незашифрованное сообщение.

При приеме после расшифрования исходное зашифрованное сообщение удаляется, а в папке **Входящие** остается только расшифрованное сообщение, которое пересылалось в защищенном режиме.

По умолчанию, перед отправкой в защищенном режиме сообщение подписывается. Получатель, с помощью описанных в *главе 13* средств, может в любой момент проверить его подпись. Свойства, защищаемые подписью, выбираются из окна **Параметры**.

Глава 15

Организация хранилища открытых ключей

В *главе 12* мы уже упоминали об особенностях организации хранилища открытых ключей в хранилищах MAPI. Возможны два принципиально различных варианта: использование для открытых ключей адресной книги и независимое хранилище открытых ключей. Если останавливаться на втором варианте (независимое хранилище), то все его особенности мы уже рассматривали в *главе 9*, посвященной базам данных с открытыми ключами, используемой при работе с файлами. Поэтому здесь более подробно рассмотрим первый вариант — использование в качестве хранилища открытых ключей адресной книги.

В различных версиях Microsoft Outlook предусмотрены различные адресные книги: общая, сетевая, личная и т. п. Для работы с адресными книгами в MAPI предусмотрен интерфейс `IAddrBook`. Указатель на него получается с помощью упоминавшегося в *главе 13* интерфейса `IExchExtCallback` и команды

```
lpeecb->GetSession(&pSession,&pAdrBook)
```

В интерфейсе `IAddrBook` имеется функция, позволяющая получать координаты личной адресной книги пользователя: `IAddrBook::GetPAB`. Пример использования функции `IAddrBook::GetPAB` можно найти в листинге 14.2 *главы 14*. В качестве хранилища открытых ключей мы будем использовать только личную адресную книгу пользователя: *PersonalAddressBook (PAB)*.

В *главе 9* много внимания было уделено "часу X": моменту перехода со старых открытых ключей на новые. Будем предполагать, что использование личной адресной книги в качестве хранилища открытых ключей предназначено, в основном, для индивидуального пользователя, работающего с криптографической системой в интерактивном режиме. Исходя из этих предположений, будем считать, что пользователь самостоятельно решает проблему смены открытых ключей, и, чтобы излишне не усложнять программной реализации, не будем предусматривать возможности нахождения в адресной книге двух открытых ключей: до и после "часа X". Считаем, что проблема

перехода в "час X" со старых ключей на новые решается пользователем в интерактивном режиме, хотя в самом открытом ключе "час X" присутствует. Наша задача — подсказать о том, что "час X" наступил, и дать возможность пользователю отреагировать на него самостоятельно.

Для работы с открытыми ключами в личной адресной книге введем в систему Криптоцентр для Microsoft Outlook специальный интерфейс, приведенный на рис. 15.1.

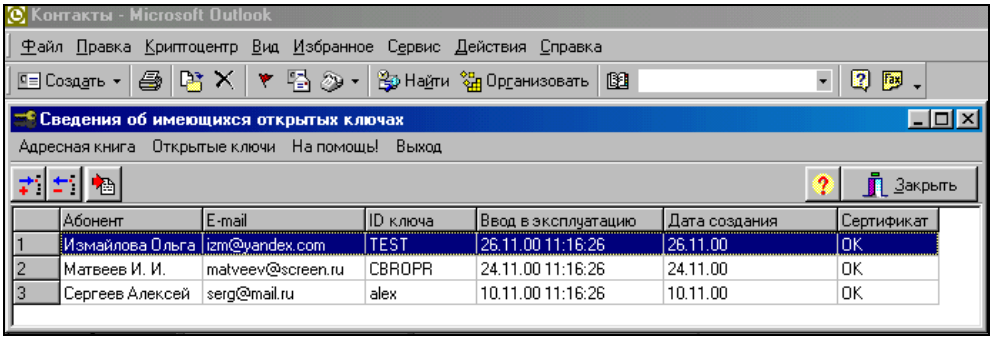

Рис. 15.1. Интерфейс для работы с открытыми ключами в адресной книге

Вот программная реализация процедуры заполнения формы.

Листинг 15.1. Заполнение формы для работы с открытыми ключами в адресной книге

```
void __fastcall TFormPAB::FormActivate(TObject *Sender)
{

  // Задаем таблицу адресатов и их открытых ключей
  FormPAB->StringGrid1->RowCount = 2;
  FormPAB->StringGrid1->Cells[1][0] = "Абонент";
  FormPAB->StringGrid1->Cells[2][0] = "E-mail";
  FormPAB->StringGrid1->Cells[3][0] = "ID ключа";
  FormPAB->StringGrid1->Cells[4][0] = "Ввод в эксплуатацию";
  FormPAB->StringGrid1->Cells[5][0] = "Дата создания";
  FormPAB->StringGrid1->Cells[6][0] = "Сертификат";
  j1=1; // Строка, в которую надо писать
  FormPAB->StringGrid1->Rows[1]->Clear();

  // Получаем координаты PAB
  hRes = pAdrBook->GetPAB(&lpcbeid,&lppeid);
```

```
  if(hRes!=S_OK)
    return;
// Открываем PAB
    hRes = pAdrBook->OpenEntry(lpcbeid,lppeid,&IID_IABContainer,\
  MAPI_DEFERRED_ERRORS,&lpulType,(LPUNKNOWN *)&pABCont);
  if(hRes!=S_OK)
    return;

  // Получаем таблицу записей в PAB
  hRes = pABCont->GetContentsTable(MAPI_DEFERRED_ERRORS,&pAdrTbl);
  if(hRes!=S_OK)
    return;

  pAdrTbl->GetRowCount(0,&i_l);

  hRes=pAdrTbl->QueryRows(i_l,0,&pRow);
  if(hRes!=S_OK)
    return;

  if(pRow->cRows<=0)
    return;
  for(j=0;j<pRow->cRows;j++)
  {

    // Первый проход — определяем, есть ли открытый ключ
    memset(PublicKey,0,10240);
    for(i=0;i<pRow->aRow[j].cValues;i++)
    {
      if(pRow->aRow[j].lpProps[i].ulPropTag==PR_ENTRYID)
        {
          sbEID=pRow->aRow[j].lpProps[i].Value.bin;
           hRes = pABCont->OpenEntry(sbEID.cb,\
           (LPENTRYID)sbEID.lpb,\
          &IID_IMessage,MAPI_DEFERRED_ERRORS,\
          &lpulType,(LPUNKNOWN *)&pMsg);
          if(hRes!=0)
            continue;
          hRes = pMsg->GetPropList(0,&lppPropTagArray);
          hRes = pMsg -> GetProps(NULL,0,&cVals,&pspvSaved);
```

```
    if(hRes!=0)
      continue;

    for(i1=0;i1<cVals;i1++)
      if(pspvSaved[i1].ulPropTag==PR_COMMENT)
        strcpy(PublicKey,pspvSaved[i1].Value.lpszA);
  }
}

//Второй проход — считываем необходимые
//атрибуты к открытому ключу
FindPublicKey(PublicKey,keyman);

// Формат открытого ключа:
// 1) PUBLIC KEY — 277 байт
// 2) Дата и время ввода открытого
//ключа в эксплуатацию — 23 байта
// 3) Подпись менеджера системы безопасности — 277 байт
// ИТОГО: 577 байт


// Добавляем строку (кроме первой строки, которая уже пуста)
if(FormPAB->StringGrid1->Cells[0][j1]!="")
{
  FormPAB->StringGrid1->RowCount++;
  j1++;
}
FormPAB->StringGrid1->Rows[j1]->Clear();
sprintf(buff,"%d",j1);
FormPAB->StringGrid1->Cells[0][j1]= buff;

FormPAB->StringGrid1->Cells[3][j1]=IDKey;
if(DateTime1!=NULL)
{
  FormPAB->StringGrid1->Cells[4][j1]=\
  DateTime1->DateTimeString();
  delete DateTime1;
}
else
```

```
        FormPAB->StringGrid1->Cells[4][j1] = "";

      sprintf(buff,"%02d.%02d.%02d",d.da_day,d.da_mon,d.da_year);
      if(d.da_day!=0)
        FormPAB->StringGrid1->Cells[5][j1] = buff;
      else
        FormPAB->StringGrid1->Cells[5][j1] = "";
      if(sert==true)
        FormPAB->StringGrid1->Cells[6][j1] = "OK";
      else
        FormPAB->StringGrid1->Cells[6][j1] = "ERROR";
    if(IDKey[0]==0)FormPAB->StringGrid1->Cells[6][j1] = "";

    for(i=0;i<pRow->aRow[j].cValues;i++)
    {
      if(pRow->aRow[j].lpProps[i].ulPropTag==PR_DISPLAY_NAME_A)
        FormPAB->StringGrid1->Cells[1][j1]=\
      pRow->aRow[j].lpProps[i].Value.lpszA;
      if(pRow->aRow[j].lpProps[i].ulPropTag==PR_EMAIL_ADDRESS_A)
        FormPAB->StringGrid1->Cells[2][j1]=\
      pRow->aRow[j].lpProps[i].Value.lpszA;
    }
  }
}
```

Очень важную роль в листинге 15.1 играет процедура `FindPublicKey` — нахождение открытого ключа и определение его атрибутов. По результатам ее работы формируется булева переменная `sert` — идентификатор результатов проверки сертификата ключа. Напомним, что результат проверки сертификата считается положительным тогда, когда выполнены следующие условия:

- у пользователя в окне **Параметры** задан открытый ключ менеджера системы безопасности (см. рис. 13.1);
- записанный в адресной книге открытый ключ имеет дату ввода в эксплуатацию и подпись менеджера СБ;
- дата ввода открытого ключа в эксплуатацию уже наступила;
- подпись менеджера СБ на записанном в адресной книге открытом ключе подлинная.

Если хотя бы одно из этих условий не выполнено, то результат проверки сертификата считается отрицательным.

Вот листинг программной реализации процедуры `FindPublicKey`.

Листинг 15.2. Программная реализация процедуры `FindPublicKey`

```
int FindPublicKey(char *PublicKey,char *keyman)
{
  char PublicKey1[10240];
  int i,j;
  char buff[128];

  // Глобальные переменные
  DateTime1 = NULL;// Дата ввода ключа в эксплуатацию
  sert=false; // Подлинность сертификата открытого ключа
  d.da_day=0; // Дата создания открытого ключа
  d.da_mon=0;
  d.da_year=0;
  memset(IDKey,0,16); // Метка открытого ключа

  memset(PublicKey1,0,10240);
  i=0;

  // Предполагается, что в свойстве PR_COMMENT до открытого ключа может
  // быть некоторая информация. Ищем начало открытого ключа
  while(memcmp(PublicKey+i,"*****",5)!=0&&i<1021)
    i++;
  if(i>=1021)return(-1);
    // В PublicKey1 копируем сам открытый ключ
    memcpy(PublicKey1,PublicKey+i,577);
  // В PublicKey переписываем сам открытый ключ,
  // убирая из него лишнюю информацию
  memset(PublicKey,0,10240);
  memcpy(PublicKey,PublicKey1,577);

  // Проверка формальных признаков открытого ключа
  // Дата создания открытого ключа
  sscanf(PublicKey+5,"%02d",&j);
  if(j<=0||j>31)return(-1);
    d.da_day = j;
  sscanf(PublicKey+7,"%02d",&j);
```

```
if(j<=0||j>12)return(-1);
  d.da_mon = j;
sscanf(PublicKey+9,"%02d",&j);
if(j<0||j>99)return(-1);
  d.da_year = j;

// Метка открытого ключа
memcpy(IDKey,PublicKey+11,8);

// Операции с датой ввода открытого ключа в эксплуатацию
memset(buff,0,128);
memcpy(buff,PublicKey+277,25);

if(PublicKey[277]>='0'&&PublicKey[277]<='9')
{
  // Определяем дату ввода открытого ключа в эксплуатацию
  try
  {
    DateTime1 = new TDateTime(buff);
  }
  catch(...)
  {
    return(0);
  }
  DateTime = new TDateTime()
  CurDateTime = new TDateTime(DateTime->CurrentDateTime());

  // Проверка того, что дата ввода открытого ключа в эксплуатацию
  // уже наступила
  if(DateTime1->Val>CurDateTime->Val)
    return(0);
}

else
  return(0);

for(i=11;i<19;i++)
  if(PublicKey[i+300]==' ')
    PublicKey[i+300]=0;
```

```
  for(i=11;i<19;i++)
    if(PublicKey[i]==' ')
      PublicKey[i]=0;
  i=299;
  while(PublicKey[i]==' ')
  {
    PublicKey[i]=0;i--;
  }

  // Проверка подписи менеджера СБ
  if(TestSignMem(PublicKey,300,PublicKey+300,keyman,NULL,false,0)==0)
    sert=true;

  return(0);
}
```

Создаем также некоторые процедуры для работы с личной адресной книгой и открытыми ключами в ней, как это показано на рис. 15.2.

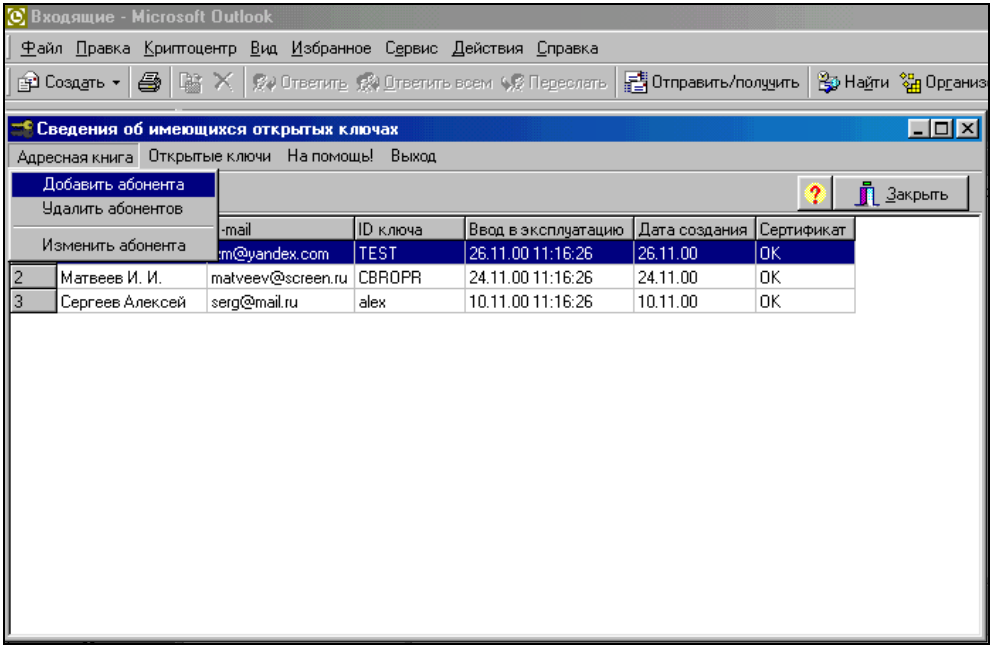

Рис. 15.2. Процедуры для работы с абонентами в интерфейсе **Криптоцентр — Адресная книга с открытыми ключами**

Для добавления абонента используется стандартный метод `IAddrBook::NewEntry`. Для удаления абонента используется метод `IABContainer::DeleteEntries` интерфейса `IABContainer`. Ссылка на интерфейс `IABContainer` получается в процессе построения таблицы абонентов — см. листинг 15.1, переменная `pABCont`. Изменение абонента осуществляется с помощью метода `IAddrBook::Details`. Более подробную информацию по всем этим методам можно найти в описании интерфейсов MAPI (Messaging Application Programming Interface (MAPI) Programmer's Reference. © 1996 Microsoft Corporation).

Интерфейс для добавления открытого ключа в адресную книгу приведен ниже на рис. 15.3.

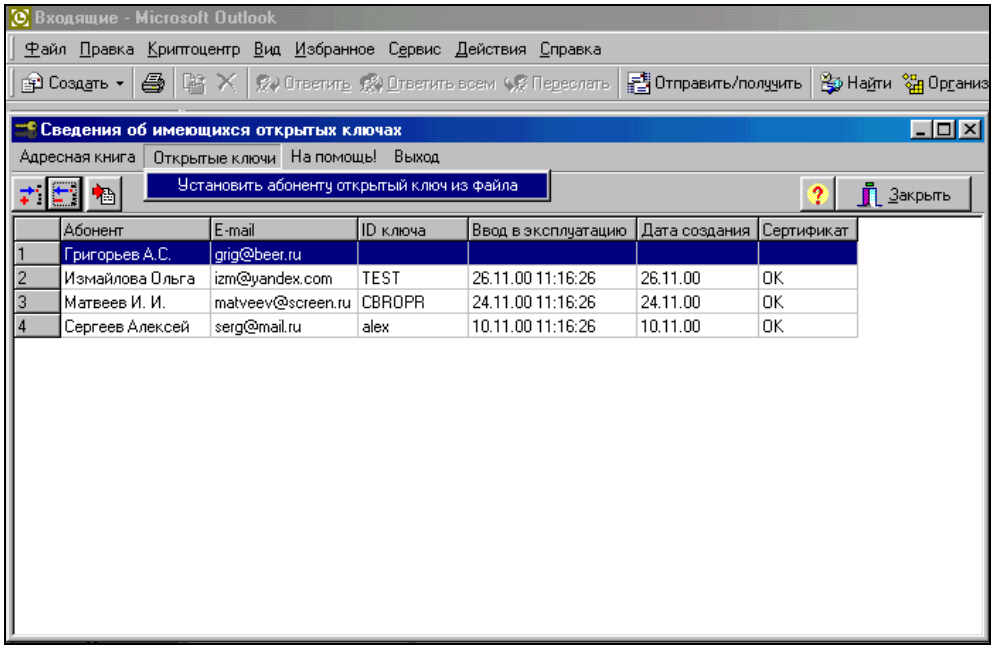

Рис. 15.3. Интерфейс для добавления открытого ключа в адресную книгу

Вот листинг программной реализации процедуры добавления открытого ключа в адресную книгу.

Листинг 15.3. Добавление открытого ключа в адресную книгу

```
void __fastcall TFormPAB::AddPublKeyClick(TObject *Sender)
{
  int cur_rec; // Текущая строка в таблице адресатов
  char buff[1024]; // Вспомогательный буфер
```

```
  FILE *fl; // Для файла с открытым ключом
  char filename[128]; // Для имени файла с открытым ключом
  char PublicKey[600]; // Для самого открытого ключа
  LPMAPITABLE pAdrTbl = NULL; // Таблица адресов в адресной книге
  int i,j;
  LPSRowSet pRow = NULL;
  SBinary sbEID;
  HRESULT hRes;
  ULONG FAR i_l;
  LPSPropValue pspvSaved = NULL;
  ULONG FAR lpulType;
  LPMESSAGE pMsg=NULL; // Для найденного адресата
  ULONG cVals;

  // Текущая строка в таблице адресатов
  cur_rec=StringGrid1->Selection.Top;

start:

  memset(PublicKey,0,600);
  if(StringGrid1->Cells[0][cur_rec]=="")
    return;
  sprintf(buff,"Укажите имя файла с открытым ключом
  абонента   %s",StringGrid1->Cells[1][cur_rec]);
  OpenDialog1->Title = buff;

  // Диалог открытия файла с открытым ключом
  if(OpenDialog1->Execute()!=IDOK)
    return;
  strcpy(filename,OpenDialog1->FileName.c_str());

  // Открываем и считываем файл с открытым ключом
  if((fl=fopen(filename,"rb"))==NULL)
  {
    if(strrchr(filename,'\\')!=NULL)
      sprintf(buff,\
      "Ошибка при открытии файла %s",strrchr(filename,'\\')+1);
    else
      sprintf(buff,"Ошибка при открытии файла %s",filename);
```

```
    MessageBox(0,buff,"ОШИБКА",MB_ICONERROR | MB_OK);
    return;
  }
  // Длина файла с открытым ключом ограничена
  if(filelength(fileno(fl))>PUBLLENGTH)
  {
file_err:
    if(strrchr(filename,'\\')!=NULL)sprintf(buff,\
      "Файл %s не содержит открытого ключа",strrchr(filename,'\\')+1);
else
      sprintf(buff,"Файл %s не содержит открытого ключа",filename);
    MessageBox(0,buff,"ОШИБКА",MB_ICONERROR | MB_OK);
    fclose(fl);
    goto start;
  }
  // Считываем открытый ключ
  j=fread(PublicKey,1,PUBLLENGTH,fl);
  fclose(fl);

  if(j!=PUBLLENGTH)
  {
err1:
    MessageBox(0,"Ошибка при чтении из файла","",\
    MB_ICONERROR | MB_OK);
    return;
  }
  for(i=0;i<j;i++)
    if(PublicKey[i]==0)
      PublicKey[i]=' ';

  // Открываем таблицу адресов MAPI
  hRes = pABCont->GetContentsTable(MAPI_DEFERRED_ERRORS,&pAdrTbl);
  if(hRes!=S_OK)
  {
    MessageBox(0,"Не найдена таблица адресов","",\
    MB_ICONERROR | MB_OK);
    return;
  }
```

```
// Определяем количество адресов
hRes=pAdrTbl->GetRowCount(0,&i_l);
if(hRes<0)
{
  MessageBox(0,"Ошибка в таблице адресов","",MB_ICONERROR | MB_OK);
  return;
}

hRes=pAdrTbl->QueryRows(i_l,0,&pRow);
if(hRes!=S_OK)
{
  MessageBox(0,"Ошибка при опросе таблицы адресов","",\
  MB_ICONERROR | MB_OK);
  return;
}
// Ищем нужный адрес и его идентификатор
for(i=0;i<pRow->aRow[cur_rec-1].cValues;i++)
  if(pRow->aRow[cur_rec-1].lpProps[i].ulPropTag== PR_ENTRYID)
    sbEID=pRow->aRow[cur_rec-1].lpProps[i].Value.bin;

// Открываем свойства нужного адресата
hRes = pABCont -> OpenEntry(sbEID.cb,(LPENTRYID)sbEID.lpb, \
&IID_IMessage,MAPI_DEFERRED_ERRORS | MAPI_MODIFY,&lpulType,\
 (LPUNKNOWN *)&pMsg);


if(hRes!=S_OK)
{
  MessageBox(0,"Ошибка при открытии адресата","",\
  MB_ICONERROR | MB_OK);
  return;
}

// Свойства адресата
hRes = pMsg -> GetProps(NULL,0,&cVals,&pspvSaved);
if(hRes!=S_OK)
{
  MessageBox(0,"Ошибка при опросе свойств адреса","",\
  MB_ICONERROR | MB_OK);
```

```
    return;
  }

  // Вписываем в свойство PR_COMMENT открытый ключ
  memset(buff,0,1024);

  for(i=0;i<cVals;i++)
    if(pspvSaved[i].ulPropTag==PR_COMMENT)
      strcpy(buff,pspvSaved[i].Value.lpszA);

  strcat(buff,PublicKey);

  pspvSaved[0].ulPropTag = PR_COMMENT;
  pspvSaved[0].Value.lpszA=buff;
  hRes=pMsg->SetProps(1,pspvSaved,NULL);
  hRes=pMsg->SaveChanges(0);
  if(hRes!=S_OK)
    MessageBox(0,"Ошибка при сохранении изменений","",\
    MB_ICONERROR | MB_OK);

  // Корректировка таблицы формы FormPAB
  FormCorrect();
}
```

Формат открытых ключей, используемых в адресной книге, целесообразно выбрать точно таким же, что и при организации базы данных открытых ключей при работе с файлами. Файл с открытым ключом вырабатывается из секретного ключа в интерактивном режиме с помощью упоминавшегося в *главе 12* модуля MCS-key (см. рис. 15.4).

При выработке открытого ключа пользователь имеет возможность самостоятельно установить дату и время его ввода в эксплуатацию.

После выработки файл с открытым ключом проходит сертификацию у менеджера СБ. Полученный сертифицированный файл с открытым ключом рассылается всем адресатам, с которыми будет вестись переписка в защищенном режиме. Получатель файла с открытым ключом добавляет его к себе в адресную книгу самостоятельно.

Как видно из рис. 15.4, пользователь сам может выступить в роли менеджера СБ и сертифицировать свой собственный или иной открытый ключ (функция **Подписать файл с открытым ключом**).

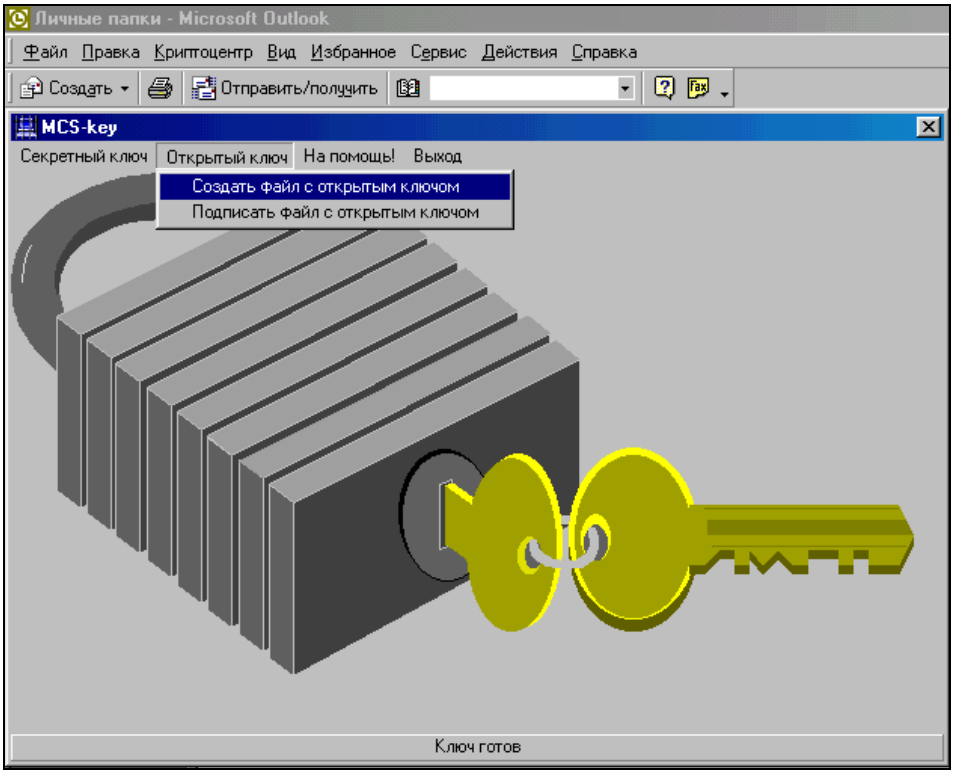

Рис. 15.4. Выработка файла с открытым ключом

Отметим, что сертификация открытых ключей в системе Криптоцентр для Microsoft Outlook необязательна. Система полностью сможет выполнять все криптографические функции без сертифицированных открытых ключей. В этой ситуации вся ответственность за гарантию подлинности используемых открытых ключей ложится на самого пользователя. Если пользователь твердо уверен, что используемые открытые ключи — истинные, то он может отказаться от их сертификации. В этом случае, однако, при выполнении операции отправки в защищенном режиме, требующей открытого ключа, при обнаружении несертифицированного открытого ключа система будет выдавать предупреждение, приведенное ниже на рис. 15.5.

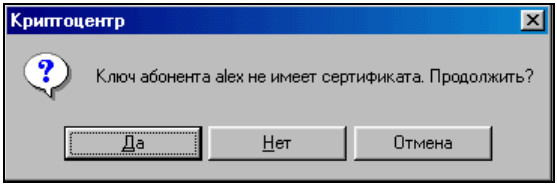

Рис. 15.5. Предупреждение об отсутствии сертификата открытого ключа

Как уже отмечалось в *главе 13*, в случае обнаружения несертифицированного ключа при проверке подписи, результат проверки выдается маленькими буквами.

Подведем теперь некоторые итоги рассмотрения системы Криптоцентр для Microsoft Outlook.

- Система Криптоцентр для Microsoft Outlook подключается к MAPI с помощью стандартного механизма Exchange Client Extension и добавляет в Microsoft Outlook возможности осуществления различных криптографических операций.
- Пользователь системы Криптоцентр с ее помощью может самостоятельно выполнить все основные криптографические операции: выработку собственного секретного и открытого ключа, прием открытых ключей адресатов, запись их в личную адресную книгу, отправку и прием сообщений в защищенном режиме, шифрование сообщений для их хранения в зашифрованном виде, подпись сообщений.
- При желании, несколько пользователей, ведущих переписку в защищенном режиме, могут договориться между собой и сертифицировать свои открытые ключи у менеджера системы безопасности. Таким менеджером, например, может быть один из этих пользователей.
- Система Криптоцентр позволяет зашифровывать не только сообщения, пересылаемые по электронной почте, но и отдельные поля в типовых сервисах Microsoft Outlook: **Контакты**, **Заметки**, **Дневник**, **Задачи** и т. п.

Таким образом, система Криптоцентр для Microsoft Outlook может быть полезна, в основном, для индивидуальных пользователей, общающихся друг с другом по электронной почте; для организации удаленных рабочих мест; для небольших компаний, заключающих по электронной почте сделки со своими контрагентами, ведущих электронную торговлю; для связи с поставщиками и заказчиками; да и во многих других подобных Internet-приложениях.

Часть IV

Встраивание гарантированных алгоритмов в Lotus Notes

Глава 16. Notes API-приложения

Глава 17. Ключевая книга

Глава 18. База данных с открытыми ключами

Глава 19. Шифрование с использованием ключевой книги

Глава 20. Шифрование по списку

Глава 21. Организация электронной подписи

Lotus Notes — это программная среда для рабочих групп, повышающая эффективность совместной работы. Notes обеспечивает совместную работу пользователей, использующих разные операционные системы и аппаратное обеспечение, независимо от разделяющих их технических, организационных и географических границ.

Notes состоит из двух основных программ: сервера Notes и клиента Notes.

Сервер Notes устанавливается на компьютер (сервер), работающий под управлением Windows и предоставляющий различные услуги пользователям клиентов Notes и другим серверам Notes, в том числе хранение общих баз данных и маршрутизацию почты.

Клиент Notes устанавливается на компьютер (рабочую станцию), работающий под управлением Windows и взаимодействующую с серверами Notes так, что его пользователь может осуществлять доступ в общие базы данных, а также получать и отправлять почту.

По данным Lotus, существует несколько десятков миллионов пользователей Notes. По словам одного из разработчиков Lotus Notes, Билла Крайсла, Notes и Domino:

- представляют технологию разработки баз данных;
- содержат процессор поточной обработки;
- используют составное хранение объектов;
- включают клиент-серверные вычисления;
- содержат текстовый процессор;
- содержат электронную почту;
- используют распределенные данные в масштабах предприятия;
- используют Intranet;
- являются мультиплатформной интегрированной средой разработки.

Более подробно система Notus описана в книге Крайсл Б. "Изучи сам Lotus Notes" / пер. с англ., Минск, ООО "Попурри", 1998.

Поскольку Lotus Notes является "мультиплатформной интегрированной средой разработки", обладающей к тому же большим набором сервисных возможностей, то имеются основания поставить задачу встраивания гарантированных криптографических алгоритмов в различные приложения Notes, используя имеющиеся в нем средства разработчика, называемые *Notes C API*, ориентированные на язык программирования C, а также такие механизмы Notes, как агенты и конструкторы. Этим вопросам и посвящена эта часть книги.

Глава 16

Notes API-приложения

Основными объектами Lotus Notes являются специализированные базы данных.

Каждая база данных Notes обычно содержит сведения о какой-то одной области, например: обсуждение нового продукта, новости, инструкции по технике безопасности или нормативные документы предприятия и т. п. База данных может быть предназначена как для личного, так и для общего пользования. Большинство баз данных предназначены для общего пользования. База данных Notes представляет собой отдельный файл, содержащий документы. Документы базы данных Notes могут рассматриваться как записи базы данных. Однако документ базы данных представляет собой сложный объект, который может содержать форматированный текст, рисунки, объекты и другие типы данных.

Большинство баз данных хранятся на одном или нескольких серверах Domino, доступных нескольким пользователям. По типу доступа базы данных разделяются на локальные базы данных и общие базы данных.

Документ Notes — это элемент базы данных, содержащий некоторую информацию. Размеры документов могут быть любыми: от краткого ответа на вопрос до многостраничного биржевого отчета, содержащего текст, диаграммы и рисунки.

Тип информации, которую можно ввести в документ, определяется типом полей в его форме. Например, текст вводится в текстовые поля, а числа — в числовые поля. В форматируемые поля можно поместить текст, рисунки, связи и вложенные файлы.

Разработка собственных приложений Notes сводится, как правило, к разработке собственных Notes-БД или собственных средств обработки существующих Notes-БД.

Для разработки собственных Notes — приложений на языке C в среде Lotus Notes имеется специализированная динамическая библиотека NNotes.DLL,

которая обеспечивает различные возможности обработки как баз данных целиком, так и отдельных записей в них. Описание всевозможных функций, имеющихся в библиотеке NNotes.DLL, можно найти в *Notes C API Reference*.

Существуют различные способы *подключения* собственных приложений к Lotus Notes. Их обзор изложен в *Notes C API User*, здесь же мы подробнее остановимся на способе добавления собственных команд в основное меню Lotus Notes, подобно тому, как это было осуществлено в Microsoft Outlook. Однако механизм подключения к основному меню в случае Lotus Notes будет уже другим. Сразу же оговоримся, что все подобные операции мы будем осуществлять на рабочей станции, на которой работает некоторый клиент Notes.

У каждой рабочей станции Notes имеется файл NOTES.INI, содержащий конфигурацию рабочей станции. Для подключения собственной динамической библиотеки, содержащей Notes-приложение, достаточно в файле NOTES.INI задать строку `AddInMenus`. Например, если наше криптографическое Notes-приложение (которое, аналогично со случаем Microsoft Outlook, мы назовем *Криптоцентр для Lotus Notes*) содержится в динамической библиотеке NMCS.DLL, то в файл NOTES.INI необходимо будет добавить строку

```
AddInMenus=NMCS.DLL
```

Что должно входить в состав криптографического Notes-приложения? По аналогии с системой Криптоцентр для Microsoft Outlook, Notes-Криптоцентр обеспечивает загрузку другой динамической библиотеки, содержащей криптографические процедуры, предназначенные для работы с записями в базах данных Notes. Механизм загрузки библиотеки с криптографическими процедурами точно такой же, как и для Microsoft Outlook (см. листинги 12.1 и 12.3). В результате, при запуске клиента Notes имеем запрос на ввод секретного ключа системы Криптоцентр (рис. 16.1).

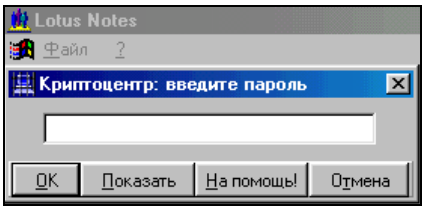

Рис. 16.1. Запуск клиента Notes с криптографическим приложением

Для подключения к основному меню Lotus Notes используем команду `AddInMenuProc` (описание команды — см. *Notes C API User, Chapter 11–4*).

Листинг 16.1. Формирование криптографических команд в основном меню Notes

```
/* Notes include files */
#include <global.h>
#include <addinmen.h>

#define IDM_KEY_INPUT 0
#define IDM_KEY_INPUT_FROM_REESTR 1
#define IDM_KEY_INPUT_FROM_TM 2
#define IDM_KEY_INFO 3
#define IDM_KEY_GENERATE 4
#define IDM_PUBLIC_KEYS 5
#define IDM_KEYBOOK 6

extern "C" __declspec(dllexport)NAMRESULT LNCALLBACK
AddinMenuProc (WORD wMsg, LONG lParam)
{
  char buff[256];
  GETKEYNAMEFROMREESTR ProcAdd1;
  KEYINP ProcAdd2;
  GETKEYNAME ProcAdd3;
  SETKEYNAMETOREESTR ProcAdd4;
  GENERATEKEY ProcAdd5;
  PKADRBOOK ProcAdd11;
  KEYBOOK ProcAdd13;

  memset(buff,0,256);
  if(Correct == false)return (NAM_NOERROR);

  switch (wMsg)
  {
  case NAMM_INIT:
  {
    NAM_INIT_INFO *pInitInfo;
    pInitInfo = (NAM_INIT_INFO *)lParam;

    if(pInitInfo->wMenuItemNumber==0)
    {
```

```
  pInitInfo->wMenuID = IDM_KEY_INPUT_FROM_REESTR;
  ConvertNormaToLotus("КЦ: &Ввод секретного ключа",\
  buff);
  strcpy(pInitInfo->MenuItemName,buff);
  return (NAM_INIT_CONTINUE);
}
if(pInitInfo->wMenuItemNumber==1)
{
  pInitInfo->wMenuID = IDM_KEY_INPUT;
  ConvertNormaToLotus("КЦ: Ввод секретного ключа из \
  &файла",buff);
  strcpy(pInitInfo->MenuItemName,buff);
  return (NAM_INIT_CONTINUE);
}
if(pInitInfo->wMenuItemNumber==2)
{
  pInitInfo->wMenuID = IDM_KEY_INPUT_FROM_TM;
  ConvertNormaToLotus("КЦ: Ввод секретного ключа из \
  &TM",buff);
  strcpy(pInitInfo->MenuItemName,buff);
  return (NAM_INIT_CONTINUE);
}
if(pInitInfo->wMenuItemNumber==3)
{
  pInitInfo->wMenuID = IDM_KEY_INFO;
  ConvertNormaToLotus("КЦ: &Информация о секретном \
  ключе",buff);
  strcpy(pInitInfo->MenuItemName,buff);
  return (NAM_INIT_CONTINUE);
}
if(pInitInfo->wMenuItemNumber==4)
{
  pInitInfo->wMenuID = IDM_KEY_GENERATE;
  ConvertNormaToLotus("КЦ: &MCS-key",buff);
  strcpy(pInitInfo->MenuItemName,buff);
  return (NAM_INIT_CONTINUE);
}
if(pInitInfo->wMenuItemNumber==5)
{
```

```
    pInitInfo->wMenuID = IDM_PUBLIC_KEYS;
    ConvertNormaToLotus("КЦ: &Открытые ключи",buff);
    strcpy(pInitInfo->MenuItemName,buff);
    return (NAM_INIT_CONTINUE);
  }
  if(pInitInfo->wMenuItemNumber==6)
  {
    pInitInfo->wMenuID = IDM_KEYBOOK;
    ConvertNormaToLotus("КЦ: &Ключевая книга",buff);
    strcpy(pInitInfo->MenuItemName,buff);
    return (NAM_INIT_STOP);
  }

}

case NAMM_COMMAND:
{
  NAM_COMMAND_INFO *cmdInfo;
  cmdInfo = (NAM_COMMAND_INFO *)lParam;
  if(cmdInfo->wMenuID==IDM_KEY_INFO)
  {
    memset(buff,0,128);
    if(memcmp((char *)sign_key,buff,80)==0)
    {
      sprintf(buff,\
      "%s\nВ память не загружен",keyname);
      MessageBox(0,buff,\
      "Текущий секретный ключ",\
      MB_ICONWARNING | MB_OK);
    }
    else
    {
      sprintf(buff,\
      "%s\nМетка ключа — %s\nЗагружен в \
      память",keyname,KeyLabel);
      MessageBox(0,buff,\
      "Текущий секретный ключ",MB_OK);
    }
    return (NAM_NOERROR);
```

```
    }
    if(cmdInfo->wMenuID==IDM_KEY_GENERATE)
    {
      ProcAdd5 = (GENERATEKEY)\
      GetProcAddress(hinstLib,"_GenerateSecretKey");
       (ProcAdd5)(sign_key);
      return (NAM_NOERROR);
    }
    if(cmdInfo->wMenuID==IDM_PUBLIC_KEYS)
    {
      ProcAdd11 = (PKADRBOOK)GetProcAddress\
      (hinstLib,"_NotesAdrBookWithPublicKey");
      if (ProcAdd11 != NULL)(ProcAdd11)();
        return (NAM_NOERROR);
    }
    if(cmdInfo->wMenuID==IDM_KEYBOOK)
    {
      ProcAdd13 = (KEYBOOK)\
      GetProcAddress(hinstLib,"_NotesKeyBook");
      if (ProcAdd13 != NULL)(ProcAdd13)();
        return (NAM_NOERROR);
    }

    memset(buff,0,128);
    ProcAdd1 = (GETKEYNAMEFROMREESTR)\
    GetProcAddress(hinstLib,"_GetKeyNameFromReestr");

    if (ProcAdd1 == NULL)
    {
err_key_inp:
      MessageBox(0,"Ошибка при вводе секретного \
      ключа","Криптоцентр",MB_ICONERROR | MB_OK);
      return (NAM_NOERROR);
    }
    else
    {
       (ProcAdd1)(buff);

      if(cmdInfo->wMenuID==IDM_KEY_INPUT)
```

```
{
  ProcAdd3 = (GETKEYNAME)\
  GetProcAddress(hinstLib,"_GetKeyName");
  if (ProcAdd3 == NULL)
    goto err_key_inp;
  if((ProcAdd3)(buff)!=0)
    goto err_key_inp;
}

if(cmdInfo->wMenuID==IDM_KEY_INPUT_FROM_TM)
{
  ProcAdd1 = (GETKEYNAMEFROMREESTR)\
  GetProcAddress(hinstLib,"_GetKeyNameFromTM");
  if (ProcAdd1 == NULL)
    goto err_key_inp;
  (ProcAdd1)(buff);
}
if(buff[0]==0)
  goto err_key_inp;
ProcAdd2 = (KEYINP)\
GetProcAddress(hinstLib,"_PrepearSecretKeyL");
if (ProcAdd2 == NULL)goto err_key_inp;
  if((ProcAdd2)\
  (buff,sign_key,crypt_key,KeyLabel,pi,l)!=0)
    goto err_key_inp;
strcpy(keyname,buff);
if(cmdInfo->wMenuID==IDM_KEY_INPUT||\
cmdInfo->wMenuID==IDM_KEY_INPUT_FROM_TM)
{
  sprintf(buff,"Установить %s как ключ \
  по умолчанию?",keyname);
  if(MessageBox(0,buff,"Криптоцентр",\
  MB_ICONQUESTION | MB_YESNO)!=ID_YES)\
    return (NAM_NOERROR);
  ProcAdd4 = (SETKEYNAMETOREESTR)\
  GetProcAddress(hinstLib,\
  "_SetKeyNameToReestr");
  (ProcAdd4)(keyname);
}
```

```
    } // else...
    return (NAM_NOERROR);
  }
  default:
    return (NAM_NOERROR);
  }
}

// Перекодировка русских букв под формат Notes
void ConvertNormaToLotus(char *buff,char *buff1)
{
  int i,j;
  i=j=0;
  memset(buff1,0,128);
  while(i<strlen(buff))
  {
    if(buff[i]<0)
    {
      buff1[j]=5;j++;
    }
    buff1[j]=buff[i];
    i++;
    j++;
  }
}
```

Результат подключения криптографических команд к основному меню Notes приведен ниже на рис. 16.2.

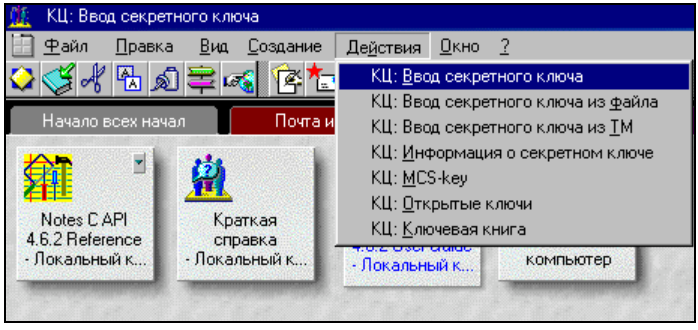

Рис. 16.2. Криптографические команды в основном меню Lotus Notes

Нетрудно заметить, что, в отличие от системы Криптоцентр для Microsoft Outlook, в основном меню Lotus Notes присутствуют только команды, связанные с секретными и открытыми ключами. Команды, с помощью которых осуществляются операции шифрования и подписи, индивидуальны для каждой базы данных Notes и в основном меню отсутствуют.

Есть смысл вкратце обсудить включенные в основное меню команды. Точнее говоря не все, а только те, которые имеют свою специфику для Notes. Таких здесь две, входящих последними в приведенное меню: **КЦ: Открытые ключи** и **КЦ: Ключевая книга**. Первые 5 команд — те же, что и аналогичные команды системы Криптоцентр для Microsoft Outlook.

При организации хранилища открытых ключей в Lotus Notes мы сталкиваемся примерно с теми же проблемами, что и в Microsoft Outlook. Использовать ли для хранилища открытых ключей адресную книгу или создавать независимое хранилище? Возможен как тот, так и другой вариант. Все зависит от потребностей пользователей и условий эксплуатации криптографической системы. Особенности использования адресной книги Notes в качестве хранилища открытых ключей будут рассмотрены в *главе 18*.

А вот ключевой книги в Microsoft Outlook не было. Что это такое? Давайте разбираться.

Как уже отмечалось ранее, основными объектами Notes являются базы данных, причем они могут быть как локальными, так и общими, хранящимися на сервере Domino. Каждый пользователь имеет свой индивидуальный секретный ключ и право доступа к нескольким базам данных, хранящимся на сервере. Естественно, в этих условиях целесообразно сделать еще *индивидуальные секретные ключи для каждой общей базы данных*. Информация, хранящаяся в этой базе, будет зашифровываться с помощью ее индивидуального секретного ключа. А все пользователи, имеющие доступ к этой базе данных, должны иметь доступ к этому ключу. Каким образом пользователь получает доступ к индивидуальному секретному ключу базы данных? С помощью ключевой книги. Она индивидуальна для каждого пользователя и в ней хранятся все индивидуальные секретные ключи баз данных, к которым пользователь имеет доступ. Но, естественно, хранятся не в открытом виде, а зашифрованные с помощью индивидуального секретного ключа пользователя. Более подробно об организации ключевой книги мы поговорим в *главе 17*.

Подведем краткий итог нашего первого знакомства с Lotus Notes.

- Основными объектами Lotus Notes являются специализированные базы данных. Они могут быть как локальными, так и общими, хранящимися на сервере.
- Для подключения к Lotus Notes гарантированных криптографических процедур мы будем использовать несколько способов: добавление

криптографических команд в основное меню, агенты и конструкторы баз данных.

- Команды, связанные с секретными и открытыми ключами, не зависят от выбранной базы данных и добавляются в основное меню Lotus Notes. Команды, связанные с шифрованием и подписью, формируются отдельно для каждой базы данных.
- Для шифрования информации, находящейся в общей базе данных, применяется индивидуальный секретный ключ этой базы. Он хранится в зашифрованном виде в ключевой книге пользователя.
- В качестве хранилища открытых ключей в Lotus Notes, по аналогии с Microsoft Outlook, можно использовать либо независимое хранилище, либо адресную книгу.

Глава 17

Ключевая книга

Рассмотрим более подробно ключевую книгу, используемую в системе Криптоцентр для Lotus Notes.

- Ключевая книга представляет из себя файл, содержащий набор секретных ключей, используемых при шифровании.
- Каждая база данных имеет свой индивидуальный секретный ключ в ключевой книге.
- Все секретные ключи в ключевой книге зашифрованы на индивидуальном секретном ключе пользователя.

Имя файла с ключевой книгой может быть прописано в реестре Windows. Сам файл с ключевой книгой не является секретным, поскольку все хранящиеся в ней ключи зашифрованы на индивидуальном секретном ключе пользователя. Для работы с ключевой книгой создадим небольшой интерфейс (рис. 17.1), позволяющий пользователю добавлять и удалять секретные ключи в ключевую книгу для различных Notes-БД.

Вот программная реализация процедуры построения этой формы.

Листинг 17.1. Процедура построения формы для работы с ключевой книгой

```
extern "C" __declspec(dllexport) int NotesKeyBook()
{
  char far PathName[512];
  char buff[512];

  // Считывание имени файла с ключевой книгой из реестра Windows
  TRegistry *Reg = new TRegistry();

  memset(keybook,0,128);
```

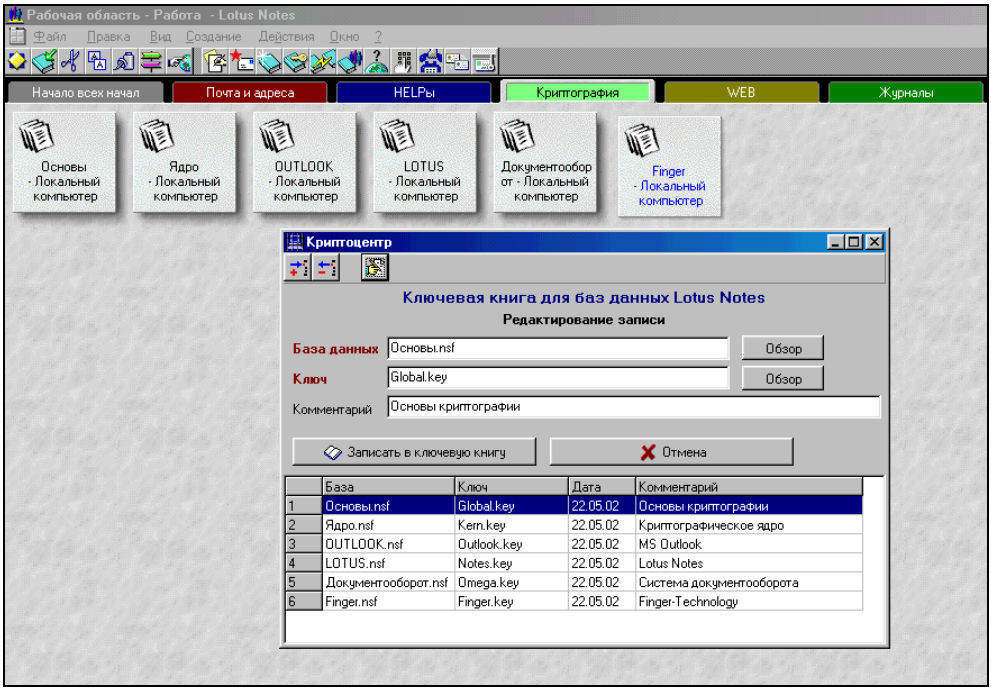

Рис. 17.1. Интерфейс для работы с ключевой книгой

```
Reg->RootKey = HKEY_CURRENT_USER;
if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\CryptoCentre\\Notes",\
false)==true)
  if(Reg->ValueExists("Ключевая книга")==true)
    strcpy(keybook,Reg->ReadString("Ключевая книга").c_str());
delete Reg;


// Если в реестре Windows нет ссылки на ключевую книгу,
// то она создается заново
if(keybook[0]==0)
{
  new_kb:
  FormNewKB = new TFormNewKB(NULL);
  OSGetDataDirectory(PathName);
  FormNewKB->DirectoryListBox1->Directory = PathName;

  // Имя файла с ключевой книгой по умолчанию
```

```
    sprintf(buff,"%s\\Notes.kb",PathName);
    FormNewKB->Edit1->Text = buff;
    if(FormNewKB->ShowModal()!=IDOK)
    {
      delete FormNewKB;return(-1);
    }
    strcpy(keybook,FormNewKB->Edit1->Text.c_str());
    delete FormNewKB;
    make_new_file(keybook);
    // Вносим ключевую книгу в реестр Windows
    TRegistry *Reg = new TRegistry();
    Reg->RootKey = HKEY_CURRENT_USER;
    if(Reg->OpenKey(\
    "\\SOFTWARE\\AlbaSoft\\CryptoCentre\\Notes",true)==true)
      Reg->WriteString("Ключевая книга",keybook);
    delete Reg;
  }

  // Если имя в реестре есть, но самого файла нет
  if(access(keybook,0)!=0)
    goto new_kb;

  // Форма для работы с ключевой книгой
  FormKB = new TFormKB(NULL);
  FormKB->StringGrid1->Cells[1][0] = "База";
  FormKB->StringGrid1->Cells[2][0] = "Ключ";
  FormKB->StringGrid1->Cells[3][0] = "Дата";
  FormKB->StringGrid1->Cells[4][0] = "Комментарий";
  FormKB->StringGrid1->RowCount = 2;

  if(FormKB->ShowModal()!=IDOK)
  {
    delete FormKB; return(1);
  }
  delete FormKB;
  return(0);
}
```

Все записи в ключевой книге — одинакового формата. Каждая запись — 512 байт. Вот процедура заполнения формы `FormKB` для работы с ключевой книгой.

Листинг 17.2. Заполнение формы для работы с ключевой книгой

```
// Структура для записей в ключевой книге
typedef struct{
  char dbName[32];
  int skLen;
  char sk[166];
  char dt[9];
  char comment[128];
  char FKName[32];
  char rezerv[141];
}KBREC;

int NKBRec; // Число записей в ключевой книге

void __fastcall TFormKB::FormActivate(TObject *Sender)
{
  FILE *fl;
  char buff[512];
  KBREC kbr;

  StringGrid1->RowCount = 2;
  StringGrid1->Rows[1]->Clear();
  NKBRec=0;
  if((fl=fopen(keybook,"r+b"))==NULL)
    return;

  // Каждая запись — 512 байт
  // Структура записи в KeyBook
  // 32 байта — имя файла с базой данных
  // 170 байт — ключ в зашифрованном виде
  // 9 байт — дата внесения ключа в КВ
  // 128 байт — комментарий
  // 173 байта — резерв
  while(!feof(fl))
```

```
  {
    memset(&kbr,0,512);
    fread(&kbr,512,1,fl);
    if(kbr.dbName[0]!=0)
    {
      NKBRec++;
      StringGrid1->RowCount = NKBRec+1;
      sprintf(buff,"%d",NKBRec);
      StringGrid1->Cells[0][NKBRec]= buff;
      StringGrid1->Cells[1][NKBRec]=kbr.dbName;
      StringGrid1->Cells[2][NKBRec]=kbr.FKName;
      StringGrid1->Cells[3][NKBRec]=kbr.dt;
      StringGrid1->Cells[4][NKBRec]=kbr.comment;
    }//if(kbr.dbName[0]!=0)

  }// while(!feof(fl))
  fclose(fl);
}
```

Теперь остановимся на том, какие ключи будем записывать в ключевую книгу. Как уже неоднократно отмечалось ранее, целесообразно иметь все ключи, одинаковые по формату, т. е. записывать в ключевую книгу в качестве индивидуальных ключей для каждой базы данных ключи в формате нашего обычного базового секретного ключа, который используется в качестве индивидуального секретного ключа пользователя. Однако в секретном ключе пользователя, как уже отмечалось в *главе 8*, предусмотрен пароль, не записанный в файле, но служащий дополнительным рубежом защиты при возможной компрометации цифровой части ключа. Этот пароль в индивидуальном ключе для базы данных не нужен! База данных — это не человек, запомнить и ввести с клавиатуры пароль она не может, поэтому при выработке секретного ключа для базы данных пароль в нем целесообразно *не вводить*. Все равно при использовании этого ключа он не играет роли. Зато в том случае, если для базы данных используется ключ без пароля, имеется возможность сделать *криптографически совместимыми* процедуры шифрования текстовых полей в Lotus Notes и Microsoft Outlook. Более подробно о всех особенностях процедур шифрования с использованием ключевой книги мы поговорим в *главе 19*, а сейчас, забегая вперед, приведем пример криптографической совместимости процедур шифрования текстовых полей.

Зашифруем в базе данных Notes некоторое поле с помощью ключевой книги. Например, так, как это показано на рис. 17.2 и 17.3.

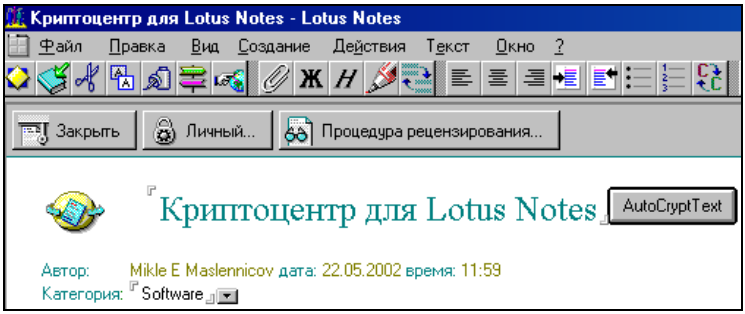

Рис. 17.2. Текстовое поле Notes-БД до зашифровывания с помощью ключевой книги

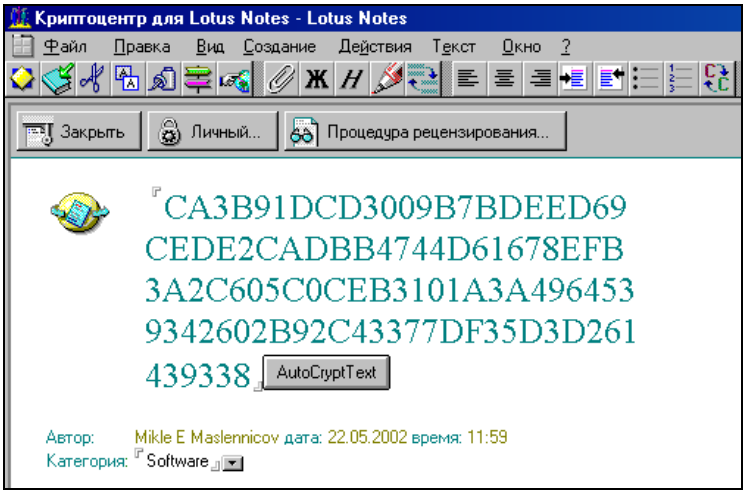

Рис. 17.3. Текстовое поле Notes-БД после зашифровывания с помощью ключевой книги

Выделяем зашифрованный текст и помещаем его в CLIPBOARD. Запускаем Microsoft Outlook и вставляем зашифрованный текст в одно из текстовых полей письма (например, в поле **Тема**).

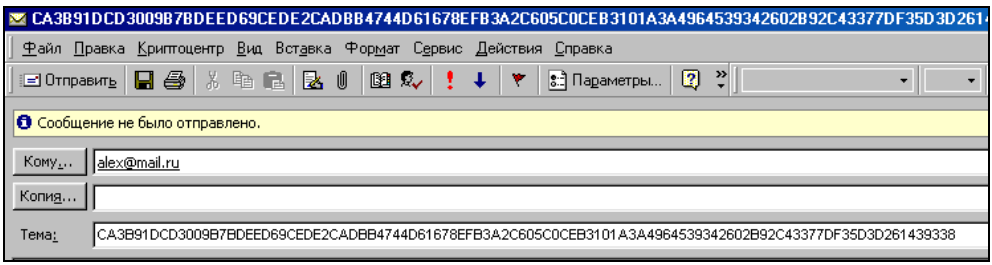

Рис. 17.4. Зашифрованное поле из Notes-БД в поле **Тема** сообщения Microsoft Outlook

В системе Криптоцентр для Microsoft Outlook загружаем в качестве секретного ключа секретный ключ из ключевой книги для рассматриваемой базы данных Lotus Notes. *Если этот ключ не имел пароля*, то в Microsoft Outlook возможно расшифрование текста, зашифрованного в Lotus Notes с использованием ключевой книги (рис. 17.5).

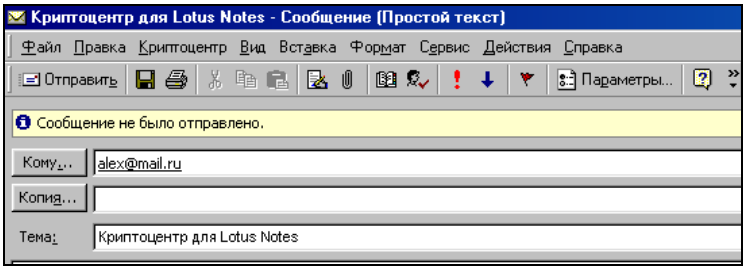

Рис. 17.5. Расшифрование поля, зашифрованного в Lotus Notes

Еще раз подчеркнем, что ключевая книга индивидуальна для каждого пользователя Lotus Notes, поскольку секретные ключи баз данных в ней зашифрованы с помощью индивидуального секретного ключа пользователя. Храниться она может в любом удобном месте: на локальном диске или на сервере. Изготовление ключевых книг может быть как индивидуальное, так и централизованное. Второй вариант более предпочтителен для крупных организаций с большим числом пользователей и баз данных Lotus Notes, однако при нем менеджер системы безопасности, занимающийся изготовлением ключевых книг, должен иметь секретные ключи всех пользователей. Возможен некоторый "промежуточный" вариант, когда центральный менеджер СБ изготавливает секретные ключи для различных баз данных и рассылает их региональным менеджерам, чтобы те на местах произвели их установку в ключевые книги обслуживаемых ими пользователей.

Наконец, возможна и такая ситуация, когда пользователь сам изготавливает и записывает в свою ключевую книгу секретный ключ для определенной базы данных. Это может быть справедливо, в первую очередь, для локальных баз данных, поля которых требуется защитить от посторонних при хранении. В системе Криптоцентр для Lotus Notes есть все необходимые для этого инструменты. Секретный ключ вырабатывается с помощью специализированного модуля *MCS-key* (см. рис. 16.2), пароль в нем целесообразно не задавать, метка ключа не играет роли, открытый ключ для такого файла не нужен. Файл с выработанным секретным ключом с помощью интерфейса для работы с ключевой книгой заносится в ключевую книгу, после чего он может быть удален или оставлен в надежном месте в качестве резервной копии.

Ключевая книга значительно упрощает процедуры зашифровывания/расшифрования полей в базах данных Lotus Notes, но в то же время позволяет четко разграничивать права пользователей к доступу к информации

в тех или иных базах. Доступ имеет только тот, у кого есть в ключевой книге секретный ключ соответствующей базы данных. Однако следует иметь в виду, что компрометация одной ключевой книги (компрометация секретного ключа одного пользователя и расшифрование всех секретных ключей в его ключевой книге) автоматически означает компрометацию всех баз данных, к которым этот пользователь имел доступ. Поэтому для особо важных баз данных возможно наличие нескольких секретных ключей для различных полей. В существующей реализации системы Криптоцентр для Lotus Notes разграничение прав доступа предусмотрено только на уровне баз данных, однако, используя приведенные в этой главе рассуждения, разработчик аналогичной системы без труда сможет их обобщить и при необходимости довести до уровня отдельных полей, отдельных пользователей, временных интервалов, увязать с определенными событиями и т. п. Все зависит от конкретных потребностей заказчика. Но следует помнить, что при шифровании с использованием ключевой книги мы *не используем систему с открытым распределением ключей*, стойкость не зависит от сложности решения задачи логарифмирования в конечных полях и поэтому с криптографической точки зрения является более высокой, чем, например, стойкость при реализации отправки/приема в защищенном режиме для Microsoft Outlook. Там стойкость не превосходит сложности логарифмирования, а здесь есть все возможности довести ее до уровня 10^{100}. При таких оценках всегда следует помнить, что потенциальный злоумышленник, не найдя способов криптографического взлома, будет искать "обходные пути", попытается вызвать компрометацию секретных ключей из ключевой книги. И в этих условиях неоднократно упоминаемые в этой книге требования по обеспечению обычной безопасности (не допускать криптовирусов, программных закладок, несанкционированного сканирования оперативной памяти) приобретают первостепенное значение. Среда Lotus Notes — сложная и многофункциональная интерфейсная оболочка, ориентированная на доступ к общим ресурсам с помощью различных коммуникационных средств, в ней легко затаиться потенциальному злоумышленнику. Не следует пренебрегать и имеющимися в стандартном программном обеспечении Lotus Notes средствами защиты. Но к вопросу о том, обеспечивают ли они гарантированную защиту, следует подходить весьма осторожно и скептически.

Глава 18

База данных с открытыми ключами

Как уже отмечалось в *главе 16*, проблемы, возникающие при построении хранилища для открытых ключей в Lotus Notes, во многом схожи с аналогичными проблемами для Microsoft Outlook. Поэтому в данной главе, аналогично как и в *главе 15*, посвященной открытым ключам в Outlook, мы уделим внимание только использованию адресных книг Lotus Notes в качестве хранилища открытых ключей.

Для просмотра всех существующих открытых ключей системы Криптоцентр для Lotus Notes мы создадим специальный интерфейс (см. рис. 18.1), визуально показывающий всю криптографически значимую информацию: идентификатор ключа, дату его изготовления и ввода в эксплуатацию, наличие сертификата.

Адресная книга в Notes — это просто специализированная база данных, одна среди многих, с которыми Notes имеет дело. Однако любая база данных типа **Адресная книга** имеет особый статус и особые средства обработки. Среди упоминавшихся в *главе 16* функций *Notes C API Reference* имеется функция `NAMEGetAddressBooks`, выдающая список и расположение всех адресных книг в Notes. Эту функцию мы будем использовать для поиска и перечисления всех личных адресных книг клиента Lotus Notes.

Итак, для того, чтобы использовать Notes-БД в качестве хранилища открытых ключей системы Криптоцентр, она, во-первых, должна быть типа **Адресная книга**, а, во-вторых, прописана как личная (локальная) адресная книга в параметрах настройки Lotus Notes. Из рис. 18.1 видно, что число абонентов, перечисленных в интерфейсе для работы с открытыми ключами, больше, чем число абонентов в одной адресной книге. Здесь мы имеем некоторое отличие от системы Криптоцентр для Microsoft Outlook: там все абоненты выбирались из одной локальной адресной книги. В Lotus Notes локальных адресных книг может быть несколько, но все они должны быть прописаны в параметрах настройки рабочего места Lotus Notes как локальные адресные книги (рис. 18.2).

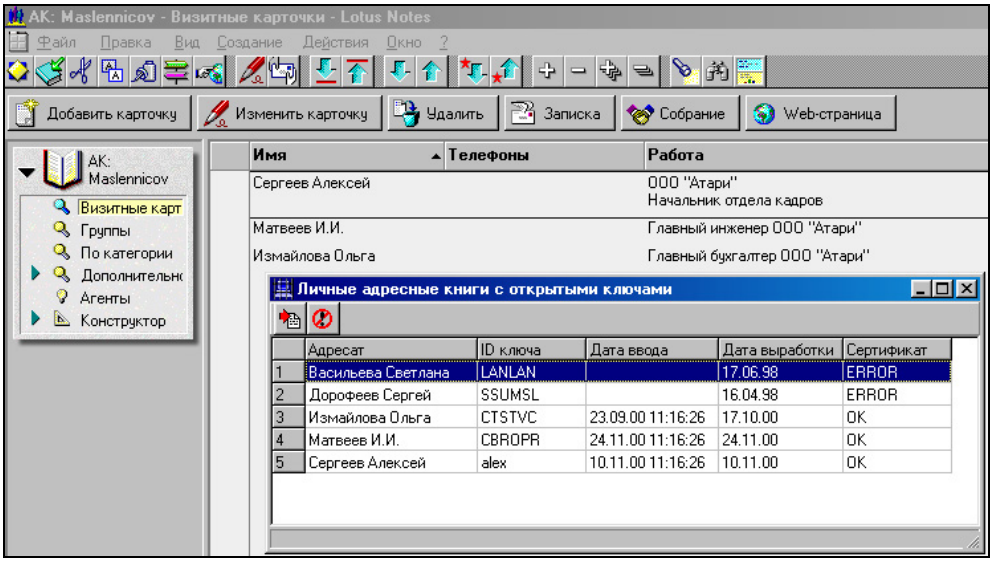

Рис. 18.1. Интерфейс для работы с открытыми ключами

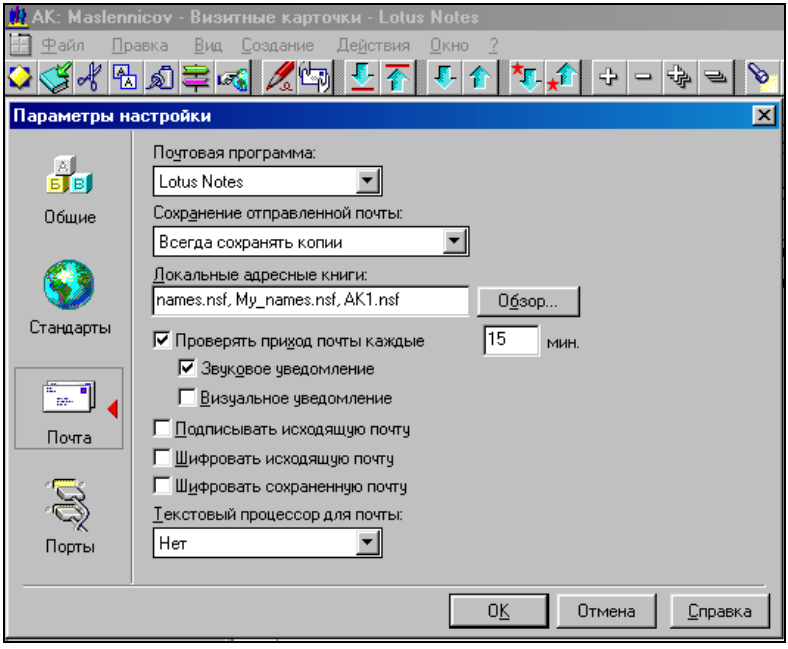

Рис. 18.2. Параметры настройки рабочего места Lotus Notes

Рассмотрим теперь листинг процедуры построения приведенной на рис. 18.1 формы для работы с открытыми ключами.

Листинг 18.1. Форма для работы с открытыми ключами

```
extern "C" __declspec(dllexport) int NotesAdrBookWithPublicKey()
{
  char buff[256];
  WORD    wOption;
  STATUS  error;
  WORD    wCount;
  WORD    wLength;
  WORD    wEntry;
  WORD    wEntryLen;
  HANDLE  hReturn;
  char   *pszReturn;
  char    achPort[MAXPATH];
  char    achServer[MAXPATH];
  DBHANDLE far rethDB;
  TRegistry * Reg; // Для определения открытого ключа менеджера СБ

  // Определяем из реестра Windows имя файла с открытым ключом
  // менеджера СБ
  Reg = new TRegistry();
  Reg->RootKey = HKEY_CURRENT_USER;
  if(Reg->OpenKey(\
  "\\SOFTWARE\\AlbaSoft\\CryptoCentre\\KeyMan",false)==true)
    strcpy(keyman,Reg->ReadString("Файл").c_str());
  delete Reg;

  // Перечисляем личные адресные книги Notes
  wOption = NAME_GET_AB_TITLES | NAME_DEFAULT_TITLES ;
  error = NAMEGetAddressBooks(NULL,wOption,&wCount,&wLength,&hReturn);
  if (error) return(ERR(error));
    if (!wCount)
    {
       MessageBox(0,"Ни одной личной адресной книги не
       найдено","Криптоцентр",MB_ICONERROR | MB_OK);
       return(ERR(error));
    }


  pszReturn = OSLock(char, hReturn);
```

```
  // Создаем форму для работы с открытыми ключами
  FormPublKeys = new TFormPublKeys(NULL);
  NField = 0;
  FormPublKeys->StringGrid1->Cells[1][0] = "Адресат";
  FormPublKeys->StringGrid1->Cells[2][0] = "ID ключа";
  FormPublKeys->StringGrid1->Cells[3][0] = "Дата ввода";
  FormPublKeys->StringGrid1->Cells[4][0] = "Дата выработки";
  FormPublKeys->StringGrid1->Cells[5][0] = "Сертификат";
  FormPublKeys->StringGrid1->RowCount = 2;


  // Просмотр всех адресных книг

  for (wEntry = 0; wEntry < wCount; wEntry++)
  {
    // Длина записи
    wEntryLen = strlen(pszReturn);

    // Подготовка к открытию базы данных
    OSPathNetParse(pszReturn, achPort, achServer, achFile);

    error = NSFDbOpen(achFile, &rethDB);
    if (error)
    {
err1:
      sprintf(buff,"Ошибка при открытии базы данных %s",achFile);
      MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
      return(ERR(error));
    }

    // Сканирование базы данных
    error = NSFSearch (
    rethDB,           // указатель БД
    NULLHANDLE,       // формула поиска (ищем все)
    NULL,             // заголовок формулы поиска (отсутствует)
    0,                // флаг поиска
    NOTE_CLASS_DATA,  // класс поиска — данные
    NULL,             // начальная дата (не используется)
```

```
    NoteAction,        // действия, совершаемые с найденной записью
    &rethDB,           // параметр в функции действий с записью
    NULL);             // конечная дата (не используется)

    // Сдвигаем указатель
    if (wOption & NAME_GET_AB_TITLES)
    {
      pszReturn += wEntryLen+1;
      wEntryLen = strlen(pszReturn);
    }

    pszReturn += wEntryLen+1;
  } // Конец просмотра адресных книг

  OSUnlock(hReturn);
  OSMemFree(hReturn);

  // Переупорядочивание таблицы формы
  // в лексикографическом порядке по именам
  NumFormPublKeys();
  FormPublKeys->ShowModal();
  delete FormPublKeys;

  return(0);
}
```

Основное место в листинге 18.1 занимает процедура сканирования базы данных — `NSFSearch`. В ней в качестве параметра приводится ссылка на функцию `NoteAction` — действия, совершаемые с найденной записью. Вот листинг этой функции.

Листинг 18.2. Функция `NoteAction`

```
STATUS LNPUBLIC NoteAction(
VOID far *db_handle,
SEARCH_MATCH far *pSearchMatch,
ITEM_TABLE far *summary_info)
{
  char buff[256];
  SEARCH_MATCH  SearchMatch;
```

```
  STATUS error;

  memcpy((char*)&SearchMatch, (char*)pSearchMatch,\
  sizeof(SEARCH_MATCH));

  if (!(SearchMatch.SERetFlags & SE_FMATCH))
    return (NOERROR);

  // Открываем запись
  if (error = NSFNoteOpen (
  *(DBHANDLE far *)db_handle,  // указатель Notes-БД
  SearchMatch.ID.NoteID,       // ID записи
  0,                           // флаг
  &note_handle))               // указатель записи
    return (ERR(error));

  // Запись открыта. Теперь сканируем все ее поля
  if (error = NSFItemScan (
            note_handle,    // указатель записи
            FieldAction,    // действия, совершаемые с полем
            &note_handle))  // параметр в функции действий с полем
  {
    NSFNoteClose (note_handle);
    return (ERR(error));
  }

  // Закрываем запись
  if (error = NSFNoteClose (note_handle))
    return (ERR(error));

  return (NOERROR);
}
```

Нелегко в Notes добраться до открытых ключей в адресной книге! Сначала мы перечисляли все личные адресные книги, открывали и сканировали соответствующие им базы данных. Сканирование базы данных осуществляется с помощью функции `NSFSearch`, в которой в качестве параметра указана другая функция, `NoteAction`, определяющая действия, которые надо совершить с найденной базой данных. Эти действия — открытие базы данных и сканирование всех ее записей. Сканирование записей осуществляется

с помощью функции `NSFItemScan`, в которой, в свою очередь, в качестве параметра дается ссылка на функцию `FieldAction`, определяющую действия, которые надо совершить с полями найденной записи. И только в функции `FieldAction` мы наконец-то имеем дело с открытыми ключами.

Листинг 18.3. Функция `FieldAction`

```
STATUS LNPUBLIC field_action
(WORD unused,
WORD item_flags,
char far *name_ptr,
WORD name_len,
VOID far *item_value,
DWORD item_value_len,
VOID far *note_handle1)
{
  STATUS error;
  char buff[10240];
  char buff1[1024];
  char    item_name[256];        /* name of the field */
  WORD    binary_datatype;       /* datatype of field in binary */
  char    ascii_datatype[256];   /* datatype of field in ASCII */
  int i;
  char IDkey[16];

  strncpy(item_name, name_ptr, name_len);
  item_name[name_len] = '\0';

  memcpy((char far *)(&binary_datatype), item_value, sizeof(WORD));

  // Графа "Адресат" в таблице открытых ключей
  if(stricmp(item_name,"FullName")==0)
  {
    NField++;
    FormPublKeys->StringGrid1->RowCount = NField+1;
    sprintf(buff,"%d",NField);
    FormPublKeys->StringGrid1->Cells[0][NField] = buff;

    // Получаем содержимое текстового поля
```

```
NSFItemGetText(note_handle,item_name,buff,(WORD) sizeof (buff));
// Перекодировка
ConvertLotusToNorma(buff,buff1);

if(strrchr(buff1,';')!=NULL)
  FormPublKeys->StringGrid1->Cells[1][NField] =
  strrchr(buff1,';')+1;
else
  FormPublKeys->StringGrid1->Cells[1][NField] = buff1;

// Для открытых ключей системы Криптоцентр
// используется специальное поле
if(NSFItemIsPresent (
note_handle,
"CryptoCentrePublicKey",
(WORD) strlen ("CryptoCentrePublicKey"))==true)
{

  i=NSFItemGetText(
  note_handle,
  "CryptoCentrePublicKey",
  buff,
  (WORD) sizeof (buff));
  // Находим все параметры открытого ключа
  FindPublicKey(buff,keyman);

  FormPublKeys->StringGrid1->Cells[2][NField]=IDKey;
  if(DateTime1!=NULL)
  {
    FormPublKeys->StringGrid1->Cells[3][NField]=
    DateTime1->DateTimeString();
    delete DateTime1;
  }
  else
    FormPublKeys->StringGrid1->Cells[3][NField] = "";

  sprintf(buff,"%02d.%02d.%02d",d.da_day,d.da_mon,d.da_year);
  if(d.da_day!=0)
    FormPublKeys->StringGrid1->Cells[4][NField] = buff;
```

```
      else
        FormPublKeys->StringGrid1->Cells[4][NField] = "";

      if(sert==true)
        FormPublKeys->StringGrid1->Cells[5][NField] = "OK";
      else
        FormPublKeys->StringGrid1->Cells[5][NField] = "ERROR";
      if(IDKey[0]==0)
        FormPublKeys->StringGrid1->Cells[5][NField] = "";

    }
  }
  return (NOERROR);
}
```

В отличие от Криптоцентра для Microsoft Outlook, в адресных книгах Notes мы легко можем создать специальные поля под открытый ключ. Стандартный интерфейс Lotus Notes с помощью конструктора позволяет отображать их на экране в удобном для нас виде, например так, как это показано ниже на рис. 18.3.

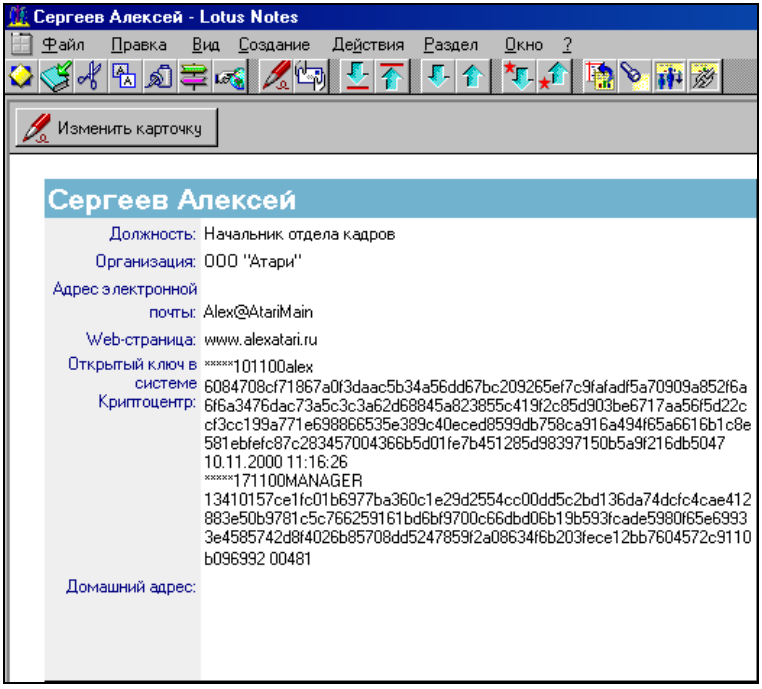

Рис. 18.3. Запись в личной адресной книге Notes с открытым ключом системы Криптоцентр

Все остальные проблемы хранилища открытых ключей в Lotus Notes решаются так же, как и в Microsoft Outlook. Сертификация открытых ключей — аналогичная, открытый ключ менеджера СБ может быть точно таким же, храниться для обеих этих систем в одном и том же месте. Файлы с открытыми ключами, добавляемыми в адресную книгу Notes, точно такие же, что и для Microsoft Outlook, их выработка и рассылка может быть произведена один раз для обеих этих систем. Для добавления и удаления открытых ключей в Notes используется специальный интерфейс, в котором применяются упоминавшиеся в листингах 18.1—18.3 стандартные функции для работы с Notes-БД и их полями.

Глава 19

Шифрование с использованием ключевой книги

Шифрование с использованием ключевой книги — это, на мой взгляд, самый простой способ организации гарантированной защиты информации в Lotus Notes. О таком шифровании уже шла речь в *главах 16* и *17*, напомним и еще раз подчеркнем основные его особенности.

- Для каждой Notes-БД, содержащей конфиденциальную информацию, вырабатывается отдельный секретный ключ. Этот ключ будет использоваться в качестве ключа для шифрования отдельных полей в записях этой БД. Система с открытым распределением ключей в этом случае не используется.
- Каждый пользователь, допущенный к базе данных, содержащей конфиденциальную информацию, получает ее секретный ключ и заносит его в свою индивидуальную ключевую книгу. В ключевой книге секретный ключ для каждой БД хранится в зашифрованном виде. Шифрование секретного ключа для каждой БД в ключевой книге осуществляется с помощью личного секретного ключа пользователя.

Таким образом, процедуры зашифровывания и расшифрования с использованием ключевой книги сводятся к следующим операциям:

- поиск в ключевой книге секретного ключа соответствующей базы данных;
- подготовка найденного секретного ключа к использованию;
- зашифровывание/расшифрование отдельных полей в записях базы данных.

Вот листинг реализации первых двух этих операций.

Листинг 19.1. Поиск и подготовка секретного ключа при шифровании с использованием ключевой книги

```
// dbName — имя файла с Notes-БД
// pi, k — параметры секретного ключа пользователя
int GetKeyFromKB(char *dbName,unsigned char *pi,unsigned char *k)
```

```
{
  FILE *fl;
  int i;
  KBREC kbr;

  // Определяем имя файла с ключевой книгой
  TRegistry *Reg = new TRegistry();

  Reg->RootKey = HKEY_CURRENT_USER;
  if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\
  CryptoCentre\\Notes",false)==true)
    if(Reg->ValueExists("Ключевая книга")==true)
      strcpy(keybook,Reg->ReadString("Ключевая книга").c_str());
  delete Reg;

  if(keybook[0]==0)return(-1);

    // Открываем файл с ключевой книгой
    if((fl=fopen(keybook,"rb"))==NULL)return(-1);

      while(!feof(fl))
      {
        memset(&kbr,0,512);
        fread(&kbr,512,1,fl);
        if(stricmp(kbr.dbName,dbName)==0)
        {
          fclose(fl);
          goto key_ok;
        }
      }
  // Ключ для Notes-БД не найден
  fclose(fl);
  return(-1);

key_ok:
  // Две процедуры для подготовки найденного секретного ключа
  // к использованию
  if(DecryptMemASCII(kbr.sk,kbr.skLen)!=0)return(-1);
    if(PrepareKeyForNotes(kbr.sk,pi,k)!=0)return(-1);
      return(0);
}
```

Вариантов зашифровывания и проверки секретного ключа для Notes-БД может быть множество. Все эти операции осуществляются в приведенном ранее листинге 19.1 с помощью процедур `DecryptMemASCII` и `PrepareKeyForNotes`. В них используются не раз уже описанные в этой книге функции из нашего криптографического ядра: `CryptBlock`, `Crypt`, `DecryptBlock`, `Decrypt`, а также универсальное средство для распознавания зашифрованного текста — обобщенный маркант. Читателю, прочитавшему вторую часть книги, не составит особого труда написать собственную реализацию этих процедур.

Рассмотрим теперь более подробно организацию операций зашифровывания/расшифрования отдельных полей в записях Notes-БД. Как уже отмечалось в *главе 16*, эта организация индивидуальна для каждой базы данных и может быть реализована с помощью типовых средств, имеющихся в Lotus Notes — агентов и конструкторов. Общие сведения об агентах и конструкторах можно получить из справочной документации по Lotus Notes, вот пример взятого оттуда описания агента.

> Агенты подобны работникам сцены в театральном производстве. Они работают "за сценой" — заполняют документы, отправляют почту, выполняют поиск, архивируют устаревшие документы. Пользовательские агенты, созданные разработчиком или пользователями, могут выполнять нетривиальные функции, например изменение значений полей или выборку данных из других приложений.
>
> Агенты могут быть либо личными — созданными и используемыми лично вами, либо общими — созданными разработчиком и используемыми коллективом сотрудников. Личные и общие агенты являются элементами структуры и хранятся в базе данных. Тип агента (личный или общий) не может быть изменен после создания агента.
>
> Агент содержит формулу отбора документов, триггер и одно или несколько действий. Агент использует простые типы действий, которые могут быть связаны с представлением, и действия формы: стандартное действие Notes, формулу с @-функциями или программу LotusScript. В отличие от действий представления или формы, которые могут быть выполнены только в открытом представлении или документе, действия агента работают с предварительно выделенным набором документов, включаются в предопределенное время (ежечасно, ежедневно, еженедельно, ежемесячно) или при выполнении заданных условий (по приходе почты, при изменении документов, при вставке документов, а также если агент запущен из меню).
>
> Используете ли вы базы данных Notes или разрабатываете их, вы всегда найдете подходящие агенты, которые помогут вам работать быстро и эффективно.

Конструктор позволяет разработчику обеспечивать удобное представление базы данных и выполнение некоторых действий с ее записями. В нем также могут быть использованы стандартные действия Notes, формулы с @-функциями или программы LotusScript.

Таким образом, механизмы, используемые как в агентах, так и в конструкторе Lotus Notes, основаны на стандартных действиях Notes, формулах

с @-функциями и программах LotusScript. С точки зрения своих возможностей, более предпочтительными для встраивания криптографических операций в Notes-БД являются программы LotusScript. В частности, в них имеется такая важная возможность, как загрузка внешней динамической библиотеки и выполнение отдельных процедур из нее. Поэтому все операции шифрования с использованием ключевой книги можно реализовать в нашей криптографической DLL-библиотеке (ее далее будем обозначать MCSNOTES.DLL), а их использование для каждой конкретной базы данных организовывать с помощью программ LotusScript, используя их или в агентах или в иных действиях, реализованных с помощью конструктора.

Наиболее просто реализовать операции шифрования простых текстовых полей. Причем операции зашифровывания и расшифрования можно реализовать с помощью одной процедуры, используя обобщенный маркант: если при анализе текста обнаружен обобщенный маркант, то выполняется операция расшифрования, в противном случае — зашифровывание. Вот пример реализации процедуры шифрования простых текстовых полей в записях Notes-БД.

Листинг 19.2. Шифрование простых текстовых полей с использованием ключевой книги

```
extern "C" __declspec(dllexport) PChar AutoCryptText
(PChar InputText,PChar LDBName)
{
  char *a;
  char buff[1024];
  char dbName[64];
  unsigned char pi[512];
  int i;

  // Процедура проверки введенного секретного ключа
  memset(buff,0,128);
  if(memcmp((char *)crypt_key,buff,80)==0)
  {
    MessageBox(0,"Не введен секретный ключ системы
    Криптоцентр","Криптоцентр",MB_ICONERROR | MB_OK);
    return(InputText);
  }

  memset(dbName,0,64);
  ConvertLotusToNorma(LDBName,dbName);
```

```
  // Получение секретного ключа БД из ключевой книги
  if(GetKeyFromKB(dbName,pi,pi+256)!=0)
  {
    sprintf(buff,
    "Ошибка при считывании ключа для базы данных %s",dbName);
    MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
    return(InputText);
  }

  a=(char *)malloc(2*strlen(InputText)+64);
  memset(a,0,2*strlen(InputText)+64);
  ConvertLotusToNorma(InputText,a);

  // Пытаемся зашифровать текст
  i=EncryptMem(NULL,a,strlen(a),pi,pi+256);

  // Если зашифровывание прошло успешно
  if(i==0)
    InputText = a;

  // Если при зашифровывании обнаружилось, что текст уже зашифрован,
  // то расшифровываем его
  if(i==1)
  {
    i=DecryptMem(NULL,a,strlen(a),pi,pi+256);
    // Если расшифрование прошло успешно
    if(i==0)
      InputText = a;
  }
  free(a);
  return(InputText);
}
```

Пример выполнения операции шифрования текстового поля приводился ранее в *главе 17* (см. рис. 17.2 и 17.3). Приведем теперь на рис. 19.1 и 19.2 программу LotusScript, реализующую в конструкторе кнопку **AutoCryptText**.

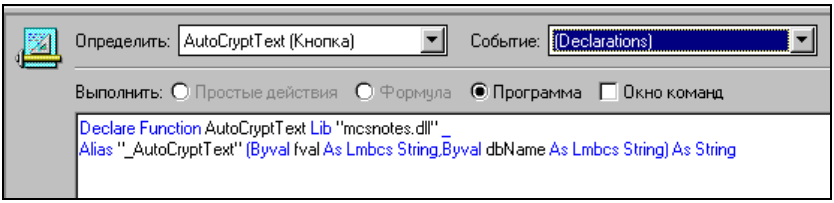

Рис. 19.1. Реализация кнопки **AutoCryptText**. Декларация

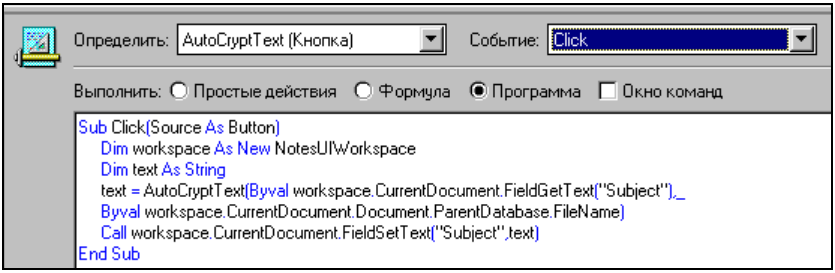

Рис. 19.2. Реализация кнопки **AutoCryptText**. Нажатие

Для текстовых полей в языке LotusScript имеются две очень удобные процедуры: `FieldGetText` и `FieldSetText`. `FieldGetText` позволяет получать значение шифруемого текстового поля в виде стринга, который затем подается на вход процедуры `AutoCryptText`. После выполнения операции зашифровывания/расшифрования с помощью процедуры `FieldSetText` полученное значение может быть сразу же отображено на экране.

Более сложными оказываются процедуры шифрования полей типа `RichText`. Каждое поле этого типа является, по сути, некоторым контейнером, содержащим несколько записей, различных по своему формату и назначению. В таких ситуациях в Lotus Notes применяется следующий прием: вводится процедура сканирования контейнера, в которой в качестве параметра выступает функция, определяющая действия, которые надо проводить с найденным объектом. С подобными ситуациями мы уже встречались в *главе 18* при работе с личными адресными книгами.

Имеются трудности и при внесении отдельных изменений в контейнер `RichText`. Проще всего оказывается просто переписать его заново, заменяя в нем зашифрованные или расшифрованные записи. Не любую запись можно будет шифровать. Полный список всевозможных типов записей в контейнере `RichText` мы приведем в *главе 21*, посвященной организации электронной подписи, а сейчас же примем за правило, что при шифровании с использованием ключевой книги в контейнерах типа `RichText` мы будем подвергать шифрованию только те их части, которые имеют тип `SIG_CD_TEXT` (85H) — текст.

С этими полями мы уже будем поступать как с обычными текстовыми полями. К ним можно применять процедуру автоматического шифрования

(если поле было зашифровано, то оно расшифровывается, в противном случае — зашифровывается), либо просто зашифровывания или просто расшифрования. Вот пример организации этих процедур.

Листинг 19.3. Организация шифрования `RichText`-полей

```
// Автоматическое шифрование (параметр A)
extern "C" __declspec(dllexport) int DBAutoCryptRichText(PChar
nIDstr,PChar FName,PChar LDBName)
{
  return(CryptRichText(nIDstr,FName,LDBName,'A'));
}


// Обычное зашифровывание (параметр E)
extern "C" __declspec(dllexport) int DBEncryptRichText(PChar nIDstr,PChar
FName,PChar LDBName)
{
  return(CryptRichText(nIDstr,FName,LDBName,'E'));
}


// Обычное расшифрование (параметр D)
extern "C" __declspec(dllexport) int DBDecryptRichText(PChar nIDstr,PChar
FName,PChar LDBName)
{
  return(CryptRichText(nIDstr,FName,LDBName,'D'));
}

// Обработка контейнера RichText
int CryptRichText(PChar nIDstr,PChar FName,PChar LDBName,char type)
{
  FILE *fl; // Для временного файла
  char buff[10240];
  STATUS        error;
  unsigned char pi[512]; // Для ключей
  char dbName[64]; // Notes-БД
  char tmpname[256]; // Имя временного файла для контейнера
  TRegistry * Reg;
  short i;

  BLOCKID item_blockid;
  WORD  wDataType;
  BLOCKID  bidRichText;
```

```
DWORD  dwItemSize;
HANDLE  hCompound;

// Проверка наличия секретного ключа в ОП
memset(buff,0,128);
if(memcmp((char *)crypt_key,buff,80)==0)
{
  MessageBox(0,"Не введен секретный ключ системы
  Криптоцентр","Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}

memset(dbName,0,64);
ConvertLotusToNorma(LDBName,dbName);

// Считывание секретного ключа из ключевой книги
if(GetKeyFromKB(dbName,pi,pi+256)!=0)
{
  sprintf(buff,
  "Ошибка при считывании ключа для базы данных %s",dbName);
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}

sscanf(nIDstr,"%d",&note_handle);

// Временный файл для контейнера
// В каталоге \\TMP создаем временный файл
strcpy(tmpname,"C:\\CENTRE\\TMP");

// Имя каталога для временного файла записано в реестре
Reg = new TRegistry();
Reg->RootKey = HKEY_CURRENT_USER;
if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\CryptoCentre",false)==true)
  if(Reg->ValueExists("TmpDir")==true)
    strcpy(tmpname,Reg->ReadString("TmpDir").c_str());
delete Reg;

strcat(tmpname,"\\NOTES.TMP");

if(make_new_file(tmpname)!=0)
```

```
{
  MessageBox(0,"Ошибка при создании временного
  файла","Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}


// Сначала создаем файл с нужным Rich-контейнером

error = NSFItemInfo(note_handle,FName,strlen(FName),
        &item_blockid,&wDataType,
&bidRichText,&dwItemSize);
if(error==ERR_ITEM_NOT_FOUND)
{
  sprintf(buff,"Поле %s не найдено",FName);
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}
if((fl_tmp=fopen(tmpname,"w+b"))==NULL)
  return(-1);

i=1;

fwrite(&i,2,1,fl_tmp);
// Перечисления полей в Rich-контейнере
if(type=='A')
  error = EnumCompositeBuffer(bidRichText,dwItemSize,
          &AutoCryptFields,pi);
if(type=='E')
  error = EnumCompositeBuffer(bidRichText,dwItemSize,
          &EncryptFields,pi);
if(type=='D')
  error = EnumCompositeBuffer(bidRichText,dwItemSize,
          &DecryptFields,pi);
fclose(fl_tmp);
if(error)
{
  sprintf(buff,
  "Ошибка при формировании временного файла с контейнером");
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
```

```
}

// Удаляем старый Rich-конейнер
error = NSFItemDelete(note_handle,FName,strlen(FName));
if(error)
{
  sprintf(buff,"Ошибка при удалении открытого Rich-текста");
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}
// Создаем новый Rich-контейнер
error = CompoundTextCreate(note_handle,FName,&hCompound);
if(error)
{
  sprintf(buff,
  "Ошибка при создании поля для шифрованного Rich-текста");
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}
// Вписываем временный файл в созданный Rich-контейнер
error = CompoundTextAssimilateFile(hCompound,tmpname,0L);
if(error)
{
  sprintf(buff,
  "Ошибка при записи временного файла в поле шифрованного
  Rich-текста");
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}

// Закрываем поле
error = CompoundTextClose(hCompound, NULL, NULL, NULL, 0);
if(error)
{
  sprintf(buff,
  "Ошибка при закрытии поля с шифрованным Rich-текстом");
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}
// Обновляем Notes-БД
error = NSFNoteUpdate(note_handle, 0);
```

```
  if(error)
  {
    sprintf(buff,"Ошибка при обновлении документа");
    MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }
  // Удаляем временный файл
  unlink(tmpname);
  return(0);
}
```

Одну из основных ролей в этом листинге играет процедура сканирования Rich-контейнера — EnumCompositeBuffer, в которой в качестве параметра выступают ссылки на процедуры действий с найденными объектами и их параметры. В качестве таких параметров мы берем секретный ключ, на котором необходимо осуществлять шифрование. Вот пример организации процедуры AutoCryptFields, остальные строятся аналогично.

Листинг 19.4. Процедура AutoCryptFields

```
STATUS LNCALLBACK AutoCryptFields(char *RecordPtr,WORD RecordType,DWORD
RecordLength,void *pCtx)
{
  int i;
  short j;
  char *a;
  unsigned char pi[512];

  if(RecordLength==0)return(NOERROR);
  // Шифруем только текстовые поля в контейнере
  switch(RecordType)
  {
  case SIG_CD_TEXT:
    memset(pi,0,512);
    memcpy(pi,(unsigned char *)pCtx,296);
    if((a=(char *)malloc(RecordLength*2+64))==NULL)
      return(ERROR);
    memcpy(a,RecordPtr+8,RecordLength-8);
    memcpy(&j,RecordPtr+2,2);
    i=EncryptMem(NULL,a,RecordLength-8,pi,pi+256);
```

```
    // Если зашифровывание осуществлено успешно
    if(i==0)
    {
      j=8+RecordLength*2+32;
      fwrite(RecordPtr,2,1,fl_tmp);
      fwrite(&j,2,1,fl_tmp);
      fwrite(RecordPtr+4,4,1,fl_tmp);
      fwrite(a,j-8,1,fl_tmp);
      free(a);
      break;
    }
    // Если текст был ранее зашифрован
    if(i==1)
    {
      i=DecryptMem(NULL,a,RecordLength-8,pi,pi+256);
      if(i!=0)
      {
        free(a);
        goto wrt;
      }
      j=8+(RecordLength-56)/2;
      fwrite(RecordPtr,2,1,fl_tmp);
      fwrite(&j,2,1,fl_tmp);
      fwrite(RecordPtr+4,4,1,fl_tmp);
      fwrite(a,j-8,1,fl_tmp);
      free(a);
      if(j%2!=0)fputc(' ',fl_tmp);
        break;
    }
    free(a);

  default:
    fwrite(RecordPtr,RecordLength,1,fl_tmp);
    if(RecordLength%2!=0)fputc(' ',fl_tmp);
  }
  return(NOERROR);
}
```

Шифрованный текст записывается в базу данных, но при этом он автоматически не отображается на экране, подобно тому, как это было для простых текстовых полей. На рис. 19.3—19.6 приведен пример организации шифрования `RichText`-полей и соответствующего действия, реализованного с помощью программы LotusScript.

Еще один объект, который может быть подвергнут шифрованию с использованием ключевой книги, — это файлы-вложения. При их шифровании мы будем применять идеи, в некоторой степени схожие с теми, которые применялись при шифровании `Rich`-контейнера.

- Каждый файл-вложение копируется во временный каталог и там шифруется как обычный файл. Первоначальный файл-вложение из документа удаляется, на его место пишется зашифрованный (или расшифрованный) файл.
- Для поиска всех файлов-вложений будем использовать реализованные в Notes функции перечисления вложений.
- Признаком зашифрованного файла будет обобщенный маркант. Благодаря ему, появляется возможность создания функции `AutoCryptOneNotesAttachment`, автоматически выполняющей операцию зашифровывания или расшифрования в зависимости от результатов поиска обобщенного марканта.

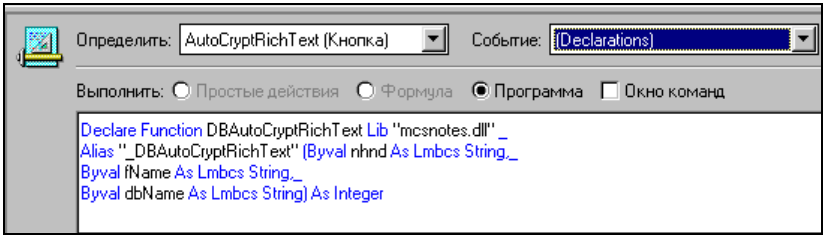

Рис. 19.3. Реализация кнопки **AutoCryptRichText**. Декларация

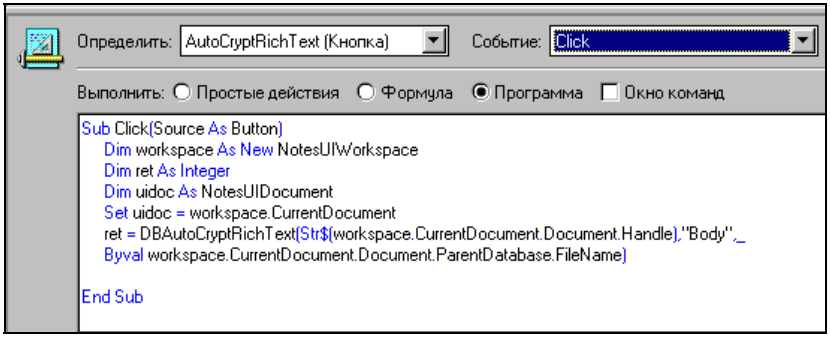

Рис. 19.4. Реализация кнопки **AutoCryptRichText**. Нажатие

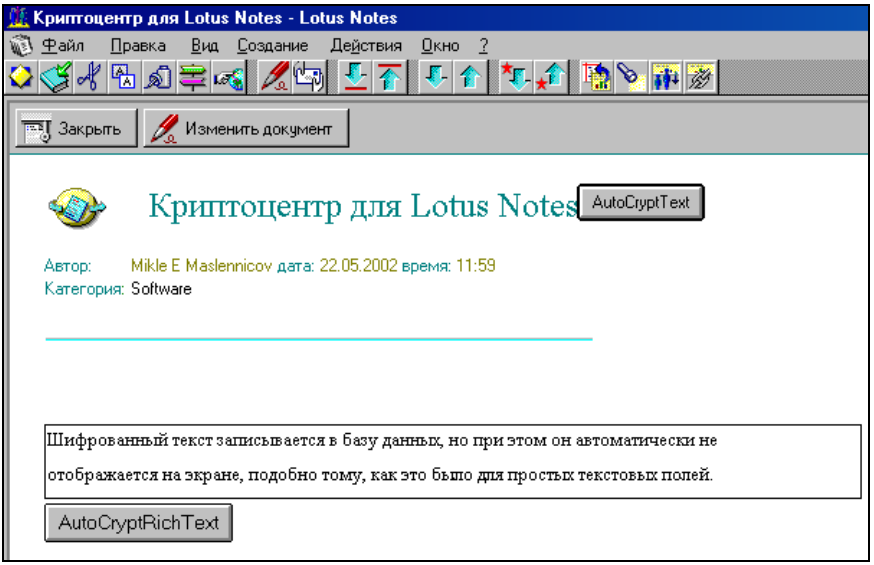

Рис. 19.5. Поле типа RichText до зашифровывания

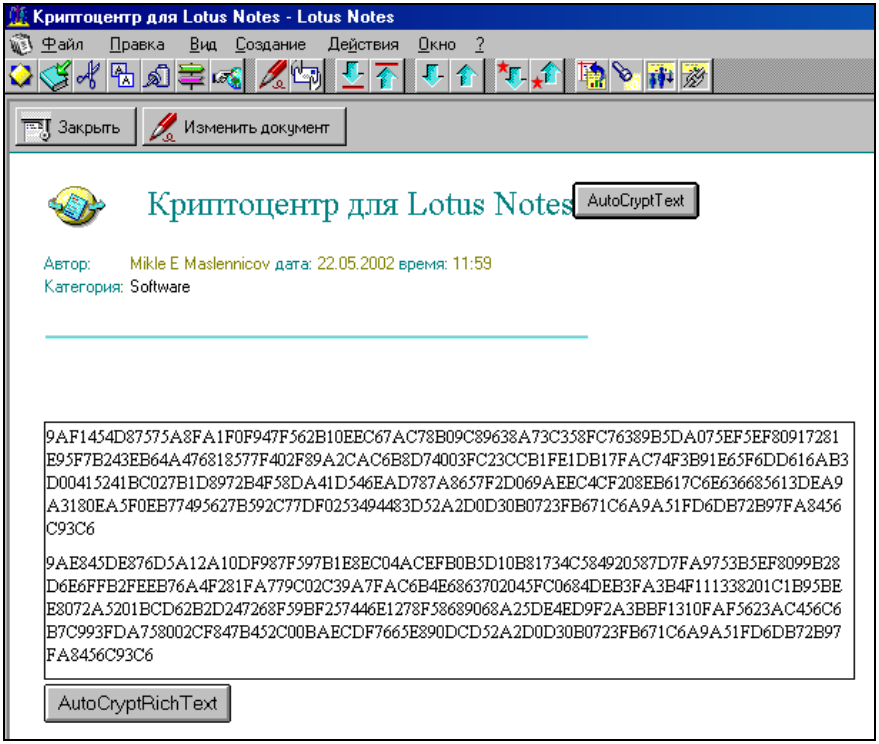

Рис. 19.6. Поле типа RichText после зашифровывания

Отметим, однако, одну особенность при реализации операций шифрования файлов-вложений. В них целесообразно дать возможность шифрования как всех вложений подряд, так и выборочно, отдельных, наперед заданных файлов-вложений. В этой связи будем предполагать, что при вызове процедуры шифрования файлов-вложений в ней присутствует в качестве параметра порядковый номер шифруемого вложения. Если, например, этот номер равен 0, то шифруются все файлы-вложения.

Вот пример реализации процедуры шифрования файлов-вложений.

Листинг 19.5. Процедура шифрования файлов-вложений

```
// AttNum — порядковый номер вложения
// Если AttNum=0, то шифруются все вложения

// Автоматическое шифрование (параметр A)
extern "C" __declspec(dllexport)
int DBAutoCryptAttachment(PChar nIDstr,short AttNum,PChar LDBName)
{
  return(CryptNotesAttachment(nIDstr,AttNum,LDBName,'A'));
}

// Обычное шифрование (параметр E)
extern "C" __declspec(dllexport) int DBEncryptAttachment(PChar
nIDstr,short AttNum,PChar LDBName)
{
  return(CryptNotesAttachment(nIDstr,AttNum,LDBName,'E'));
}

// Обычное расшифрование (параметр D)
extern "C" __declspec(dllexport) int DBDecryptAttachment(PChar
nIDstr,short AttNum,PChar LDBName)
{
  return(CryptNotesAttachment(nIDstr,AttNum,LDBName,'D'));
}



int CryptNotesAttachment(
PChar nIDstr,short AttNum,PChar LDBName,char type)
{
  char buff[10240];
```

```
STATUS        error;
unsigned char pi[512];
char dbName[64];
BLOCKID item_blockid;
WORD  wDataType;
BLOCKID  bidAttach;
DWORD  dwItemSize;
unsigned short AttCur;
unsigned short AttAll;
char tmpname[256]; // временный файл для вложения
TRegistry * Reg;
int i;
unsigned short j;

// Проверка наличия секретного ключа в ОП
memset(buff,0,128);
if(memcmp((char *)crypt_key,buff,80)==0)
{
  MessageBox(0,"Не введен секретный ключ системы
  Криптоцентр","Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}

// Считывание ключа из ключевой книги
memset(dbName,0,64);
ConvertLotusToNorma(LDBName,dbName);


if(GetKeyFromKB(dbName,pi,pi+256)!=0)
{
  sprintf(buff,
  "Ошибка при считывании ключа для базы данных %s",dbName);
  MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}

sscanf(nIDstr,"%d",&note_handle);

// Временный файл для вложения
```

```
// В каталоге \\TMP создаем временный файл
strcpy(tmpname,"C:\\CENTRE\\TMP");

Reg = new TRegistry();
Reg->RootKey = HKEY_CURRENT_USER;
if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\CryptoCentre",false)==true)
  if(Reg->ValueExists("TmpDir")==true)
    strcpy(tmpname,Reg- >ReadString("TmpDir").c_str());
delete Reg;

strcat(tmpname,"\\NOTES.TMP");

if(make_new_file(tmpname)!=0)
{
  MessageBox(0,"Ошибка при создании временного
  файла","Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}

if(AttNum!=0)
{
  AttCur = 0;
  // Поиск нужного вложения
  error = NSFItemInfo(note_handle,ITEM_NAME_ATTACHMENT,
  strlen(ITEM_NAME_ATTACHMENT),&item_blockid,&wDataType,
         &bidAttach,&dwItemSize);
  while(error!=ERR_ITEM_NOT_FOUND)
  {
    AttCur++;
    if(AttNum!=AttCur)
      goto next;

    // Нужное вложение найдено
    if(type=='A')
      i=AutoCryptOneNotesAttachment(\
      note_handle,item_blockid,bidAttach,pi,tmpname);
    if(type=='E')
      i=EncryptOneNotesAttachment(\
      note_handle,item_blockid,bidAttach,pi,tmpname);
```

```
      if(type=='D')
        i=DecryptOneNotesAttachment(\
        note_handle,item_blockid,bidAttach,pi,tmpname);
      if(i==-1)
      {
        sprintf(buff,"Ошибка при обработке вложения\
        № %d",AttCur);
        MessageBox(0,buff,"Криптоцентр",
        MB_ICONERROR | MB_OK);
      }
      break;

next:
      error = NSFItemInfoNext(note_handle,
            item_blockid,ITEM_NAME_ATTACHMENT,
            strlen(ITEM_NAME_ATTACHMENT),
            &item_blockid,&wDataType,
            &bidAttach,&dwItemSize);
    }
    if(error==ERR_ITEM_NOT_FOUND)
    {
      sprintf(buff,"Вложение № %d не найдено",AttNum);
      MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
      return(-1);
    }
  } // if(AttNum!=0)

  // Шифрование всех вложений
  else
  {
    AttAll=0;
    error = NSFItemInfo(note_handle,ITEM_NAME_ATTACHMENT,
            strlen(ITEM_NAME_ATTACHMENT),
            &item_blockid,&wDataType,
            &bidAttach,&dwItemSize);
    while(error!=ERR_ITEM_NOT_FOUND)
    {
      AttAll++;
      error = NSFItemInfoNext(note_handle,item_blockid,
              ITEM_NAME_ATTACHMENT,strlen(ITEM_NAME_ATTACHMENT),
```

```
                &item_blockid,&wDataType,&bidAttach,&dwItemSize);
    }
    if(AttAll==0)return(0);// Вложений нет, шифровать нечего
      for(j=0;j<AttAll;j++)
      {
        error = NSFItemInfo(note_handle,ITEM_NAME_ATTACHMENT,
             strlen(ITEM_NAME_ATTACHMENT),&item_blockid,&wDataType,
             &bidAttach,&dwItemSize);
        if(error!=ERR_ITEM_NOT_FOUND)
        {
          if(type=='A')
            i=AutoCryptOneNotesAttachment(\
            note_handle,item_blockid,bidAttach,pi,tmpname);
          if(type=='E')
            i=EncryptOneNotesAttachment(\
            note_handle,item_blockid,bidAttach,pi,tmpname);
          if(type=='D')
            i=DecryptOneNotesAttachment(\
            note_handle,item_blockid,bidAttach,pi,tmpname);
          if(i==1)
            continue;
          if(i==-1)
          {
err1:
            sprintf(buff,
            "Ошибка при обработке вложения № %d",j+1);
            MessageBox(0,buff,"Криптоцентр",
            MB_ICONERROR | MB_OK);
            return(-1);
          }
          error = NSFNoteUpdate(note_handle, 0);
          if(error)
            goto err1;
        } // if(error!=ERR_ITEM_NOT_FOUND)
      }// Цикл по всем вложениям
  } // if(AttNum==0)
  NSFNoteUpdate(note_handle, 0);
  unlink(tmpname);
  return(0);
}
```

А вот реализация упоминавшейся ранее процедуры `AutoCryptOneNotesAttachment`.

Листинг 19.6. Реализация процедуры автоматического шифрования одного файла-вложения

```
int AutoCryptOneNotesAttachment(NOTEHANDLE note_handle,BLOCKID
item_blockid,BLOCKID bidAttach,unsigned char *pi,char *tmpname)
{
  STATUS        error;
  unsigned char *pData;
  char attname[128];
  int i;
  short j;

  pData = OSLockBlock(unsigned char, bidAttach);

  j=0;
  memcpy(&j,pData+8,2);
  memset(attname,0,128);
  memcpy(attname,pData+38,j);
  OSUnlockBlock(bidAttach);

  // Раскрываем файл-вложение во временном каталоге
  error = NSFNoteExtractFile(note_handle,item_blockid,tmpname,NULL);
  if(error)
    return(-1);

  // Является ли файл зашифрованным
  i=test_file(pi,pi+256,tmpname);



  // Если при тестировании произошла ошибка
  if(i!=0&&i!=1)
    return(-1);
  // Для незашифрованного файла
  if(i==0)
  {
```

```
    i=crypt_file(pi,pi+256,40,tmpname,true);
    if(i!=0)
      return(-1);
  }
  // Для зашифрованного файла
  else
  {
    i=decrypt_file(pi,pi+256,40,tmpname,true);
    if(i!=0)return(-1);
  }
  // Удаляем старый файл-вложение
  error =  NSFNoteDetachFile(note_handle,item_blockid);
  if(error)
    return(-1);

  // Подсоединяем новый файл-вложение
  error = NSFNoteAttachFile (note_handle,ITEM_NAME_ATTACHMENT,
          strlen(ITEM_NAME_ATTACHMENT),tmpname,
          attname, COMPRESS_NONE);
  if(error)
    return(-1);

  return(0);
}
```

Пример реализации действия по зашифровыванию всех файлов-вложений на языке LotusScript приведен на рис. 19.7—19.11.

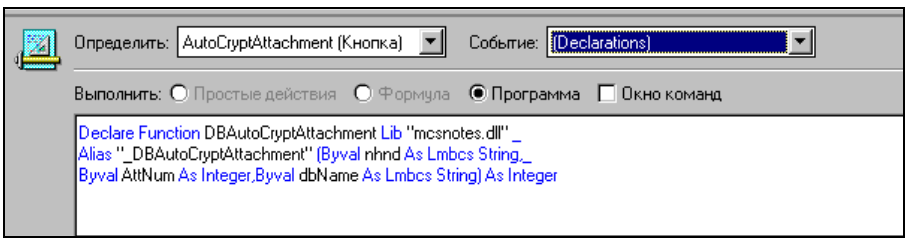

Рис. 19.7. Реализация кнопки **AutoCryptAttachment**. Декларация

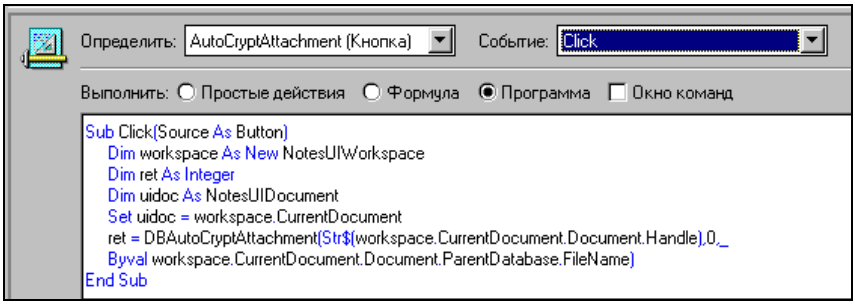

Рис. 19.8. Реализация кнопки **AutoCryptAttachment**. Нажатие

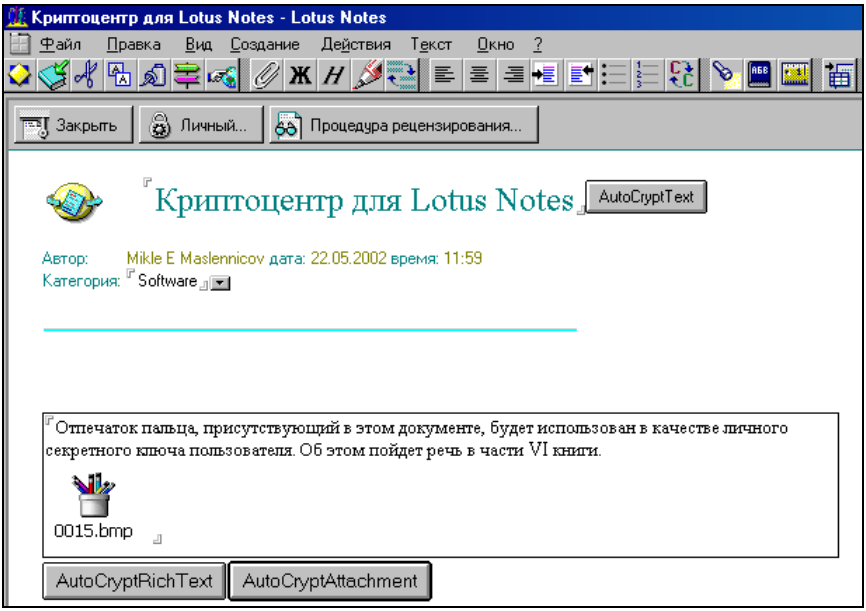

Рис. 19.9. Документ с файлом-вложением

Рис. 19.10. Файл-вложение до зашифровывания

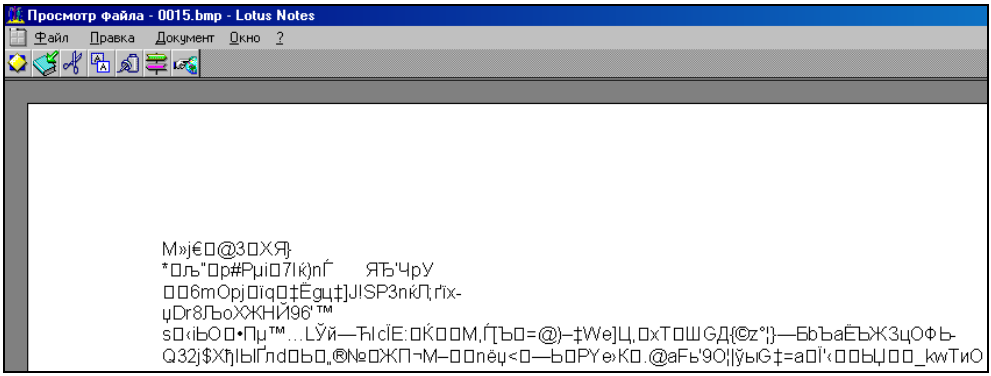

Рис. 19.11. Файл-вложение после зашифровывания

Описанные в этой главе методы работы с различными объектами в Notes-БД будут аналогичным образом применяться и при осуществлении операций шифрования по списку и электронной подписи. Об особенностях выполнения этих операций пойдет речь в следующих двух главах книги.

Глава 20

Шифрование по списку

Описанное в предыдущей главе шифрование с использованием ключевой книги удобно при централизованном разграничении прав доступа к определенным данным, хранящимся в Notes-БД. Однако на практике часто бывает необходимо, чтобы сам пользователь — он же исполнитель Notes-документа — устанавливал круг лиц, допущенных к содержащейся в нем информации. В этом случае возможно применение так называемого шифрования по списку, когда исполнитель документа заранее оговаривает список допущенных к отдельным его частям лиц и шифрует эти части таким образом, что расшифровать их могут только лица, указанные в этом списке.

Как на практике реализовать подобную схему? Очевидно, что здесь уже не обойтись без системы с открытым распределением ключей. Все лица, указанные в списке, должны иметь индивидуальные секретные ключи, и у пользователя-изготовителя документа должны быть их открытые ключи. Само шифрование по списку может быть построено следующим образом.

- Изготовитель документа вместе с документом вырабатывает случайный секретный ключ, использующийся при шифровании по списку.
- Секретный ключ, использующийся для шифрования по списку, в открытом виде нигде не хранится.
- Для каждого пользователя, указанного в списке, по схеме с открытым распределением ключей вырабатывается общий секретный ключ, который будет использоваться для шифрования секретного ключа, применяющегося при шифровании по списку.
- В документ вносится список допущенных к нему лиц, и для каждого такого пользователя в документ вписывается секретный ключ для шифрования по списку, зашифрованный на общем секретном ключе.

Таким образом, при шифровании по списку в документе вместе с зашифрованными полями должен присутствовать список допущенных к ним лиц и секретный ключ для шифрования по списку, сам в свою очередь

зашифрованный по схеме с открытым распределением ключей на общих ключах всех лиц из списка.

Вот такая, на первый взгляд, немного неуклюжая схема. Она вносит достаточно много дополнительной информации в документ (список пользователей и зашифрованный ключ для каждого пользователя из списка). Однако она дает возможность гибкого управления доступом к сведениям, содержащимся в Notes-документах, на уровне пользователя-изготовителя документа.

Большую часть этих проблем берет на себя программная реализация алгоритма шифрования по списку. Однако сам пользователь-изготовитель должен в интерактивном режиме задать список допущенных к нему лиц и только после этого выполнять операции шифрования.

Шифрование по списку отличается от шифрования с использованием ключевой книги только способом получения секретного ключа. Все остальные особенности, связанные с шифрованием текстовых полей, полей типа `RichText`, а также файлов-вложений, подробно рассмотренные в предыдущей главе, остаются без изменений. Поэтому далее в этой главе мы будем рассматривать только особенности организации ключевой системы при шифровании по списку, не касаясь больше самих вопросов шифрования полей в Notes-документах.

Выработка секретного ключа для шифрования по списку

Вспомним *главу 8*, посвященную вопросам выработки секретных ключей. Какой способ избрать для выработки секретного ключа для шифрования по списку? Ведь это может быть весьма динамичная процедура, пользователь-изготовитель документа может давать различные права для доступа к различным полям в документе, создавать множество различных документов с ограниченным правом доступа. Наверное, было бы неправильно каждый раз заставлять его для этого играть в компьютерную игру. Ведь он как пользователь должен иметь личный секретный ключ, который мы и постараемся использовать для генерации случайных секретных ключей, применяющихся при шифровании по списку. А идеи такой генерации тоже описаны в *главе 8*: постараемся приспособить для этих целей хэш-функцию. Какую информацию будем подвергать хэшированию? Очевидно, текущие дату и время. Вот листинг одного из возможных вариантов реализации процедуры выработки случайного секретного ключа для шифрования по списку.

Листинг 20.1. Выработка случайного секретного ключа для шифрования по списку

```
getdate(&d); // Текущая дата
gettime(&t); // Текущее время
```

```
sprintf(buff,"%d%d%d%d%d%d%d",d.da_day,d.da_mon,d.da_year,t.ti_hour,t.ti_
min,t.ti_sec,t.ti_hund);


HeshMem(buff,strlen(buff),hesh.h1); // Хэшируем текущую дату и время


memset(publ_crypt_key,0,80);
memcpy(publ_crypt_key,hesh.h1,64);


x[0]=d.da_year/256;
x[1]=d.da_year%256;
x[2]=d.da_mon;
x[3]=d.da_day;
x[4]=t.ti_hour;
x[5]=t.ti_min;
x[6]=t.ti_sec;
x[7]=t.ti_hund;


// Шифруем хэш-функцию с помощью личного секретного ключа пользователя
Crypt(x,pi,k,40,(unsigned char *)publ_crypt_key,56);
```

Какие поля должны быть добавлены в документ? Очевидно, что необходимы следующие поля.

- `CryptoCentreAbonents` — список абонентов, допущенных к определенному полю документа. В качестве списка абонентов выступает поле типа TextList (набор надписей). У такого поля должна стоять отметка **Разрешить несколько значений** (см. рис. 20.1).

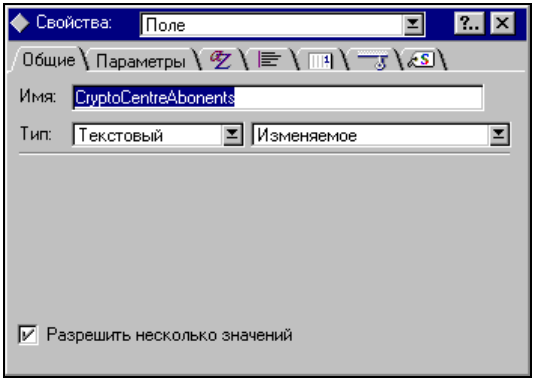

Рис. 20.1. Поле `CryptoCentreAbonents`

- Поле `CryptoCentreAbonents` задается пользователем-изготовителем документа. В качестве имен пользователей в списке достаточно задать начало одной из записей поля `FullName` одной из адресных книг пользователя. Если такое начало относится к нескольким записям, то используется открытый ключ первой из найденных записей.
- Для каждой записи поля `CryptoCentreAbonents` в одной из адресных книг пользователя должен быть открытый ключ системы Криптоцентр (запись в поле `CryptoCentrePublicKey`).
- Следующее необходимое поле — это текстовое поле `CryptoCentreFrom`. В него записывается идентификатор ключа пользователя-изготовителя документа. Это поле необходимо для организации поиска открытого ключа у получателя документа.

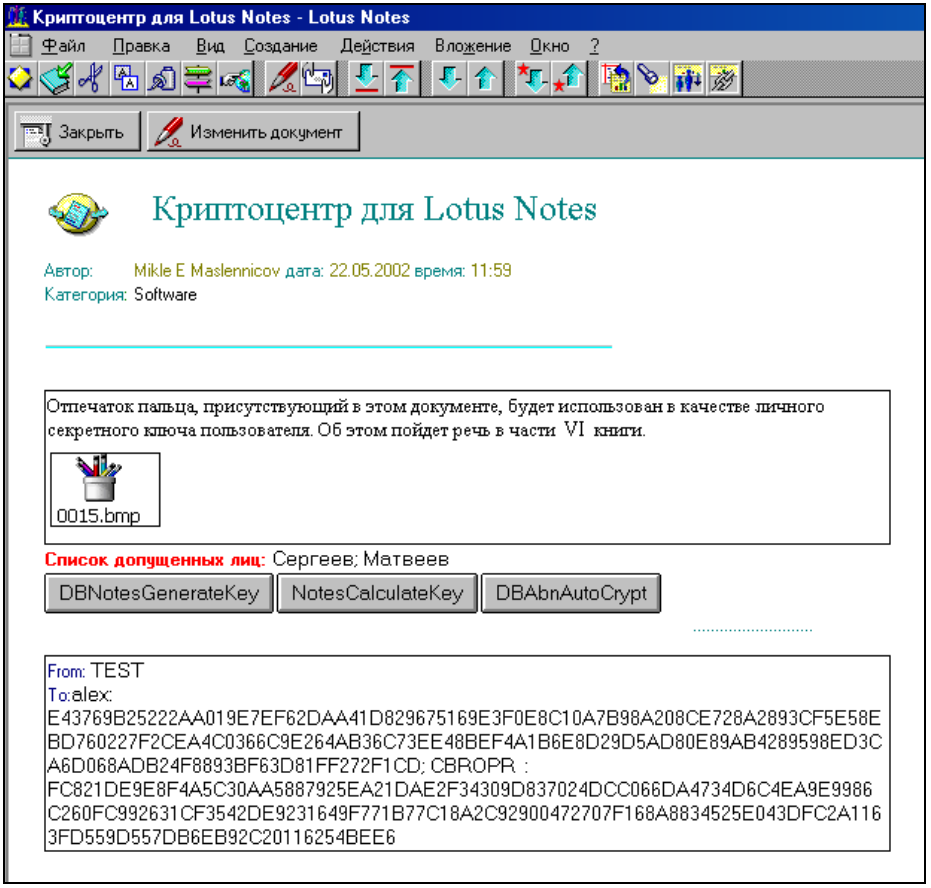

Рис. 20.2. Поле, шифруемое по списку, до зашифровывания

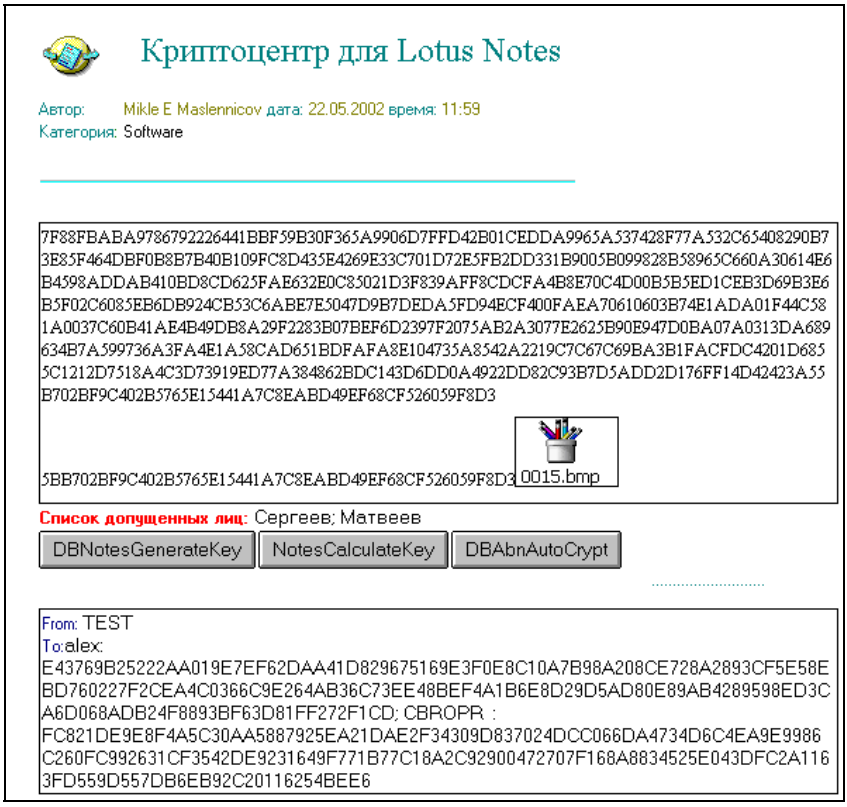

Рис. 20.3. Поле, шифруемое по списку, после зашифровывания

- И, наконец, последнее необходимое поле — это `CryptoCentreTo`. Это также поле типа `TextList`, у него должна стоять отметка **Разрешить несколько значений**, в него будет записываться зашифрованный на различных общих ключах секретный ключ для шифрования по списку. Вот пример документа, содержащего поле, зашифрованное по списку.

На рис. 20.2 и 20.3 мы видим три кнопки: `DBNotesGenerateKey`, `NotesCalculateKey` и `DBAbnAutoCrypt`. Рассмотрим их назначение.

Для зашифровывания по списку необходимо выработать случайный секретный ключ. Он сам, в свою очередь, шифруется с использованием системы с открытым распределением ключей и записывается в поле `CryptoCentreTo`, что осуществляется с помощью кнопки `DBNotesGenerateKey` и видно на рис. 20.2 и 20.3. Однако в некоторый момент пользователю-исполнителю может потребоваться расшифровать зашифрованный ранее текст и посмотреть или изменить его содержание. В этом случае он может использовать процедуру *вычисления* записанного в таком виде секретного ключа, т. е. провести его расшифрование на одном из общих ключей списка абонентов. Эти

функции выполняются с помощью кнопки `NotesCalculateKey`. С ее помощью можно проверить, что необходимый для зашифровывания/расшифрования секретный ключ вычисляется.

Сама операция зашифровывания/расшифрования осуществляется с помощью кнопки `DBAbnAutoCrypt`. В нее также встроена процедура вычисления секретного ключа.

Ниже на рис. 20.4–20.9 приведена реализация этих кнопок на языке LotusScript:

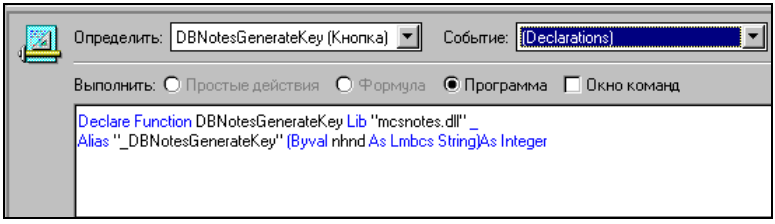

Рис. 20.4. Кнопка `DBNotesGenerateKey`. Декларация

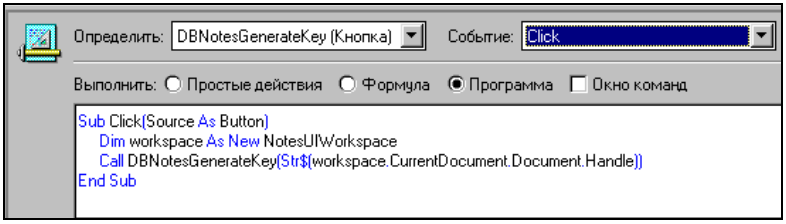

Рис. 20.5. Кнопка `DBNotesGenerateKey`. Нажатие

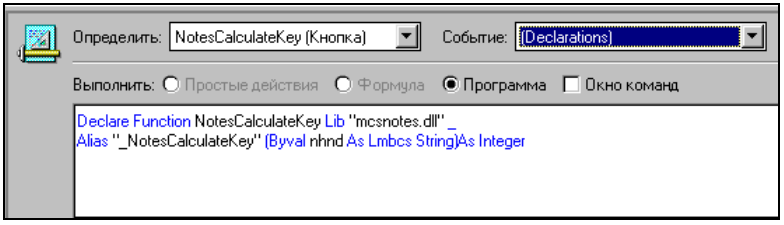

Рис. 20.6. Кнопка `NotesCalculateKey`. Декларация

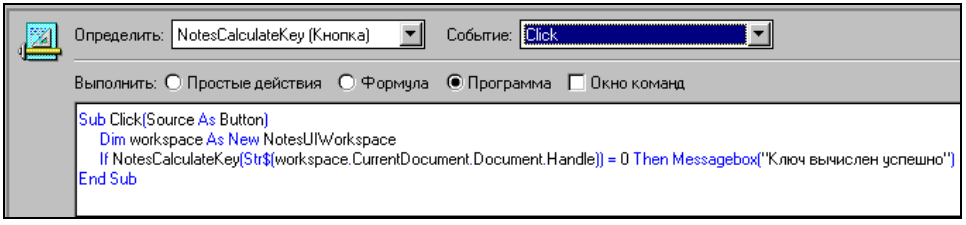

Рис. 20.7. Кнопка `NotesCalculateKey`. Нажатие

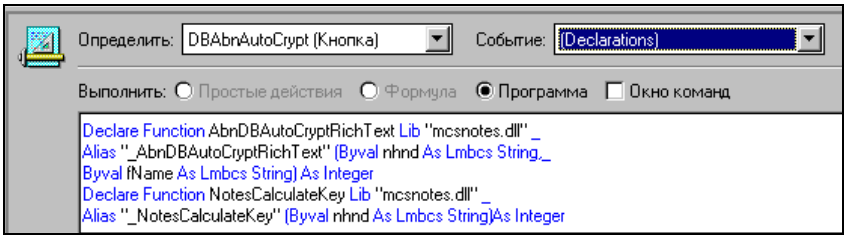

Рис. 20.8. Кнопка `DBAbnAutoCrypt`. Декларация

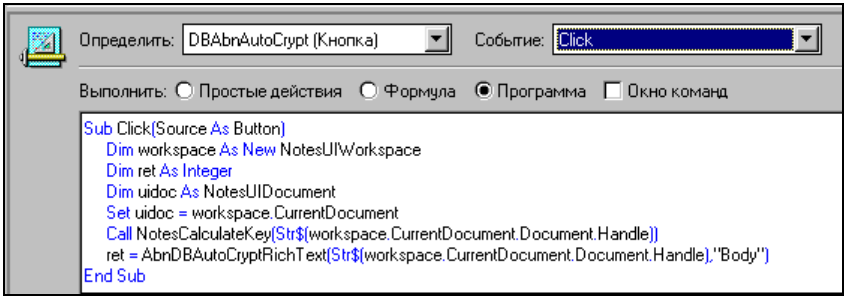

Рис. 20.9. Кнопка `DBAbnAutoCrypt`. Нажатие

Вот листинг процедуры `NotesCalculateKey`.

Листинг 20.2. Процедура `NotesCalculateKey`

```
extern "C" __declspec(dllexport) int NotesCalculateKey(PChar nIDstr)
{
  TRegistry * Reg;
  char *a;
  char buff[1024];
  unsigned char pi[512];
  int i;
  WORD NAbn;
  WORD AbnLen;
  short j1;
  unsigned char publ_crypt_key1[512];

  // Проверка наличия в ОП секретного ключа пользователя
  memset(buff,0,128);
  if(memcmp((char *)crypt_key,buff,80)==0)
  {
    MessageBox(0,"Не введен секретный ключ системы
```

```
    Криптоцентр","Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }

  sscanf(nIDstr,"%d",&note_handle);

  memset(publ_crypt_key,0,80);

  if(NSFItemIsPresent(note_handle,CRYPTOCENTRE_FROM,
  (WORD) strlen (CRYPTOCENTRE_FROM))==false)
  {
    MessageBox(0,"Не найден отправитель конфиденциального
    сообщения","Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }
  if(NSFItemIsPresent(note_handle,CRYPTOCENTRE_TO,
  (WORD) strlen (CRYPTOCENTRE_TO))==false)
  {
no_abn:
    MessageBox(0,"Не найдены получатели конфиденциального
    сообщения","Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }

  // Список получателей
  NAbn = NSFItemGetTextListEntries(note_handle,CRYPTOCENTRE_TO);
  if(NAbn ==0)
    goto no_abn;
  // Открытый ключ менеджера СБ нужен
  // для проверки сертификатов открытых ключей
  Reg = new TRegistry();
  memset(keyman,0,128);
  Reg->RootKey = HKEY_CURRENT_USER;
  if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\CryptoCentre\\KeyMan",\
  false)==true)
    strcpy(keyman,Reg->ReadString("Файл").c_str());
  delete Reg;

  // Отправитель
```

```
memset(AbnName,0,512);
NSFItemGetText(note_handle,CRYPTOCENTRE_FROM,AbnName,
(WORD) sizeof (AbnName));

// Расшифрование, когда отправитель — сам пользователь
if(memicmp(key_label,AbnName,strlen(AbnName))==0)
{
  j1=0;
  AbnLen=0;
  while(AbnLen==0&&j1<NAbn)
  {
    memset(AbnName,0,512);
    AbnLen = NSFItemGetTextListEntry(note_handle,
    CRYPTOCENTRE_TO,j1,AbnName,(WORD)sizeof(AbnName));
    if(strrchr(AbnName,':')==NULL)
      AbnLen=0;
    j1++;
  }
  if(AbnLen==0)
    goto no_abn;
  memset(publ_crypt_key1,0,512);
  memcpy(publ_crypt_key1,strrchr(AbnName,':')+2,176);
  memset(strrchr(AbnName,':'),0,1);

} // Расшифрование, когда отправитель — сам пользователь

// Когда отправитель — другой абонент
else
{
  j1=0;
  AbnLen=0;

  while(j1<NAbn)
  {
    memset(AbnName+16,0,256);
    AbnLen = NSFItemGetTextListEntry(note_handle,
             CRYPTOCENTRE_TO,j1,AbnName+16,256);
    j1++;
    if(AbnLen==0 || strrchr(AbnName+16,':')==NULL)
```

```
        continue;
      if(memicmp(AbnName+16,key_label,strlen(key_label))==0)
        break;
    }

    if(memicmp(AbnName+16,key_label,strlen(key_label))!=0)
    {
      sprintf(buff,\
      "Абонент %s к данному конфиденциальному сообщению \
      не допущен",key_label);
      MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
      return(-1);
    }
    memset(publ_crypt_key1,0,512);
    memcpy(publ_crypt_key1,strrchr(AbnName+16,':')+2,176);

  } // Когда отправитель — другой абонент

  memset(any_pkey,0,64);
  sert=false;
  memset(IDKey,0,16);
  memset(abonent,0,16);
  strcpy(abonent,AbnName);

  // Поиск и подготовка открытого ключа
  if(PrepareCryptPKey()!=0)
  {
no_public_key:
    sprintf(buff,"Открытый ключ абонента %s не найден",AbnName);
    MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }
  memset(buff,0,128);
  if(memcmp((char *)any_pkey,buff,64)==0)
    goto no_public_key;

  if(sert==false)
  {
    sprintf(buff,"Открытый ключ абонента %s не имеет сертификата.
```

```
    Продолжить?",AbnName);
    if(MessageBox(0,buff,"Криптоцентр",\
    MB_ICONWARNING | MB_YESNO)!=IDYES)
      return(-1);
  }
  // Вычисление общего ключа
  prepear_com_key(any_pkey,crypt_key1,pi,pi+256);

  // Расшифрование секретного ключа для шифрования по списку
  i=DecryptMem(NULL,publ_crypt_key1,176,pi,pi+256);
  if(i!=0)
  {
err_key:
    sprintf(buff,"Ошибка при восстановлении разового ключа");
    MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }

  // Для проверки
  srand(1813);
  for(i=0;i<8;i++)
    buff[i]=(unsigned char)rand();
  if(memcmp((unsigned char *)publ_crypt_key1+56,buff,8)!=0)
    goto err_key;
  memcpy((unsigned char *)publ_crypt_key,publ_crypt_key1,64);

  return(0);
}
```

Из листинга 20.2 видно, что процедура `NotesCalculateKey` достаточно универсальная: она применяется как самим пользователем-изготовителем документа для расшифрования собственно зашифрованного документа, так и для его расшифрования получателем. Все различие заключается в способе построения общего ключа и в выборе записи в поле `CryptoCentreTo`: пользователь-изготовитель выбирает первую запись в этом поле, а получатель находит в нем предназначенную для него уникальную запись.

И в заключение этой главы отметим одну особенность реализации процедуры выработки секретного ключа для шифрования по списку.

Результатом такой выработки является заполнение поля `CryptoCentreTo`. Что с ним делать? Есть два варианта: записать непосредственно в базу данных

или просто отобразить на экране. Из прошлой главы мы помним о тех проблемах, которые возникали при организации процедуры шифрования различных типов полей в документах Notes. Текстовые поля после зашифровывания/расшифрования легко отображаются на экране, а для полей типа RichText результат зашифровывания/расшифрования записывается в базу данных без отображения на экране. Если после осуществления криптографических операций с такими полями пользователь выполнит операцию сохранения, то содержимое экрана, на котором не отобразился результат зашифровывания/расшифрования, восстановит в базе данных первоначальные значения поля `RichText`. Эта проблема может быть разрешена с помощью всяческих ухищрений (закрытие-открытие документа средствами LotusScript) либо путем использования подобных криптографических операций в различных Notes-агентах, когда отображения на экране еще нет.

Как уже отмечалось ранее, шифрование по списку отличается от шифрования с использованием ключевой книги только способом подготовки секретного ключа. Все остальные алгоритмы, в том числе и алгоритм шифрования `RichText` полей, остаются без изменения. Следовательно, остается актуальной проблема отображения результатов зашифровывания/расшифрования `RichText` полей. Но может потребоваться шифрование и обычных текстовых полей, результат зашифровывания которых сразу же отображается на экране. Для таких случаев целесообразно иметь две различных процедуры выработки секретного ключа для шифрования по списку: одна процедура заносит результат непосредственно в базу данных Notes, а другая — только отображает на экране. Кстати говоря, аналогично можно сделать и процедуры шифрования текстовых полей: одна выдает результат только на экран, а другая — заносит только в Notes-БД. Здесь и далее для процедур, осуществляющих запись результата только в Notes-БД, в их названии мы будем в начале употреблять символы "DB", а если такие символы отсутствуют, то будем полагать, что результат отображается только на экране.

На рис. 20.10 и 20.11 приведен пример реализации кнопки `NotesGenerateKey`, осуществляющей выработку секретного ключа для шифрования по списку и выдачу результата только на экран.

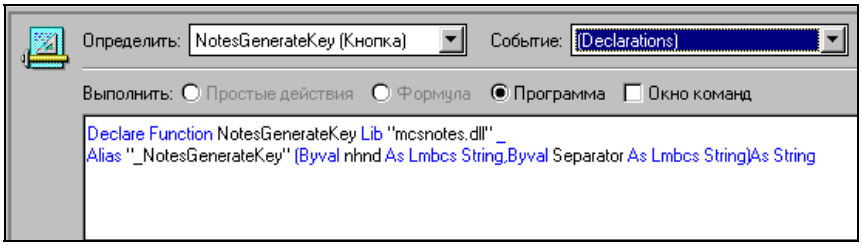

Рис. 20.10. Кнопка `NotesGenerateKey`. Декларация

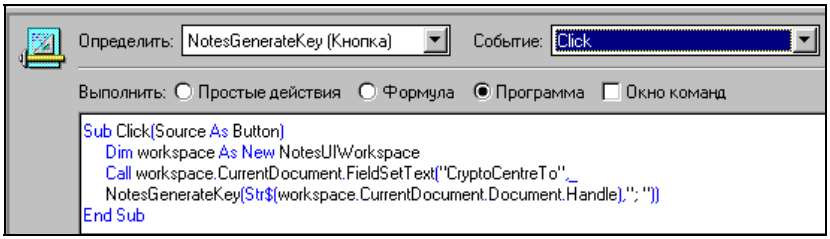

Рис. 20.11. Кнопка `NotesGenerateKey`. Нажатие

Шифрование по списку — это пример организации закрытой почты в Lotus Notes. Письмо в Notes — это документ в одной специализированной Notes-БД. Пользователь-отправитель определяет в нем круг лиц, допущенных к конфиденциальной части письма, и осуществляет шифрование по списку. Тех проблем, которые встречались в Microsoft Outlook (сбор свойств в один файл, его зашифровывание, формирование нового сообщения с этим файлом в качестве вложения), здесь нет.

Глава 21

Организация электронной подписи

Проблемы, связанные с организацией электронной подписи Notes-документов, очень похожи на аналогичные проблемы подписи сообщений в Microsoft Outlook: необходимо задать хэш-вектор, определяющий информацию, защищаемую подписью, вычислить по хэш-вектору хэш-функцию, каким-то образом включить хэш-вектор в саму подпись. Операция вычисления (или проверки) самой подписи при известной хэш-функции — стандартная, о ней мы подробно говорили в *части II* книги.

Очевидно, что под саму подпись в Notes-документах целесообразно выделить отдельное поле. Назовем его `CryptoCentreSigns`. Тип данных этого поля — `TextList` (набор надписей). В нем могут содержаться несколько подписей различных абонентов. Поле `CryptoCentreSigns` может быть отображено на экран. В этом случае в свойствах поля необходимо установить флажок **Разрешить несколько значений**, иначе все подписи будут восприниматься как одна и система проверки нескольких подписей работать не будет! Каждая подпись состоит из следующих компонентов, приведенных ниже в табл. 21.1.

Таблица 21.1. *Компоненты подписи в Notes-документе*

Смещение	Длина	Компонент	Значение
0	8	`KeyLabel`	Идентификатор ключа абонента, осуществившего подпись
8	4	`SignSize`	Длина подписи в HEX-формате
12	22	`SignDateTime`	Время осуществления подписи в формате DD.MM.YY HH:MM:SS.HH
34	192	`Sign`	Цифровая часть подписи в HEX-формате
226	—	`HashVector`	Перечень полей, защищаемых подписью

Рассмотрим на примере (см. рис. 21.1) операцию осуществления электронной подписи в Notes-документе.

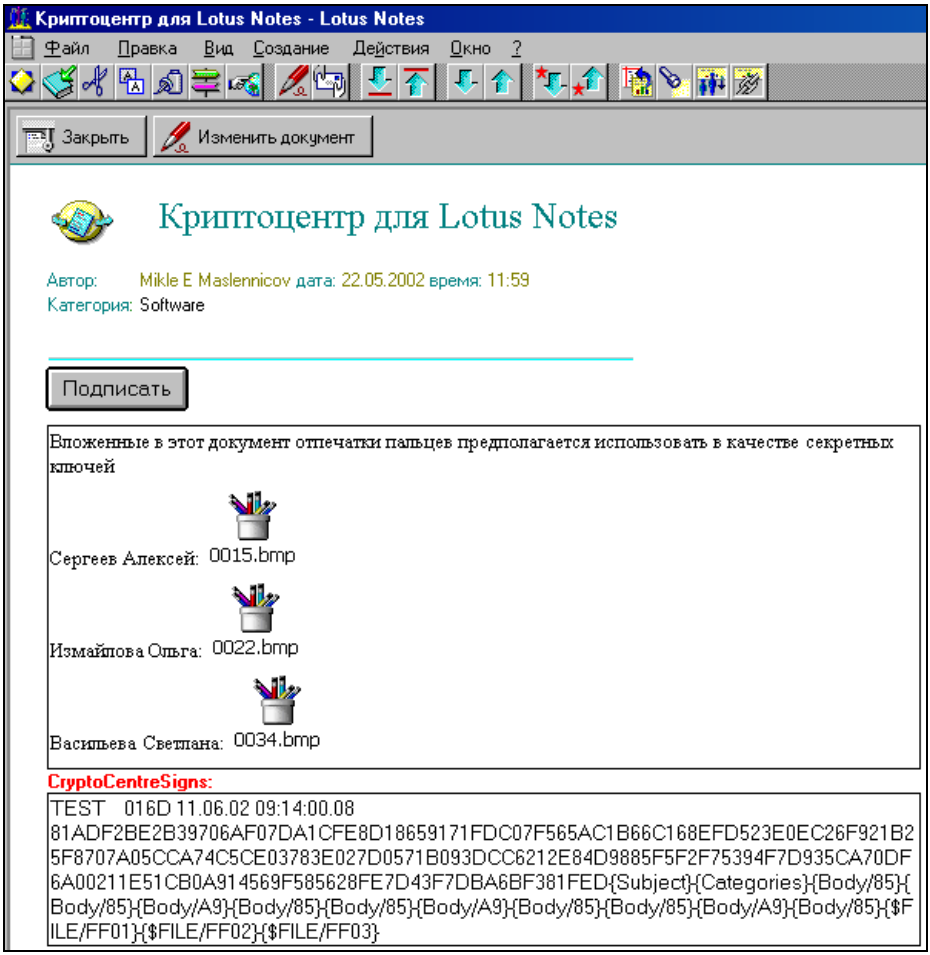

Рис. 21.1. Пример электронной подписи Notes-документа

Подпись была осуществлена с помощью кнопки **Подписать**. На рис. 21.2 и 21.3 приводится реализация этой кнопки на языке LotusScript.

Первый же взгляд на эти рисунки показывает отличие системы электронной подписи Notes-документов от подписи сообщений Microsoft Outlook: здесь хэш-вектор фактически задается программно, путем расположения соответствующих процедур в программе LotusScript. Давайте подробнее остановимся на реализации кнопки **Подписать**.

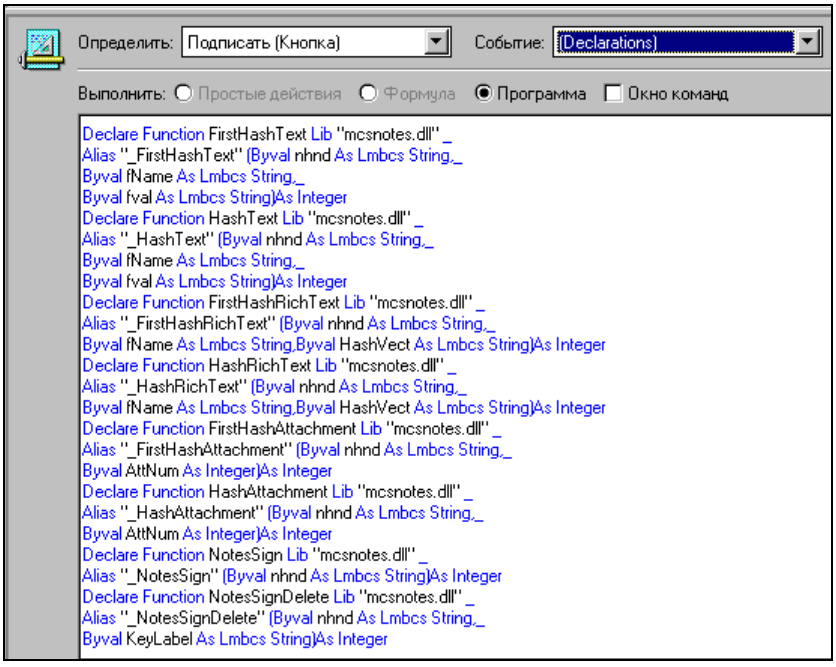

Рис. 21.2. Кнопка **Подписать**. Декларация

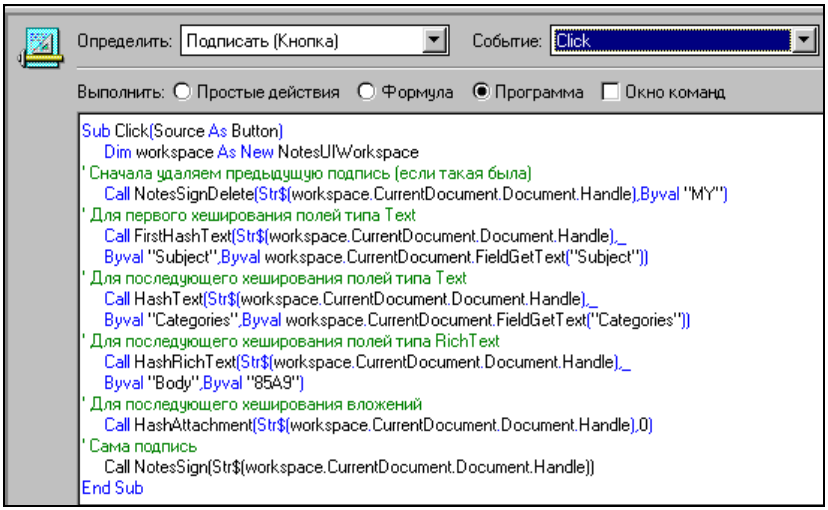

Рис. 21.2. Кнопка **Подписать**. Нажатие

Первая выполняемая процедура — `NotesSignDelete`. Ее задача — удалить из поля `CryptoCentreSigns` предыдущую подпись, если такая там имелась. Параметр `MY` указывает на то, что необходимо удалить подпись того абонента,

секретный ключ которого находится в настоящий момент в оперативной памяти. Вот листинг программной реализации процедуры `NotesSignDelete`.

Листинг 21.1. Процедура `NotesSignDelete`

```
extern "C" __declspec(dllexport) int NotesSignDelete(PChar nIDstr,char *
KeyLabel)
{
  STATUS        error;
  char tmpname[256];
  FILE *fl;
  char buff[512];
  TRegistry * Reg;
  char Signs[30000];
  WORD NSigns;
  short j;
  WORD SignLen;

  sscanf(nIDstr,"%d",&note_handle);
  NSigns = NSFItemGetTextListEntries(note_handle,\
           CRYPTOCENTRE_SIGN_FIELD);
  // При параметре "ALL" удаляются все подписи
  if(NSigns ==0||stricmp(KeyLabel,"ALL")==0)
  {
    error =NSFItemDelete(note_handle,\
    CRYPTOCENTRE_SIGN_FIELD,strlen(CRYPTOCENTRE_SIGN_FIELD));
    if(error)
      return(-1);
    NSFNoteUpdate(note_handle,0);
    return(0);
  }

  // При параметре "MY" удаляется только подпись того,
  // кто сейчас работает
  if(stricmp(KeyLabel,"MY")==0)
  {
    memset(buff,0,128);
    if(memcmp((char *)sign_key,buff,80)==0)
    {
```

```
      MessageBox(0,"Не введен секретный ключ системы
      Криптоцентр","Криптоцентр",MB_ICONERROR | MB_OK);
      return(-1);
    }
    // Проверка сохранности секретного ключа в ОП
    if(CheckKey(sign_key)!=0)
    {
err_key:
      MessageBox(0,"Ошибка в секретном ключе!",\
      "Криптоцентр",MB_ICONERROR | MB_OK);
      return(-1);
    }

    // Временный файл для других подписей
    // Имя файла — Notes.TMP,
    // каталог указанный в реестре Windows
    strcpy(tmpname,"C:\\CENTRE\\TMP");
    Reg = new TRegistry();
    Reg->RootKey = HKEY_CURRENT_USER;
    if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\CryptoCentre",false)==true)
      if(Reg->ValueExists("TmpDir")==true)
        strcpy(tmpname,Reg->ReadString("TmpDir").c_str());
    delete Reg;

    strcat(tmpname,"\\NOTES.TMP");
    if(make_new_file(tmpname)!=0)
    {
err1:
      sprintf(buff,"Ошибка при удалении подписи %s",key_label);
      MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
      return(-1);
    }
    // Переписываем все чужие подписи во временный файл
    if((fl=fopen(tmpname,"w+b"))==NULL)
      goto err1;
    for(j=0;j<NSigns;j++)
    {
      memset(Signs,0,30000);
      SignLen = NSFItemGetTextListEntry(note_handle,\
```

```
  CRYPTOCENTRE_SIGN_FIELD,j,Signs,(WORD)sizeof(Signs));
  if(memicmp(Signs,key_label,strlen(key_label))!=0&&\
  SignLen!=0)
  {
    fwrite(&SignLen,2,1,fl);
    fwrite(Signs,SignLen,1,fl);
  }
}

// Удаляем все поле с подписями
error = NSFItemDelete(note_handle,\
        CRYPTOCENTRE_SIGN_FIELD,\
        strlen(CRYPTOCENTRE_SIGN_FIELD));
if(error)
{
  fclose(fl);
  goto err1;
}

// Восстанавливаем сохраненные во временном файле подписи
fseek(fl,0L,SEEK_SET);
while(!feof(fl))
{
  SignLen=0;
  fread(&SignLen,2,1,fl);
  if(SignLen!=0)
  {
    memset(Signs,0,30000);
    fread(Signs,SignLen,1,fl);
    error = NSFItemAppendTextList(note_handle,\
            CRYPTOCENTRE_SIGN_FIELD,Signs,\
            strlen(Signs),FALSE);
    if(error)
    {
      fclose(fl);
      goto err1;
    }
  }
}
```

```
    fclose(fl);
    // Удаляем временный файл
    unlink(tmpname);
    // Обновляем Notes-документ
    NSFNoteUpdate(note_handle,0);
    return(0);
  }// Конец обработки параметра "MY"

  MessageBox(0,"Неверно задан параметр удаления
  подписей","Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}
```

Приступим теперь к хэшированию. На рис. 21.2 видно, что в перечне функций, используемых в процессе хэширования, есть функции с приставкой `First` и просто функции, без этой приставки. В чем их различие?

Дело в том, что при вычислении хэш-функции Notes-документа мы используем тот же принцип, что и при проведении аналогичной процедуры для сообщений в хранилищах MAPI *(см. главу 13)*. Там для каждого свойства вычислялась его персональная хэш-функция, затем значения всех этих хэш-функций объединялись в один массив и вычислялась хэш-функция всего полученного массива. Здесь — аналогичный принцип. Разница между функциями с приставкой `First` и без нее заключается в том, что первая обнуляет этот массив, а вторая просто дописывает в него вычисленное хэш-значение. Вот листинг наиболее простых процедур — `FirstHashText` и просто `HashText`.

Листинг 21.2. Хэширование текстовых полей

```
extern "C" __declspec(dllexport) int FirstHashText(PChar nIDstr,PChar
FName,PChar InputText)
{
  // Обнуление хэш-массива
  memset(HashNotes,0,64000);
  // Указатель в хэш-массиве
  NHash = 0;
  return(HashText(nIDstr,FName,InputText));
}


extern "C" __declspec(dllexport) int HashText(PChar nIDstr,PChar
FName,PChar InputText)
{
```

```
union HESH
{
  unsigned short h[32];
  unsigned char h1[64];
} hesh;

char Signs[30000];
WORD NSigns;
short i,j,j1;
WORD SignLen;
char buff[512];

// Если хэшируемое поле не является полем с предыдущими подписями
if(stricmp(FName,CRYPTOCENTRE_SIGN_FIELD)!=0)
{
  HeshMem(InputText,strlen(InputText),hesh.h1);
  strcpy(HashNotes+NHash,FName);
  memcpy(HashNotes+NHash+64,hesh.h1,64);
  NHash+=128;
}

// Если это предыдущие подписи
else
{
  sscanf(nIDstr,"%d",&note_handle);
  NSigns = NSFItemGetTextListEntries(note_handle,\
           CRYPTOCENTRE_SIGN_FIELD);
  if(NSigns ==0)
    return(0);
  memset(buff,0,128);
  if(memcmp((char *)sign_key,buff,80)==0)
  {
    MessageBox(0,"Не введен секретный ключ системы
    Криптоцентр","Криптоцентр",MB_ICONERROR | MB_OK);
    return(-1);
  }
  if(CheckKey(sign_key)!=0)
  {
    MessageBox(0,"Ошибка в секретном ключе!",\
```

```
      "Криптоцентр",MB_ICONERROR | MB_OK);
      return(-1);
    }
    i=0;
    for(j=0;j<NSigns;j++)
    {
      memset(Signs,0,30000);
      SignLen = NSFItemGetTextListEntry(note_handle,\
                CRYPTOCENTRE_SIGN_FIELD,j,Signs,\
                (WORD)sizeof(Signs));
      if(memicmp(Signs,key_label,strlen(key_label))==0)
      {
        sprintf(buff,"Среди предыдущих подписей\
        обнаружена более ранняя подпись %s,\n\
        которая в дальнейшем будет заменена и не\
        учитывается при хэшировании",key_label);
        MessageBox(0,buff,"Криптоцентр",\
        MB_ICONWARNING | MB_OK);
      }
      if(memicmp(Signs,key_label,\
      strlen(key_label))!=0&&SignLen!=0)
      {
        i++;
        HeshMem(Signs,strlen(Signs),hesh.h1);
        strcpy(HashNotes+NHash,CRYPTOCENTRE_SIGN_FIELD);
        HashNotes[NHash+\
        strlen(CRYPTOCENTRE_SIGN_FIELD)+1]=i;
        memcpy(HashNotes+NHash+64,hesh.h1,64);
        NHash+=128;
      }
    }
  } // Если это предыдущие подписи
  return(0);
}
```

Как и при организации процедур шифрования, наибольшую сложность при подписи представляют `RichText`-поля. В *главе 19* мы уже упоминали о том, что `RichText`-поле подобно контейнеру, в котором могут быть записаны различные свойства. Некоторые из этих свойств целесообразно защитить

электронной подписью. Приведем перечень свойств, которые могут находиться в `RichText`-контейнере.

```
SIG_CD_DOCUMENT_PRE_26    80
SIG_CD_PARAGRAPH          81
SIG_CD_PABDEFINITION      82
SIG_CD_PABREFERENCE       83
SIG_CD_FIELD_PRE_36       84
SIG_CD_TEXT               85
SIG_CD_DOCUMENT           86
SIG_CD_METAFILE           87
SIG_CD_BITMAP             88
SIG_CD_FIELD              8A
SIG_CD_FONTTABLE          8B
SIG_CD_LINK               8C
SIG_CD_LINKEXPORT         8D
SIG_CD_HEADER             8E
SIG_CD_KEYWORD            8F
SIG_CD_LINK2              91
SIG_CD_LINKEXPORT2        92
SIG_CD_CGM                93
SIG_CD_TIFF               94
SIG_CD_BITMAPHEADER       95
SIG_CD_BITMAPSEGMENT      96
SIG_CD_COLORTABLE         97
SIG_CD_PATTERNTABLE       98
SIG_CD_GRAPHIC            99
SIG_CD_PMMETASEG          9A
SIG_CD_WINMETASEG         9B
SIG_CD_MACMETASEG         9C
SIG_CD_CGMMETA            9D
SIG_CD_PMMETAHEADER       9E
SIG_CD_WINMETAHEADER      9F
SIG_CD_MACMETAHEADER      A0
SIG_CD_DDEBEGIN           A1
SIG_CD_DDEEND             A2
SIG_CD_TABLEBEGIN         A3
SIG_CD_TABLECELL          A4
SIG_CD_TABLEEND           A5
```

```
SIG_CD_STYLENAME          A6
SIG_CD_OLEBEGIN           A7
SIG_CD_OLEEND             A8
SIG_CD_HOTSPOTBEGIN       A9
SIG_CD_HOTSPOTEND         AA
SIG_CD_BUTTON             AB
SIG_CD_BAR                AC
SIG_CD_V4HOTSPOTBEGIN     AD
SIG_CD_V4HOTSPOTEND       AE
SIG_CD_EXT_FIELD          B0
SIG_CD_LSOBJECT           B1
SIG_CD_HTMLHEADER         B2
SIG_CD_HTMLSEGMENT        B3
SIG_CD_LAYOUT             B7
SIG_CD_LAYOUTTEXT         B8
SIG_CD_LAYOUTEND          B9
SIG_CD_LAYOUTFIELD        BA
SIG_CD_PABHIDE            BB
SIG_CD_PABFORMREF         BC
SIG_CD_ACTIONBAR          BD
SIG_CD_ACTION             BE
SIG_CD_DOCAUTOLAUNCH      BF
SIG_CD_LAYOUTGRAPHIC      C0
SIG_CD_OLEOBJINFO         C1
SIG_CD_LAYOUTBUTTON       C2
SIG_CD_TEXTEFFECT         C3
SIG_CD_STORAGELINK        C4
SIG_CD_TRANSPARENTTABLE   C5
SIG_CD_HORIZONTALRULE     C9
SIG_CD_ALTTEXT            CA
SIG_CD_ANCHOR             CB
SIG_CD_HTMLBEGIN          CC
SIG_CD_HTMLEND            CD
SIG_CD_HTMLFORMULA        CE
```

Справа приведен шестнадцатеричный код свойства.

Задавая операцию хэширования `RichText`-контейнера, мы должны определить те свойства, которые будут подвергнуты хэшированию. Среди свойств `RichText`-контейнера, как это видно из приведенного выше списка, достаточно много служебных и вспомогательных свойств, которые вряд ли целе-

сообразно подвергать хэшированию. В приведенном ранее примере кнопки **Подписать** хэшированию подвергаются только два свойства: 85 — текст и A9 — атрибуты файла-вложения. Вот листинг программной реализации процедуры хэширования `RichText`-контейнера.

Листинг 21.3. Хэширование `RichText`-контейнера

```
extern "C" __declspec(dllexport) int HashRichText(PChar nIDstr,PChar
FName,PChar HashVect)
{
  STATUS        error;
  BLOCKID item_blockid;
  WORD  wDataType;
 BLOCKID  bidRichText;
  DWORD  dwItemSize;

// Сканирование контейнера
  strcpy(FldName,FName);
  sscanf(nIDstr,"%d",&note_handle);
  error = NSFItemInfo(note_handle,FName,strlen(FName),\
          &item_blockid,&wDataType,&bidRichText,&dwItemSize);
  if(error==ERR_ITEM_NOT_FOUND)return(0);
    error = EnumCompositeBuffer(bidRichText,dwItemSize,\
            &HashFields,HashVect);
  return(error);
}

// Функция действия с найденной записью
STATUS LNCALLBACK HashFields(char *RecordPtr,WORD RecordType,DWORD
RecordLength,void *pCtx)
{
  unsigned char HashRecType;
  int i,j;
  union HESH
  {
    unsigned short h[32];
    unsigned char h1[64];
  } hesh;

  if(RecordLength==0)return(NOERROR);
```

```
  i=0;
  while(i<strlen((char *)pCtx))
  {
    sscanf((char *)pCtx+i,"%02X",&j);
    HashRecType=j;
    if(HashRecType==(unsigned char)RecordType)
    {
      HeshMem(RecordPtr,RecordLength,hesh.h1);
      strcpy(HashNotes+NHash,FldName);
      HashNotes[NHash+strlen(FldName)+1]=HashRecType;
      memcpy(HashNotes+NHash+64,hesh.h1,64);
      NHash+=128;
    }
    i+=2;
  }
  return(NOERROR);
}
```

С используемым здесь приемом мы уже встречались ранее в предыдущих главах, посвященных Lotus Notes. Сканируется контейнер, в качестве параметра при сканировании выступает функция действия, определяющая процедуры, которые необходимо осуществить с найденным объектом.

И, наконец, последняя функция, относящаяся к хэшированию — хэширование файлов-вложений. Она похожа на аналогичные функции, описанные в главах, посвященных шифрованию.

Листинг 21.4. Хэширование файлов-вложений

```
extern "C" __declspec(dllexport) int HashAttachment(PChar nIDstr,short AttNum)
{
  STATUS        error;
  TRegistry * Reg;
  char tmpname[256]; // временный файл для вложения
  unsigned short AttCur;
  unsigned short AttAll;
  BLOCKID item_blockid;
  WORD  wDataType;
  BLOCKID  bidAttach;
  DWORD  dwItemSize;
  int i;
```

```
unsigned short j;
char buff[512];
int ret=0;


// Временный файл для вложения
// В каталоге \\TMP создаем временный файл
strcpy(tmpname,"C:\\CENTRE\\TMP");

Reg = new TRegistry();
Reg->RootKey = HKEY_CURRENT_USER;
if(Reg->OpenKey("\\SOFTWARE\\AlbaSoft\\CryptoCentre",false)==true)
  if(Reg->ValueExists("TmpDir")==true)
    strcpy(tmpname,Reg->ReadString("TmpDir").c_str());
delete Reg;

strcat(tmpname,"\\NOTES.TMP");
if(make_new_file(tmpname)!=0)
{
  MessageBox(0,"Ошибка при создании временного\
  файла","Криптоцентр",MB_ICONERROR | MB_OK);
  return(-1);
}

// Если одно вложение
if(AttNum!=0)
{
  AttCur = 0;
  error = NSFItemInfo(note_handle,\
  ITEM_NAME_ATTACHMENT,strlen(ITEM_NAME_ATTACHMENT),\
  &item_blockid,&wDataType,&bidAttach,&dwItemSize);
  while(error!=ERR_ITEM_NOT_FOUND)
  {
    AttCur++;
    if(AttNum!=AttCur)
      goto next;
    i=HashOneNotesAttachment(note_handle,\
      item_blockid,tmpname,AttCur);
    if(i!=0)
```

```
      {
        sprintf(buff,\
        "Ошибка при обработке вложения № %d",AttCur);
        MessageBox(0,buff,"Криптоцентр",\
        MB_ICONERROR | MB_OK);
        ret=-1;
      }
      break;
next:
      error = NSFItemInfoNext(note_handle,item_blockid,\
      ITEM_NAME_ATTACHMENT,strlen(ITEM_NAME_ATTACHMENT),\
      &item_blockid,&wDataType,&bidAttach,&dwItemSize);
    }
    if(error==ERR_ITEM_NOT_FOUND)
    {
      sprintf(buff,"Вложение № %d не найдено",AttNum);
      MessageBox(0,buff,"Криптоцентр",MB_ICONERROR | MB_OK);
      ret=-1;
      goto stop;
    }
  } // if(AttNum!=0)

  // Если все вложения
  else
  {
    AttCur=0;
    error = NSFItemInfo(note_handle,ITEM_NAME_ATTACHMENT,\
    strlen(ITEM_NAME_ATTACHMENT),&item_blockid,&wDataType,&bidAttach,\
    &dwItemSize);
    while(error!=ERR_ITEM_NOT_FOUND)
    {
      AttCur++;
      i=HashOneNotesAttachment(note_handle,item_blockid,tmpname,\
      AttCur);
      if(i!=0)
      {
err1:
        sprintf(buff,\
        "Ошибка при обработке вложения № %d",j+1);
```

```
        MessageBox(0,buff,"Криптоцентр",\
        MB_ICONERROR | MB_OK);
        ret=-1;
        goto stop;
      }
      error = NSFItemInfoNext(note_handle,item_blockid,\
      ITEM_NAME_ATTACHMENT,strlen(ITEM_NAME_ATTACHMENT),\
      &item_blockid,&wDataType,&bidAttach,&dwItemSize);
    } // while(error!=ERR_ITEM_NOT_FOUND)
  } // if(AttNum==0)
stop:
  unlink(tmpname);
  return(ret);
}

int HashOneNotesAttachment(NOTEHANDLE note_handle,BLOCKID item_blockid,
char *tmpname,unsigned short AttNum)
{
  FILE *fl;
  long dlina;
  STATUS        error;
  union HESH
  {
    unsigned short h[32];
    unsigned char h1[64];
  } hesh;
  float empt_f;
  char buff[512];
  int j;

  error = NSFNoteExtractFile(note_handle,item_blockid,tmpname,NULL);
  if(error)return(-1);
    strcpy(filename,tmpname);
  fl=fopen(filename,"rb");
  dlina = filelength(fileno(fl));
  fclose(fl);
  // Форма для отображения хода процесса хэширования
  FormHash = new TFormHash(NULL);
  FormHash->Label5->Visible = true;
```

```
  FormHash->Caption = "ХЭШИРОВАНИЕ";
  FormHash->BitBtn1->Visible = false;
  FormHash->BitBtn1->Enabled = false;

  sprintf(buff,"Вложение %d",AttNum);
  FormHash->Label2->Caption = buff;
  FormHash->Label7->Caption = "";

  empt_f=dlina;
  sprintf(buff,"%.3f Кб",empt_f/1000);
  FormHash->Label4->Caption = buff;
  FormHash->Label5->Visible = false;

  FormHash->Refresh();
  FormHash->Show();

  // Внимание! Здесь файл всегда хэшируется целиком,
  // вне зависимости от наличия в нем подписей!

  if(HeshFile(0,hesh.h1)!=0)return(-1);
    FormHash->Hide();
  delete FormHash;

  strcpy(HashNotes+NHash,ITEM_NAME_ATTACHMENT);
  HashNotes[NHash+strlen(ITEM_NAME_ATTACHMENT)+1]=0xFF;
  memcpy(HashNotes+NHash+strlen(ITEM_NAME_ATTACHMENT)+2,&AttNum,2);

  memcpy(HashNotes+NHash+64,hesh.h1,64);
  NHash+=128;

  return(0);
}
```

Отметим, что и по аналогии с шифрованием, при хэшировании используется параметр, определяющий порядковый номер хэшируемого файла. Если этот параметр равен 0, то хэшируются все файлы-вложения.

В процессе проведения операций хэширования сформировался хэш-вектор, который может быть использован при проверке подписи.

```
{Subject}{Categories}
{Body/85}{Body/85}{Body/A9}
```

```
{Body/85}{Body/85}{Body/A9}
{Body/85}{Body/85}{Body/A9}{Body/85}
{$FILE/FF01}{$FILE/FF02}{$FILE/FF03}
```

Этот хэш-вектор в точности отражает порядок проведения операций хэширования и перечень свойств документа, защищенных подписью. Если в документе есть несколько подписей, то хэш-вектор у каждой подписи индивидуален и хэш-вектора для различных подписей могут не совпадать.

Операцию проверки подписи можно организовать двумя способами:

- с вычислением хэш-функции по хэш-вектору, указанному в подписи;
- с независимым вычислением хэш-функции.

Первый способ более универсален, однако в некоторых случаях он может быть не совсем удобен. Дело в том, что документ может быть достаточно объемным, а подписей в нем — много, тогда трудоемкую процедуру вычисления хэш-функции необходимо будет проделать столько раз, сколько подписей присутствует в документе. Поэтому для таких случаев можно предложить операцию проверки с независимым однократным вычислением хэш-функции. Средства LotusScript позволяют сравнительно легко организовать подобные операции. Но в этом случае перечень свойств, защищаемых подписями, должен быть заранее оговорен между всеми пользователями, подписывающими документ, иначе отклонения от этого перечня приведут к тому, что результат проверки подписи даст отрицательный результат. Правда, в таком случае всегда остается возможность провести проверку с вычислением хэш-функции по хэш-вектору, указанному в подписи. Здесь уже открываются широкие возможности для разработчиков Lotus-приложений: средства LotusScript позволяют сравнительно легко реализовывать все подобные варианты.

Часть V

Автоматизированная система электронного документооборота

Глава 22. Кто принимает участие в электронном документообороте и кто его обслуживает

Глава 23. Электронный документ и протокол к нему

Глава 24. Специализированные программные модули и конфигурационные файлы

Глава 24. Специализированные программные модули и конфигурационные файлы

Глава 25. Подготовка электронных документов

Глава 26. Пересылка электронных документов

Глава 27. Контроль за доставкой электронных документов

Глава 28. Главный менеджер системы

Глава 29. Менеджер системы безопасности

Глава 30. Взаимодействие участников документооборота с менеджерами

Производство... Вот ключевое слово к этой части книги. Поточные линии, обработка большого количества однотипных документов, минимизация ручных операций, надежность. Все то, что дает основание сравнивать предлагаемую здесь криптографическую систему с производственным процессом, организованным по типу конвейера. У каждого участника — свои, строго регламентированные функции, свое автоматизированное рабочее место, свое специализированное программное обеспечение. Все операции по возможности автоматизированы. От участника часто требуется лишь ввести индивидуальный секретный ключ, все остальное сделает за него программа. Ему остается только проконтролировать результат, периодически анализировать статистические характеристики итогов работы, при необходимости просматривать архив.

Управление производством... Тоже здесь. Логика простая: пусть мы потратим больше времени на установку и наладку производственного процесса, оснащение всех рабочих мест программными модулями и индивидуальными конфигурационными файлами, но это — одноразовые затраты. Зато потом эксплуатация системы будет простой и надежной, работе с ней легко будет обучить даже тех людей, которые по роду своей деятельности весьма далеки от криптографических проблем. Дальнейшее управление большой и разветвленной системой сделаем централизованным. От централизованного управления при необходимости проще перейти к децентрализованному, чем наоборот. Централизованное управление также постараемся по возможности автоматизировать.

Универсальность... Конечно, предлагаемая здесь система разрабатывалась с учетом потребностей банковского сектора. Там процесс электронного документооборота с полным основанием можно сравнивать с производственным конвейером: балансы, статистические отчеты, платежи, подтверждения платежей, бухгалтерские формы и тысячи других подобных типовых документов ежедневно циркулируют по банковским каналам электронного документооборота. Этот процесс нужно защитить, упорядочить и автоматизировать. Но мы постараемся по возможности не учитывать специфики отдельных документов. Будем считать, что по системе автоматизированного электронного документооборота передаются обычные файлы, пусть даже в очень большом количестве. И уделим основное внимание их защите, автоматизации процессов передачи/приема, сортировке, ведению архивов. Если затем вместо банковского сектора эта система будет использоваться на крупном промышленном предприятии, с большим числом предприятий-поставщиков, смежников, дилеров, партнеров, то для ее внедрения не потребуется практически никаких переделок. Все готово, отлажено, опробовано в реальной эксплуатации.

Аналоги... Отсутствуют! Система создавалась "с нуля", в жарких спорах и дебатах, исходя из реальных потребностей заказчика, опыта и квалификации разработчиков, не один год доводилась "до ума". Что конкретно получилось — об этом и пойдет речь.

Глава 22

Кто принимает участие в электронном документообороте и кто его обслуживает

Немного пофилософствуем, по возможности "приземляя" философию к практическим задачам.

Наша задача — построить систему гарантированной защиты информации. Используем для этой цели криптографические методы, математический аппарат, теорию Шеннона и достигаем следующей цели: при сохранении в тайне уникального секретного ключа потенциальный злоумышленник практически не имеет шансов взломать систему шифрования и электронной подписи. Дает ли это абсолютную гарантию защиты информации?

Абсолютных гарантий, как и абсолютных истин, не бывает. Эта гарантия условная, условием ее выполнения является сохранение в тайне уникального секретного ключа. А дать абсолютную гарантию того, что все пользователи большой сети электронного документооборота сохранят в полной тайне все свои уникальные секретные ключи, ни один здравомыслящий разработчик криптографической системы не может. Приставляйте к каждой девушке-операционистке, работающей с секретными ключами, милиционера с пистолетом. А если милиционер отойдет на минутку или заговорится с девушкой о высоких материях? Вот тут злодей, улучив благоприятную минутку, выскочит и стащит дискету с секретным ключом. Приставляйте двух милиционеров.

Нет, господа-товарищи, так гарантированной защиты нам не достичь. Ни с помощью одного, ни двух, ни целого взвода милиционеров. Хотя, конечно, организационные меры необходимы. Но — в разумных пределах.

Что делать? Рассуждать логически. Хорошо, не можем мы в реальных условиях дать 100% гарантию сохранности в тайне уникальных секретных ключей. Предположим, что потенциальный злоумышленник сумел получить уникальный секретный ключ. А что дальше? Секретный ключ на хлеб не намажешь, на получении секретного ключа злоумышленник не остановится. Его цель — подделать жизненно важный документ, один из тех, которые проходят по системе электронного документооборота. Причем подделать

так, чтобы подделка осталась незаметной. Например, подделать платежное поручение, передаваемое по системе документооборота.

Эта постановка задачи типична для банковских сетей. Конечно, иногда целью злоумышленника может быть просто расшифрование зашифрованного документа и ознакомление с его содержанием. Тут полученного секретного ключа достаточно, злоумышленнику в этом случае необходимо просто перехватывать шифрованные сообщения на линии связи. Но будем считать приоритетной все же задачу подделки платежного документа.

Итак, у злоумышленника есть секретный ключ и доступ к линии связи, по которой передаются платежные документы. Он составляет фальшивое платежное поручение, подписывает его с помощью украденного секретного ключа и посылает в канал связи.

Криптография закончилась, математика тоже, пошла будничная рутина. К каждому платежному документу, даже с подлинной подписью, относимся с подозрением, всячески его перепроверяем. Отправителю направляем подтверждение приема. Ведем архивы отправленных/полученных документов. Используем для подписи не один, а несколько секретных ключей различных пользователей. И эта будничная рутина, реализованная в автоматизированной системе электронного документооборота, оказывается более эффективной, чем двухметровый охранник с бритым затылком. Пусть даже какой-то секретный ключ оказался скомпрометированным, но за счет этих мер компрометация может быть быстро выявлена и обезврежена!

А разработчику автоматизированной системы остается только переложить максимум подобной однотипной работы на компьютер, запрограммировать всю эту рутину, придать ей строгий алгоритмический характер и рекомендовать заказчику приставить к каждой операционистке милиционера с пистолетом. Лишних мер при обеспечении безопасности не бывает! А заказчику могу посоветовать не связываться с разработчиком, который с пеной у рта уверяет, что обеспечил 100% гарантию информационной безопасности. END PHILOSOPHY.

Займемся теперь непосредственно системой электронного документооборота. Для краткости, будем называть ее системой OMEGA.

Итак, целью системы электронного документооборота OMEGA является удобный и простой обмен электронными документами, обеспечение их гарантированной доставки по назначению, а также защита от подделки и компрометации в процессе пересылки. Такой обмен осуществляется между некоторыми адресатами, которых мы всюду далее будем называть **пользователями** системы OMEGA. Пользователем может быть как отдельное лицо, так и некоторое подразделение в организации. Каждый пользователь должен иметь индивидуальный шестисимвольный идентификатор, состоящий из больших латинских букв и цифр. Далее приводится пример пользователей некоторой типовой компании (назовем ее, к примеру, фирма "Альфа").

- ALFDIR — директор;
- ALFBGT — бухгалтерия;
- ALFMRK — отдел маркетинга;
- ALFDEV — отдел научных разработок.

Каждый из перечисленных выше пользователей является одним участником системы электронного документооборота, хотя, например, ALFDIR — это один человек, а ALFBGT — целый отдел. Нетрудно заметить, что первые три символа в идентификаторе у всех этих пользователей одинаковы и означают сокращенное название фирмы, а три последних — индивидуальны и несут в себе информацию о характере деятельности пользователя. О причинах такого выбора идентификаторов мы поговорим чуть ниже.

Что необходимо иметь пользователю, чтобы принимать участие в электронном документообороте? Во-первых, типовые программно-аппаратные средства, такие как современный персональный компьютер (слово современный в данном контексте обозначает, что компьютер должен поддерживать работу операционной системы на базе Win-32: Windows-95, Windows-NT и т. п.), сетевые средства связи (компьютер должен быть подключен либо к локальной сети, либо иметь удаленный доступ к сети через модем), операционную систему на базе Win-32, свободное место на диске для приема почты и ведения архива и, желательно, не менее 16 Мб оперативной памяти. Во-вторых, каждый пользователь должен иметь специфические средства для электронного документооборота в системе OMEGA. К таким средствам относятся:

1. комплект **специализированных программных модулей** для организации автоматизированного рабочего места пользователя;
2. **индивидуальные конфигурационные файлы** системы OMEGA, определяющие права и возможности пользователя по участию в системе электронного документооборота;
3. **индивидуальный секретный ключ**, состоящий из файла с ключом и пароля.

У любого человека, заинтересовавшегося системой OMEGA, сразу возникает вопрос: если с типовыми программно-аппаратными средствами все ясно, то как получить вторую часть требуемого, т. е. специфические средства? Охотно отвечаем.

- Специализированные программные модули закупаются у разработчиков системы OMEGA или у фирм, получивших право на их распространение.
- Индивидуальные конфигурационные файлы могут быть подготовлены для пользователя вручную. Однако при большом числе пользователей процедуру подготовки конфигурационных файлов можно осуществлять централизованно, используя специализированное программное обеспечение **главного менеджера** системы OMEGA.

- Индивидуальный секретный ключ может быть выработан пользователем самостоятельно. Для осуществления такой выработки существуют типовые программные средства — упоминавшийся ранее в этой книге модуль MCS-key. Однако при большом количестве пользователей процедуру выработки всех ключей для них можно осуществлять централизованно, используя специализированное программное обеспечение **менеджера системы безопасности**.

Здесь мы впервые встретились с двумя менеджерами системы OMEGA: главным менеджером и менеджером системы безопасности. Заметим, что о менеджере системы безопасности ранее уже неоднократно упоминалось в этой книге, а главный менеджер появился впервые. Их задача сводится не к прямому участию в системе электронного документооборота, а только к ее обслуживанию. Причем функции этих менеджеров намного важнее, чем это может показаться на первый взгляд, и чуть дальше мы еще к этому вернемся. Сейчас же познакомимся еще с одной категорией участников системы электронного документооборота — **администраторами**.

Отвлечемся ненамного от системы электронного документооборота и вспомним наш традиционный бумажный документооборот, существующий практически в любой организации. Непременной его частью является наличие секретариата, через который проходят все входящие и исходящие бумаги, в котором имеется свой архив и свои сотрудники, работающие с этими документами, причем в их функции, как правило, не входит подготовка или исполнение самих документов, а только контроль за их своевременным прохождением, получение сводных данных, периодические проверки делопроизводства и т. п. Преимущества такой системы документооборота очевидны: пользователь имеет дело только со своим секретарем, у которого всегда можно получить необходимые данные как об отдельных документах, так и сводные характеристики документооборота в целом (например, начальнику всегда интересно знать: кто сколько документов наработал, чтобы в соответствии с этим распределить премиальный фонд). Кроме того, в некоторых случаях секретариат берет на себя организацию конфиденциальной почты: секретарь подбирает конфиденциальные документы на отправку, запаковывает их, опечатывает и вручает специальному курьеру (фельдъегерю) подготовленные пакеты, который доставляет их по назначению. Процедура весьма хлопотная и, заметим, недешевая.

Внимательный читатель наверное сразу же догадался, что администратор в системе OMEGA — это тот же секретарь, который был в бумажном документообороте. Пользователи в системе OMEGA могут быть разбиты по **администрациям**, признаком принадлежности к той или иной администрации являются первые три символа в идентификаторе пользователя. Так в вышеприведенном примере все пользователи принадлежат к одной и той же администрации ALF. Администратор системы OMEGA является участником системы документооборота в том смысле, что через него проходят элек-

тронные документы, он ведет их архив, имеет индивидуальные ключи и автоматизированное рабочее место с комплектом специальных программных модулей. Однако администратор не создает первичных электронных документов: его собственные электронные документы — **пакеты** — создаются на основе электронных документов, полученных им от пользователей. Если пользователь имеет возможности работы как в автоматическом, так и в ручном (интерактивном) режиме (человек, сидящий за компьютером, сам выбирает файлы для включения в документ и адресатов), то администратор всегда работает только в автоматическом режиме, осуществляя отправку и прием документов своих пользователей — из них выбираются все необходимые данные для отправки и приема. И еще одно весьма существенное замечание: если пользователь при подготовке документа имеет возможность зашифровать его, но в то же время имеет возможность сделать нешифрованным, то все пакеты, подготовленные администратором, обязательно будут зашифрованы. После этого их можно направлять в другие администрации практически по любому открытому каналу связи: Sprint-Net, Internet, прямой модемной связи с помощью телефонной сети и т. п. Их содержание, при условии отсутствия компрометации ключей, вскрыть практически невозможно.

Отметим целесообразность создания администраций и с точки зрения безопасности электронного документооборота. Пользователей в некоторой организации может быть достаточно много, они связаны друг с другом в рамках локальной сети. Но для связи с пользователями другой организации (например, другого филиала крупного банка) используются уже глобальные сети типа Internet, Sprint-Net или прямая модемная связь по телефонной сети. Для получения доступа к глобальной сети у потенциального злоумышленника, как правило, имеется больше возможностей, чем для доступа к локальным сетям, поэтому защита информации, посылаемой в глобальные сети, должна быть более высокой, чем в локальных сетях. С этой точки зрения оправданным является *централизованный* выход в глобальные сети: пользователь не выходит в них сам, напрямую, а только через своего администратора. А для администратора в пакете его прикладных программ предусматривается более высокая степень защиты, в частности, обязательное шифрование без использования системы с открытым распределением ключей. Кроме того, при таком подходе компрометация одного секретного ключа пользователя вообще не дает потенциальному злоумышленнику никаких реальных возможностей: вся конфиденциальная информация находится в пакетах, которые подписаны и зашифрованы администратором. Для подделки, например, платежного поручения потенциальному злоумышленнику необходимы уже *три* секретных ключа: индивидуальный секретный ключ пользователя, индивидуальный секретный ключ администратора и секретный ключ для шифрования пакетов. Это не говоря уже о том, что как у всех пользователей, так и во всех администрациях ведутся индивидуальные архивы исходящих и входящих документов и пакетов, по которым факт подделки платежного поручения сразу же выявляется.

Для создания автоматизированного рабочего места администрации, как и для пользователя, необходим современный персональный компьютер, специализированные программные модули (отличные от аналогичных модулей пользователя), конфигурационные файлы, индивидуальный секретный ключ и (опять же отличие от пользователя!) таблица секретных ключей для шифрования пакетов. Здесь же отметим, что на одном рабочем месте администрации фактически могут работать несколько администраторов, каждый из которых использует одни и те же конфигурационные файлы и таблицу секретных ключей, но имеет индивидуальный секретный ключ.

Чтобы в дальнейшем не путаться в терминологии, подчеркнем еще раз различие между администраторами и администрациями.

Под администрацией будем понимать автоматизированное рабочее место (АРМ), содержащее специальное программное обеспечение, конфигурационные файлы, таблицу ключей для связи с другими администрациями. Конфигурационные файлы и таблица ключей индивидуальны для каждой администрации. Каждая администрация имеет индивидуальный 3-символьный идентификатор, состоящий из больших латинских букв и цифр, который прописывается в ее конфигурационных файлах. Под администратором будем понимать участника системы электронного документооборота, который работает на этом АРМ. Каждый администратор имеет индивидуальный идентификатор следующего вида:

ADM\NNN

где ADM — идентификатор администрации; NNN — порядковый номер администратора данной администрации.

Каждый администратор данной администрации при работе использует специализированное программное обеспечение, конфигурационные файлы администрации и таблицу секретных ключей для связи с другими администрациями, но имеет индивидуальный секретный ключ, метка которого совпадает с идентификатором администратора.

Тут мы снова вернулись к менеджерам системы — все необходимое для администрации и администраторов готовят также они. И не просто готовят, но и обеспечивают при этом безопасность всей системы в целом. Дело в том, что при функционировании системы электронного документооборота, обрабатывающей серьезные документы (финансовые документы, оперирующие с большими суммами, крупные договоры, различные базы данных, содержащие ценные сведения, и т. п.), большое значение имеет гарантия "чистоты" программных модулей, используемых для организации документооборота и выполнения криптографических процедур. Подмена или незаметное изменение алгоритма работы этих модулей могут иметь весьма неприятные последствия. Действительно, представим на минуту, что некий злоумышленник взял и подменил процедуру проверки электронной подписи, после чего она стала всегда вне зависимости от результатов проверки

выдавать код успешного выполнения. Заметить такую подмену при налаженной системе документооборота нелегко, поскольку в большинстве случаев все документы поступают с правильной электронной подписью, а операция ее проверки визуально незаметна. Реализовать же программный модуль со схожим интерфейсом (набором входных-выходных параметров), но постоянно выдающий код успешного завершения проверки подписи, для любого более-менее квалифицированного программиста не составит особого труда. Если удастся осуществить его намеренную подмену, то дальнейшие последствия нетрудно представить. Это, в частности, могут быть и фальшивые зачисления на некоторый счет крупных сумм — вспомним фальшивые авизо в Центральном банке России.

Какие меры следует принять, чтобы обезопасить себя от подобных неприятностей? В первую очередь, конечно, чисто организационные — контролировать персонал, рабочие места, сервер. Однако в системе электронного документооборота OMEGA с помощью менеджеров имеется возможность дополнить эти традиционные меры техническими, которые позволят более полно контролировать сохранность системы. Менеджеры системы OMEGA имеют возможность проставить свою электронную подпись на "жизненно важных" для системы документооборота файлах — специализированных программных модулях, индивидуальных файлах конфигурации, файлах с открытыми ключами, причем в случае, если такая подпись будет неверной или отсутствовать, работа системы OMEGA автоматически блокируется.

Однако иногда наличие электронной подписи менеджера на файлах конфигурации сковывает действия пользователей. Действительно, если некоторый пользователь хочет добавить себе возможность связи с другим пользователем, то он должен будет обратиться к главному менеджеру за новыми файлами конфигурации, поскольку самостоятельно менять их содержимое нельзя — электронная подпись менеджера будет нарушена и система перестанет работать. Однако и такая ситуация предусмотрена в системе OMEGA и сделано все возможное, чтобы облегчить возможные неудобства. Все дело в том, что OMEGA допускает возможность обработки совершенно произвольных электронных документов: это могут быть текстовые и графические файлы, таблицы, базы данных, HTML-страницы, любые другие файлы и, в частности, программные модули и новые файлы конфигурации самой системы. **Для изменения конфигурации системы OMEGA может использоваться сама же система OMEGA**. Для такого изменения в ней предусмотрены специальные возможности, на которых мы сейчас вкратце остановимся.

- У каждого пользователя и администратора имеется свой **региональный менеджер**, через которого данный участник системы документооборота получает изменения ее конфигурации. Один региональный менеджер может обслуживать несколько различных участников системы документооборота и при этом сам не являться ни пользователем, ни администратором.

- Региональный менеджер имеет собственное автоматизированное рабочее место, включающее в себя специальное программное обеспечение, конфигурационные файлы, индивидуальный секретный ключ.
- Региональный менеджер получает изменения конфигурации системы от главного менеджера и менеджера системы безопасности в виде пакетов, содержащих новые конфигурационные файлы, программные модули, открытые ключи. После обработки этих пакетов (обработка происходит в автоматическом режиме) изменения переносятся участникам системы электронного документооборота, обслуживаемым этим региональным менеджером, а главному менеджеру и менеджеру системы безопасности направляется отчет о проведенных заменах, подписанный региональным менеджером.

Итак, суммируя вкратце содержание этой главы, можно сказать, что участниками системы электронного документооборота OMEGA являются пользователи и администраторы, а обслуживающим персоналом — менеджеры: главный, региональный и системы безопасности. И, наконец, еще одно замечание для читателей, не собирающихся оснащать с помощью OMEGA больших сетей: если число ваших пользователей невелико, то вполне можно будет обойтись без администраторов и обслуживающего персонала. Состав и назначение конфигурационных файлов пользователя будут подробно рассмотрены в *гл. 24*, и возможно, что сам пользователь сможет управлять своей конфигурацией и отвечать за свою безопасность.

Глава 23

Электронный документ и протокол к нему

Кто такие участники системы электронного документооборота — это мы выяснили в предыдущей главе. Теперь давайте определим, что будем понимать под термином *электронные документы*, которые участвуют в документообороте.

Конечной целью документооборота является обмен файлами (совершенно произвольными, как было замечено ранее) между пользователями. Для каждого файла необходимо указать адресата, которому его следует направить. Если одному и тому же адресату направляется несколько файлов, то, естественно, возникает желание объединить их в один *электронный документ*, содержащий все пересылаемые файлы, и приложить к нему *протокол*, содержащий указание о том, кому он предназначен, и, возможно, некоторую дополнительную служебную информацию. Логика простая и понятная. Теперь перейдем к различным особенностям ее реализации в системе OMEGA, поскольку в них и заключается специфика всей системы документооборота.

- Каждый файл при включении его в электронный документ, чтобы сократить объем пересылаемого документа, обязательно архивируется с помощью типового архиватора, например, WINRAR95.EXE, хотя возможно подключение и другого аналогичного архиватора.
- Каждый электронный документ обязательно имеет электронную подпись отправителя. Каждый файл, входящий в электронный документ, может иметь, а может и не иметь электронной подписи, в том числе может иметь несколько электронных подписей и не обязательно отправителя.
- Каждый электронный документ включает в себя файл-реестр с перечнем файлов, вошедших в документ. В реестре для каждого файла указывается его имя и информация о том, является ли он подписанным или нет.
- Каждый файл с электронным документом имеет имя

  `<Sign>.D`

  где `<Sign>` — первые 8 символов его электронной подписи.

- Каждый электронный документ перед отправлением может быть зашифрован с использованием системы с открытым распределением ключей, и его содержание будет доступно только конечному получателю, но недоступно администраторам, при этом подписываться будет уже зашифрованный документ. Отметим, что шифруется уже готовый документ, содержащий файлы в архивированном виде.
- К каждому электронному документу прилагается, по крайней мере, один протокол, имя файла с протоколом имеет вид

  `<Sign>.P<nn>`

  где

  - `<Sign>` — упоминавшиеся уже первые 8 символов электронной подписи документа, `P` — фиксированный символ (признак протокола);
  - `<nn>` — порядковый номер протокола в шестнадцатеричной записи.

К одному электронному документу может быть приложено несколько протоколов (случай циркулярной рассылки документа нескольким получателям).

- Каждый протокол включает в себя следующую информацию:
  - тип документа — один символ, большая латинская буква (служебная информация, используемая в дальнейшем при сортировке);
  - идентификатор получателя документа;
  - электронная подпись документа в сжатом виде.
- В момент отправления документа каждый протокол подписывается отправителем. При прохождении документа через администрацию отправителя администратор подписывает протокол и один его экземпляр направляет отправителю, а другой, вместе с документом, — в пакете для администрации получателя (или прямо получателю, если отправитель и получатель находятся в одной и той же администрации). Отправитель, получив подписанный администратором протокол, заносит эти сведения к себе в архив — с момента подписи администратора ответственность за дальнейшее прохождение документа лежит уже на нем. Далее, аналогично, ответственность за документ переходит к администратору получателя и в конечном итоге к самому получателю. Каждый переход ответственности обозначается электронной подписью на протоколе. В конечном итоге отправитель получает протокол, подписанный получателем — с этого момента данная пара документ-протокол считается закрытой, т. е. дошедшей до конечного адресата. Заметим, что закрывается не документ, а пара документ-протокол, поскольку при циркулярной рассылке возможны ситуации, когда один и тот же документ до одного получателя уже дошел, а до другого — еще нет.

- Пока ответственность за документ лежит на участнике системы электронного документооборота (пользователе или администраторе) этот документ (вернее пара документ-протокол) попадают под систему автоматического контроля за прохождением документов — о ней речь пойдет далее. Как только ответственность за документ снята, он автоматически снимается с контроля.

Отметим, что вся информация о прохождении и получении документов заносится в архивы пользователей и администраторов. Моменты передачи ответственности определяются по дате и времени соответствующей электронной подписи (дата и время присутствуют в подписи). И, наконец, еще одна особенность, заложенная в систему OMEGA: при прохождении протоколов ненужные подписи автоматически отсекаются. Например, отправителю не требуется проверять подпись администратора получателя — это задача администрации отправителя, у самого отправителя может даже не быть открытых ключей администратора получателя для проверки его подписи. Эти моменты отслеживаются в системе OMEGA, и в результате отправитель сначала получает протокол с подписью только своего администратора, а затем — только с подписью конечного получателя.

Упомянем здесь еще об одной особенности прохождения документов и протоколов по системе OMEGA. Сначала от отправителя к получателю пересылается пара документ-протокол. После получения этой пары получатель запускает обратно один протокол, в котором указывается результат приема:

- если в протоколе перед подписью получателя нет никакой служебной информации (сразу начинается подпись отправителя), то результат приема положительный;
- если в протоколе перед подписью получателя указано `ERR_NEAR`, то это свидетельствует о том, что при приеме была выявлена ошибка, которую можно исправить путем повторной отправки пары документ-протокол от ближайшего участника системы ЭД в цепочке прохождения документа. Например, если при приеме было обнаружено, что все подписи верные, за исключением подписи своего администратора на протоколе, то в этом случае результатом приема считается `ERR_NEAR`. При получении такого протокола администратором по его архиву будет вновь восстановлена пара документ-протокол и она будет направлена пользователю повторно.
- если в протоколе перед подписью получателя указано `ERR_FAR`, то это свидетельствует о том, что при приеме была выявлена ошибка, которую можно исправить только путем повторной отправки пары документ-протокол от самого отправителя. Примерами являются: неверная подпись документа, неверная подпись отправителя на протоколе, несовпадение подписи документа с подписью документа, указанной в протоколе, и т. п. Отправитель, при получении такого протокола, восстанавливает по своему архиву пару документ-протокол и направляет ее получателю повторно.

Термины **электронный документ** и **протокол** будут еще очень часто встречаться нам в этой части, сейчас же выделим следующие принципиальные моменты.

> Каждый электронный документ и протокол к нему обязательно имеют электронную подпись отправителя.
>
> При получении электронного документа отправителю сразу же направляется подтверждение приема.

Последнее означает, что связь между отправителем и получателем электронного документа в системе OMEGA должна быть двусторонняя.

Еще одна категория документов, циркулирующих по системе OMEGA, — это пакеты администратора, которые, как уже отмечалось в *главе 22*, создаются администратором на основе документов и протоколов, полученных им от пользователей и предназначенных для отправки в другие администрации. Отметим, что среди документов и протоколов, полученных от пользователей, могут быть те, которые предназначены для получателя, находящегося в той же администрации. Такие документы и протоколы направляются получателю напрямую, они не включаются в пакеты, поэтому мы будем называть их *внутренними*.

Процесс составления пакетов администратора имеет некоторые отличия от процесса составления электронных документов пользователя.

- Каждый пакет состоит из ровно одного *документа типа пакет*, подготовленного администратором из документов пользователей и протоколов, а также ровно одного протокола к документу типа пакет. Заметим, что в документ типа пакет могут включаться документы и протоколы к ним, полученные от пользователей, а также отдельные протоколы на полученные администратором входящие документы типа пакет.
- Документы и протоколы не архивируются перед включением в документ типа пакет. Дело в том, что документы являются уже архивированными файлами, а основное содержание протокола составляют его подписи, которые сжимаются с помощью специальных алгоритмов системы OMEGA.
- Каждый документ типа пакет обязательно имеет электронную подпись подготовившего его администратора.
- Каждый документ типа пакет имеет файл-реестр, содержащий информацию о вошедших в него документах и протоколах.
- Каждый файл с документом типа пакет имеет имя

  `<Sign>.D`

  где `<Sign>` первые 8 символов его электронной подписи.

- Протокол к документу типа пакет имеет имя

  <Sign>.P01

  где <Sign> — первые 8 символов электронной подписи документа типа пакет.

Состав протокола к документу типа пакет тот же, что и для протокола к документу пользователя. В качестве типа документа в протоколе к документу типа пакет всегда выбирается символ *P*. Протокол подписывается администратором, изготовившим документ типа пакет.

- Каждый пакет перед отправлением обязательно шифруется.
- Файл с пакетом имеет имя

  <TRG><Sign>.<SRC>

  где

  - <TRG> — трехсимвольный идентификатор администрации-получателя;
  - <SRC> — трехсимвольный идентификатор администрации-отправителя;
  - <Sign> — первые 5 символов электронной подписи документа типа пакет.

Порядок прохождения документов типа пакет по системе OMEGA во многом схож с порядком прохождения документов пользователей, однако здесь есть одно весьма важное отличие документов типа пакет от документов пользователя:

> Если документ типа пакет содержит только протоколы, то протокол с подтверждением его приема отправителю не направляется.

Это связано с тем, что в противном случае мы получили бы бесконечный круговорот документов типа пакет: протокол о получении документа типа пакет включается в новый документ типа пакет от получателя к отправителю, протокол его приема приводил бы к необходимости формирования нового документа типа пакет от отправителя к получателю и т. д. Если же документ типа пакет содержит документы пользователей, то тогда при его получении отправителю направляется протокол, подписанный администратором-получателем.

Документ типа пакет, содержащий документы пользователей, ставится на учет в системе автоматического контроля за доставкой. В случае, если за заданное время не поступил протокол его приема, такой документ типа пакет направляется повторно. Документы типа пакет, не содержащие документов пользователей, на учет в системе автоматического контроля за доставкой не ставятся.

Ранее уже неоднократно упоминались архивы пользователей и администраций. В них, как это интуитивно ясно, должны храниться все отправленные и полученные документы и пакеты. Давайте рассмотрим их подробнее.

Архив как пользователя так и администрации условно можно разделить на три части:

- DDB (Document Data Base) — база данных документов, содержит информацию о содержании электронных документов, используется, в частности, для восстановления документа при его повторной отправке;
- SDB (Sign Data Base) — база данных подписей, содержит информацию о прохождении документа, подписи участников системы ЭД (администраторов и пользователей), которые получили документ;
- RDB (Reestr Data Base) — база данных реестров, содержит реестры (списки файлов, вошедших в документ). Предназначена для работы с архивом в интерактивном режиме, а также для восстановления пакетов администратора при их повторной отправке.

Отметим, что пакеты администрации (как исходящие, так и входящие) *не хранятся* в архиве. Каждый пакет состоит из документов и протоколов, в архиве же хранятся только документы и протоколы, поэтому каждый пакет при необходимости может быть из них восстановлен. Нет необходимости хранить там еще и пакеты.

Вкратце остановимся на строении архивов, поскольку это имеет очень важное значение для быстродействия при большом объеме документооборота.

Архив ведется отдельно для исходящих и входящих документов. Каждая часть архива состоит из отдельных файлов, которые строятся исходя из принятого в конфигурации системы *разбиения* архива. Всевозможные типы конфигураций разбиения архива приведены ниже в табл. 23.1.

Таблица 23.1. *Типы разбиения архива*

Конфигурация	Пояснения	Пример
`*****\YY[-MM[-DD]]`	Разбиение только по датам	`C:\BASE\SDB.OUT\97-05-05`
`<ADM>***\YY[-MM[-DD]]`	Выделенная администрация	`C:\BASE\SDB.OUT\ALF\97-05-05` `C:\BASE\SDB.OUT\$$$\97-05-05`

`???***\YY[-MM[-DD]]`	Разбиение по администрациям	`C:\BASE\SDB.OUT\ALF\97-05-05`
		`C:\BASE\SDB.OUT\BET\97-05-05`
		`C:\BASE\SDB.OUT\GAM\97-05-05`

`<ADMUSR>\YY[-MM[-DD]]`	Выделенный пользователь	`C:\BASE\SDB.OUT\ALFDIR\97-05-05` `C:\BASE\SDB.OUT\$$$$$$\97-05-05`

Таблица 23.1 *(окончание)*

Конфигурация	Пояснения	Пример
<ADM>???\YY[-MM[-DD]]	Разбиение по пользователям выделенной администрации	C:\BASE\SDB.OUT\ALFDIR\97-05-05 C:\BASE\SDB.OUT\ALFBGT\97-05-05 ..... C:\BASE\SDB.OUT\$$$$$$\97-05-05
??????\YY[-MM[-DD]]	Разбиение по всем пользователям всех администраций	C:\BASE\SDB.OUT\ALFDIR\97-05-05 C:\BASE\SDB.OUT\ALFBGT\97-05-05 C:\BASE\SDB.OUT\BETDIR\97-05-05 C:\BASE\SDB.OUT\BETBGT\97-05-05 .....

Пара "текущая дата — отправитель/получатель" в большинстве случаев однозначно определяет выбор соответствующего файла. Однако каждая часть архива имеет свои особенности.

База данных документов DDB

Имена файлов выбираются в соответствии с упоминавшимся уже разбиением архива. Документы пишутся в файлы DDB последовательно, формат записи приведен ниже в табл. 23.2.

Таблица 23.2. *Формат записи в базу данных документов*

Смещение	Длина поля	Содержание поля
0	4	Значение $L-(277-115)\times N+12+1$, где • L — длина документа; • N — количество подписей в нем
4	12	Имя файла с документом
16	1	Количество подписей в документе
17	$L-277N$	Содержание документа без подписей
$17+L-277N$	$115N$	Подписи документа в сжатом виде
$17+L-(277-115)N$	2	!! — признак конца записи

Этот формат достаточно универсальный, он также будет использоваться при составлении пакетов.

Для каждого файла, входящего в DDB, существует также индексный файл. Его имя совпадает с именем основного файла, только к нему добавляется

расширение .IND. Индексный файл имеет размер 2^{17} = 131072 байт, это 2^{25} бит. При создании файла, входящего в DDB, все байты его индексного файла — нулевые. В процессе работы некоторые *биты* из индексного файла заменяются на 1 следующим образом. Имена файлов с документами формируются по первым 8 символам подписи, поэтому состоят из случайных и равновероятных шестнадцатеричных символов. Общее число всевозможных имен равно $2^{4\times 8} = 2^{32}$. Из этих 32 бит первые 25 определяют адрес бита, который заменяется на 1.

По индексному файлу строится *проверка на повторяемость* входящих документов. Это очень важная процедура, особенно при пересылке платежных документов: ведь повторный прием одного и того же такого документа может привести к повторному зачислению денежных средств, что недопустимо. Поэтому в системе OMEGA предусмотрен контроль за повторяемостью входящих документов, о котором более подробно пойдет речь в *главе 27* этой книги. Сейчас же отметим, что наличие индексного файла позволяет сделать такой контроль достаточно быстрым и надежным.

База данных подписей SDB

Это одна из наиболее важных частей архива, содержащая всю информацию о прохождении документа и о том, кто в данный момент является за него ответственным. К SDB система OMEGA обращается постоянно, а при получении протокола с результатами приема документа должна быстро найти соответствующую запись в SDB и сделать в ней пометку о приеме. С этой точки зрения, каким наиболее оптимальным способом необходимо строить SDB?

Записи в SDB строятся по паре "документ-протокол" по формату, приведенному ниже в табл. 23.3.

Таблица 23.3. *Формат записи в базе данных подписей*

Смещение	Длина поля	Содержание поля
0	12	Имя файла с протоколом
12	1	Тип документа
13	1	Индикатор циркулярного документа (0 — не циркулярный, 1 — циркулярный)
14	3	YMD — год, месяц, день (в ASCII-коде) соответствующей записи в DDB
17	4	Смещение в файле DDB
21	4	Смещение в файле RDB, соответствующее началу реестра

Таблица 23.3 *(окончание)*

Смещение	Длина поля	Содержание поля
25	2	Количество файлов, вошедших в документ (количество записей в реестре)
27	1	Индикатор того, что документ не может быть восстановлен
28	2	Максимальное время ожидания подтверждения приема, в минутах
30	1	Максимально разрешенное количество попыток отправить документ
31	6	Получатель документа
37	115	Подпись документа
152	115	Подпись отправителя на протоколе
267	115	Подпись администратора на протоколе
382	115	Подпись получателя на протоколе
497	15	Резервные поля

Запись называется *открытой*, если в ней отсутствует подпись получателя на протоколе, в противном случае запись называется *закрытой*.

Очевидно, что необходимо разделить файлы с открытыми и закрытыми записями в SDB, поскольку при получении результатов приема поиск соответствующей записи необходимо осуществлять только среди открытых записей. После получения протокола с подписью получателя документа запись из числа открытых переносится в закрытые, т. е. физически эти 512 байт переписываются из одного файла в открытой части SDB в некоторый другой файл в закрытой части SDB. Таким образом, объем открытой части SDB остается более-менее постоянным и небольшим, что позволяет выполнять упоминавшийся уже поиск нужной записи в нем достаточно быстрым. Для увеличения скорости такого поиска файлы открытой части SDB также можно разбить, но уже не по времени и отправителю/получателю, а например, по первым символам в имени подписи документа. Заполнение таких файлов, в силу того, что подпись состоит из случайных и равновероятных символов, будет равномерным.

Файлы, составляющие закрытую часть SDB, можно строить по тому же разбиению, что и аналогичные файлы для DDB — см. табл. 23.1.

База данных реестров RDB

Та часть архива, которая непосредственно интересует пользователя. Это перечни отправленных и полученных файлов, а также документы, в которые они включены. Имена файлов из RDB строятся в соответствии с разбиением из табл. 23.1, формат записи в файле приведен ниже в табл. 23.4.

Таблица 23.4. *Формат записи в базе данных реестров*

Смещение	Длина поля	Содержание поля
0	12	Имя файла
12	1	Индикатор подписи файла (S — файл подписан, N — не подписан)
13	12	Имя протокола к документу
25	6	Получатель документа
31	7	DMYHMSH — дата подписи протокола отправителем в ASCII-коде (день, месяц, год, часы, минуты, секунды, сотые доли секунды)
38	7	DMYHMSH — дата подписи протокола администратором в ASCII-коде (день, месяц, год, часы, минуты, секунды, сотые доли секунды)
45	7	DMYHMSH — дата подписи протокола получателем в ASCII-коде (день, месяц, год, часы, минуты, секунды, сотые доли секунды)
52	4	Смещение соответствующей записи в файле SDB

Для интерактивной работы с архивом в комплекте программного обеспечения как пользователя, так и администрации имеется специализированный модуль (см. рис. 23.1). Он позволяет отображать архив на экран, осуществлять типовые операции сортировки, поиска, выписки, восстановления документа и т. п.

Архивы системы OMEGA разрабатывались специально с учетом большого объема документооборота и специфических задач, которые они должны выполнять. В настоящее время одна администрация системы OMEGA реально может обработать на типовом компьютере несколько тысяч документов в день.

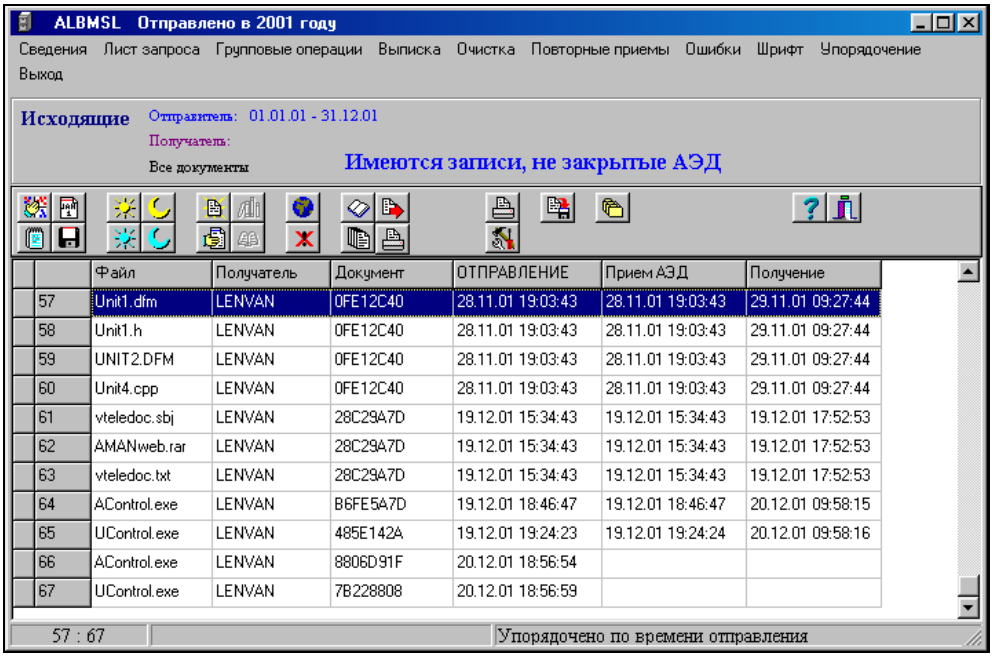

Рис. 23.1. Специализированный программный модуль для работы с архивом системы OMEGA

Глава 24

Специализированные программные модули и конфигурационные файлы

В *главе 23* мы уже упоминали о том, что как пользователи, так и администраторы для участия в системе электронного документооборота должны иметь специализированные программные модули и конфигурационные файлы. Поговорим о них несколько подробнее.

Специализированные программные модули (СПМ) реализуют типовые процедуры системы OMEGA, и они могут быть общими для нескольких пользователей в рамках локальной сети. СПМ располагаются в специально выделенном для них каталоге, который в дальнейшем мы будем называть SoftDir. Каждый пользователь или администратор должен иметь доступ к тому SoftDir, который он использует. Для обеспечения надежной работы системы электронного документооборота в некоторых случаях представляется целесообразным установить стандартные способы защиты файлов с СПМ от их случайного искажения или удаления: сетевыми средствами ограничить доступ посторонних пользователей к SoftDir, установить файлам с СПМ атрибут "только чтение", создать контрольные копии файлов с СПМ. Наиболее ценные из СПМ (например, модуль MCS.DLL, осуществляющий криптографические функции) целесообразно защитить от подделки электронной подписью менеджера. Более подробно об этом пойдет речь в *главе 28*, посвященной Главному менеджеру системы OMEGA.

Конфигурационные файлы, в отличие от СПМ, как правило индивидуальны для каждого пользователя и администрации. Они записываются в специально выделенном для них каталоге, который мы в дальнейшем будем называть SetupDir. Желательно, чтобы доступ к этому каталогу имел только тот участник системы электронного документооборота, который использует эти конфигурационные файлы. Конфигурационные файлы, как правило, состоят из отдельных строк, в начале каждой строки указывается название задаваемого параметра, обрамленное символами =, а затем его конкретное значение. Например, строка

```
=SOFT= C:\OMEGA\SOFT
```

в основном конфигурационном файле SETUP задает SoftDir для данного участника системы электронного документооборота.

Отметим следующие особенности построения конфигурационных файлов.

- Как у пользователя, так и у администрации есть основной конфигурационный файл — SETUP. В нем задаются такие основные параметры, как идентификатор (строка `=ID=`), SoftDir (`=SOFT=`), почтовые ящики (`=IN_BOX=`, `=OUT_BOX=`), расположение и структура архива (`=BASE=`, `=MASK=`), местоположение ключа (`=KEYPATH=`) и другие.
- Большинство из конфигурационных файлов имеют строку `=ID=`, задающую идентификатор участника системы ЭД (пользователя или администрации). Это означает, что другой участник системы ЭД уже не сможет использовать этот конфигурационный файл, т. е. значение строки `=ID=` этого файла должно совпадать с аналогичным значением из файла SETUP, в противном случае выполнение процедуры, использующей этот конфигурационный файл, блокируется. Примерами таких конфигурационных файлов являются файл OUT.BAS настроек параметров автоматической отправки исходящей почты пользователя, файл IN.BAS параметров сортировки принятой пользователем почты, файл MEMBER.AED со списком пользователей администрации и другие.
- Некоторые конфигурационные файлы не имеют строки `=ID=`. Это означает, что один и тот же файл (точнее, одна и та же копия этого файла) может использоваться несколькими участниками системы ЭД. Примерами таких файлов являются: файл TYPES.DOC, задающий допустимые типы электронных документов, файл ABONENTS, задающий группировку адресатов при работе пользователя в интерактивном режиме, и некоторые другие.
- В системе OMEGA имеются некоторые конфигурационные файлы, которые создаются и редактируются в процессе работы пользователя или администратора автоматически или же самим пользователем или администратором. Примером такого файла является файл ASKLIST, содержащий листы запроса к архиву. Часть записей в этом файле участник системы ЭД создает в интерактивном режиме (когда при работе с архивом выбирает параметры листа запроса и сохраняет их), а запись *Текущий лист запроса к архиву* создается автоматически при выходе из архива. Во избежание ошибок при работе системы OMEGA на такие файлы нельзя ставить опцию "только чтение" или защищать их электронной подписью менеджера.
- Некоторые конфигурационные файлы системы OMEGA состоят из нескольких записей определенной структуры. Нарушение этой структуры может привести к ошибкам при работе системы, невозможности выполнить ту или иную процедуру, зависящую от содержания этого конфигурационного файла. Примером такого файла является файл PUBLKEYS, содержащий открытые ключи участников системы ЭД. Каждая запись

этого файла содержит один открытый ключ, идентификатор его владельца, срок ввода в эксплуатацию, подпись менеджера системы безопасности. Изменение структуры файла PUBLKEYS (например, удаление или добавление записей с помощью обычного текстового редактора, при котором пропали или добавились некоторые служебные символы) может привести к тому, что подписи некоторых участников системы ЭД проверяться не будут и поступающие от них документы будут забракованы. Поэтому если вы решились на коррекцию такого файла вручную, то необходимо внимательно ознакомиться со структурой записи в нем и после окончания правки проверить (хотя бы по длине файла), что она сохранена.

В этой книге преследуется цель дать читателю только самое общее представление о системе электронного документооборота OMEGA, точное детальное описание всевозможных конфигурационных файлов и особенностей их строения можно найти в технической документации по этой системе. Поэтому остановимся только на основных конфигурационных файлах, которые непосредственно влияют на информационную безопасность, и постараемся объяснить ту логику, по которой они выбирались и применялись в этой системе.

Начнем с основного конфигурационного файла SETUP. Этот файл, как уже отмечалось ранее, состоит из отдельных параметров, которые прописываются в виде строк, каждый параметр имеет название, выделенное символами '=', и следующее за ним значение, отделенное как минимум одним пробелом. Здесь же отметим, что порядок следования строк с параметрами в файле SETUP роли не играет.

Ниже мы постараемся дать описание основных его параметров и ту роль, которую они выполняют в системе OMEGA.

Конфигурационный файл SETUP начинается со строки `=ID=`, содержащей идентификатор пользователя, и, возможно, строки `=NAME=`, содержащей название этого участника системы электронного документооборота в произвольной форме, не более 128 символов. Строка `=ID=` обязательна, строка `=NAME=` — нет. Например:

```
=ID= ALFDIR
=NAME= Директор фирмы "Альфа"
```

Далее в файле SETUP следуют строки, задающие различные параметры и возможности участника системы электронного документооборота. Часть этих строк является обязательной, при отсутствии которых система работать не будет, часть — вспомогательной, при их отсутствии выбираются значения по умолчанию. Далее мы рассмотрим и поясним некоторые возможные строки файла SETUP.

Операционная система Windows дает большую свободу для поиска при обращении к динамической библиотеке DLL. Так, если при вызове библиотеки

DLL указать только имя файла DLL без информации о диске или маршруте, то Windows производит поиск динамических библиотек в следующем порядке:

- текущий каталог;
- каталог Windows, в котором находится win.com;
- системный каталог Windows;
- каталог, в котором находится исполняемый файл текущей задачи;
- все каталоги, перечисленные в переменной PATH среды DOS;
- каталоги, отображаемые сетью.

Процедуры, осуществляющие криптографические операции (шифрование, подпись), в системе OMEGA вызываются из динамической библиотеки MCS.DLL. Это, конечно, хорошо, что Microsoft дал такую широкую свободу при вызове DLL, но если ей воспользоваться при проверке подписи платежных поручений, то трудно гарантировать защиту от подделок, связанных с подменой этого модуля при разветвленной сети. Поэтому **в системе OMEGA все специализированные программные модули должны располагаться в каталоге, указанном в строке** `=SOFT=` конфигурационного файла SETUP и нигде иначе. Строка `=SOFT=` обязана присутствовать в конфигурационном файле SETUP пользователя и администратора.

Любой пользователь осуществляет обмен электронными документами с другими участниками системы документооборота через два почтовых ящика: для входящей и исходящей почты. Эти почтовые ящики есть просто некоторые каталоги, которые задаются в файле SETUP пользователя строками `=IN_BOX=` и `=OUT_BOX=` соответственно. Строки `=IN_BOX=` и `=OUT_BOX=` у пользователя обязательны. У администрации ситуация более сложная: необходимо иметь `=IN_BOX=` и `=OUT_BOX=` своих пользователей, а кроме того, собственные каталоги для отправки и приема пакетов. Каталоги пользователей вынесены у администратора в отдельный файл MEMBER.AED, а для отсылки/приема пакетов предусмотрена строка `=POSTE=`.

Любой участник системы электронного документооборота обязан иметь личный секретный ключ. Отметим, что строка `=KEYPATH=`, указывающая каталог, где будет находиться файл с секретным ключом, обязательно должна присутствовать в файле SETUP пользователя и администрации. В этом же каталоге будут находиться также открытые ключи Главного менеджера, менеджера системы безопасности и модуль проверки подписи при тестировании программного комплекса. Строка `=KEYPATH=` допускает также запись типа:

```
=KEYPATH= C:\KEY,iButton:COM2
```

В этом случае секретный ключ будет считываться из устройства Touch Memory, подключенного к порту COM2, а открытые ключи менеджеров и программа проверки подписи — из каталога C:\KEY.

В любой почтовой системе (Microsoft Exchange, Outlook Express, Sprint-Net и др.) ведется архив исходящей и входящей почты. Система OMEGA не исключение. Архив OMEGA имеет множество различных возможностей, учитывающих специфику его использования, таких как:

- контроль за прохождением документов;
- контроль за повторяемостью входящих документов (чтобы не принять дважды один и тот же документ);
- хранение конфиденциальных документов в шифрованном виде;
- оптимизация объемов архива при циркулярной рассылке одного и того же документа многим адресатам;
- различная сортировка находящихся в архиве документов, создание системы фильтров при выборках из архива;
- возможность разбиения архива по датам и адресатам (для удобства его очистки и локализации последствий выхода из строя отдельных файлов архива);
- возможность создания и загрузки выписок из архива;
- прочие...

Раздела архива, в котором находятся сведения о содержании документа (DDB, DocumentDataBase), представлен строкой `=DDB=`. Раздел, в котором находятся сведения об (SDB — SignDataBase) и (RDB — ReestrDataBase), представлен строкой `=BASE=`. Обе эти строки обязательны для пользователя, без них система работать не будет. И, наконец, еще одна обязательная строка, относящаяся к архиву, — это `=DELIVERY_NO=`. В ней дается ссылка на файл, в который будут записываться сведения о документах, на которые не было получено подтверждение их приема.

Для архива пользователя в файле SETUP предусмотрены несколько необязательных параметров. К ним относятся:

- `=MASK=` — характеристика разбиения архива;
- `=SDB_SEPARATE=` — разбиение открытой части базы данных подписей;
- `=DDB_TYPE=` — тип базы данных документов (INERNAL или EXTERNAL — внутренняя или внешняя).

Для потенциальных потребителей системы OMEGA одной из привлекательных ее особенностей является возможность установления автоматического контроля за прохождением почты. Об этом мы уже упоминали в *главе 23*, приведем суть системы контроля вкратце.

Для любого электронного документа, отправляемого по системе OMEGA, пользователь имеет возможность установить срок, в течение которого он должен получить подтверждение его приема, причем такое подтверждение должно прийти или от своего администратора (в этом случае ответствен-

ность за дальнейшее прохождение документа ложится на администратора и в дальнейшем будет использоваться его система контроля) или от конечного получателя документа. Если срок прошел, а подтверждения нет, то документ направляется повторно, однако общее количество повторных отправок не может расти до бесконечности. Для каждого документа устанавливается лимит повторных отправок, при исчерпании которого документ переводится в раздел =DELIVERY_NO= архива и снимается с контроля.

При подготовке пользователем документа к отправке в интерактивном режиме максимальное время ожидания подтверждения приема и максимальное число попыток отправки могут быть установлены непосредственно с клавиатуры. Однако по умолчанию для каждого документа эти параметры выбираются из файла SETUP:

- =MAX_WAIT= — максимальное время ожидания подтверждения приема (в часах или сутках);
- =MAX_ATTEMPT= — максимальное число повторных отправок.

И, наконец, если эти строки в файле SETUP не заданы, то по умолчанию предполагается =MAX_WAIT= 5 day, =MAX_ATTEMPT= 3.

Как уже упоминалось ранее, в системе OMEGA предусмотрена блокировка возможности принять один и тот же документ дважды. Такая опасность существует, например, при неудачном выборе параметров автоматического контроля, когда повторная отправка осуществляется слишком рано, а документ еще просто не успел дойти до получателя. Перед принятием документа в системе OMEGA осуществляется поиск точно такого же по содержанию документа в базе данных входящих документов. Если он найден, то приема не происходит (под приемом понимается занесение документа в архив входящих документов), а уведомление о приеме найденного документа отсылается повторно. Таким образом, на один и тот же документ пользователь может получить несколько уведомлений о его получении. Первое из этих уведомлений заносится в базу данных, а остальные оказываются уже ненужными. Однако в некоторых случаях информация о повторных приемах может оказаться полезной. Например, наличие повторных приемов свидетельствует о неправильном выборе =MAX_WAIT= и =MAX_ATTEMPT= либо о халатном отношении получателя к своим обязанностям — он подолгу не включает компьютер, в результате чего у него скапливается большой объем входящей почты и теряется оперативность ее обработки. Задав строку =DOUBLE_SAVE= в файле SETUP, можно определить файл, в котором будут сохраняться все данные о повторных приемах, с тем чтобы в дальнейшем была возможность их проанализировать. Строка не является обязательной.

В файле SETUP предусмотрена строка =BASKET= — каталог, определяемый как корзина для мусора. В нее будут записываться все данные, не прошедшие тех или иных проверок в ходе работы системы OMEGA. Например, если в принятом входящем документе обнаружена неверная подпись, то документ

и протокол к нему будут помещены в корзину. Если строка =BASKET= отсутствует, то такие данные будут просто удаляться.

Несколько специфических параметров файла SETUP относятся только к пользователю и процессам, связанным с приемом входящей почты.

В комплекте специального программного обеспечения пользователя имеется модуль IN_POSTE.EXE, позволяющий контролировать состояние =IN_BOX= — почтового ящика для входящей почты. Эта программа периодически проверяет наличие в нем почты, и если обнаруживает, то выдает звуковой сигнал и сообщение на экран с предложением ее принять. Периодичность проверки почтового ящика программой IN_POSTE.EXE может быть задана в файле SETUP строкой =CHECK_INBOX= в минутах или секундах. При отсутствии строки периодичность по умолчанию составляет 30 секунд. Тип звукового сигнала может быть установлен с помощью строки =SOUND= файла SETUP: там должна быть ссылка на файл типа WAV.

Входящая почта, поступившая пользователю, заносится в его архив. Однако с помощью различных способов ее можно "разложить по полочкам", т. е. отсортировать по тем или иным признакам. Более подробно мы поговорим об этом в *главе 26*, посвященной пересылке электронных документов, однако сейчас, при описании файла SETUP, упомянем об одном самом простом способе сортировки, который может быть задан параметрами этого файла.

При приеме входящей почты, в архив пользователя (точнее в базу данных документов =DDB=) попадают не конечные файлы, а сами документы в том виде, в каком они проходили по системе документооборота. А каждый документ может содержать в себе несколько отдельных файлов. Конечно, всегда возможно вручную восстановить эти файлы из архива, однако при большом объеме документооборота это будет весьма трудоемкой процедурой. Поэтому в файле SETUP предусмотрены параметры, указывающие процедуре приема входящей почты на необходимость автоматического восстановления всех файлов из архива сразу же после осуществления приема документа. Такими параметрами являются строка =CORRECT.IN= — каталог для правильно принятых файлов, и =CORRECT.IN_MASK= — маска разбиения этого каталога.

Как уже отмечалось в *главе 22*, для изменения конфигурационных файлов системы OMEGA может использоваться сама же система OMEGA путем пересылки по ней новых конфигурационных файлов. Эти файлы поступают региональному менеджеру в его почтовый ящик для входящей почты и находятся там до тех пор, пока он не осуществит замену. Среди новых конфигурационных файлов могут быть весьма важные, например, новый комплект открытых ключей, если их вовремя не сменить, то пользователи могут в дальнейшем не понимать друг друга, используя разные ключи. Поэтому весьма важно иметь оповещение о поступлении изменений конфигурации системы, которое задается специальной командой в сценарии работы

и строкой =CHANGE_IN= в файле SETUP. В этой строке фактически указывается каталог =IN_BOX= регионального менеджера. При оповещении также будет выдаваться звуковой сигнал, указанный в строке =SOUND=.

Следующая группа параметров файла SETUP относится к вопросам организации почтовой связи. Наряду с традиционной схемой, когда пользователь и администратор связаны между собой локальной сетью, возможны случаи, когда пользователь общается со своим администратором по внешним каналам связи через Internet, Sprint-Net, модем. Такого пользователя будем называть *удаленным*. Кроме того, в OMEGA допускается, что с некоторыми адресатами пользователь связывается напрямую, без участия администратора, и даже возможны такие случаи, когда для пользователя администратора нет вообще. Во всех этих случаях пользователь может сам использовать для связи внешние каналы, и в системе OMEGA предусмотрено сопряжение с некоторыми из них. Более подробно мы коснемся этого вопроса в *главе 26*, посвященной почте, сейчас же поясним следующие параметры файла SETUP:

- =SPRINT= — каталог, содержащий почтовые ящики (IN.BOX, OUT.BOX, SENT.BOX) почты Sprint. Этот параметр является необходимым в том случае, если предполагается использование почты Sprint;
- =PMAIL.EXE= — ссылка на выполняемый файл PMAIL.EXE почты Pegasus Mail;
- =PMAIL.IN= — входящий почтовый ящик почты Pegasus Mail.

Последние две строки необходимы в том случае, если предполагается использование почты Pegasus Mail.

И, наконец, последние параметры файла SETUP — это картинка-заставка системы OMEGA и параметры контроля свободного дискового пространства. Строка =IMAGENAME= (картинка-заставка) — это ссылка на произвольный файл типа BMP, который будет появляться на экране в основном меню системы. Параметры контроля наличия свободного дискового пространства задаются строками =MIN_FREE=, каждая строка определяет параметры одного диска, содержит идентификатор диска и необходимый размер свободного дискового пространства на нем. Например, =MIN_FREE= C:10Mb означает, что на диске C должно быть не менее 10 Мбайт свободного дискового пространства. Такая проверка осуществляется с помощью соответствующей процедуры, которая включается в сценарий работы OMEGA. При невыполнении этого условия работа системы OMEGA блокируется. Количество строк =MIN_FREE= в файле SETUP может быть любым, однако в базе данных менеджера, используемой для подготовки и хранения файлов SETUP всех участников системы электронного документооборота, предусмотрено не более 8 строк в одном файле.

Обычно параметры, аналогичные тем, которые указаны в файле SETUP, принято прописывать в реестре Windows. Но в системе OMEGA они сознательно

вынесены в отдельный файл с тем, чтобы Главный менеджер системы смог затем подписать его и тем самым гарантировать сохранность настроек рабочего места пользователя или администрации. Расположение ключей, архивов, почтовых ящиков, их настройки — все это непосредственно влияет на информационную безопасность. Если, к примеру, потенциальный злоумышленник сумеет подменить в настройках каталога определяющие используемые почтовые ящики, то такая подмена в условиях почти полностью автоматизированной системы документооборота может повлечь за собой различные неприятные последствия. Поэтому мы и защищаем эти жизненно важные параметры путем подписи файла SETUP.

Вторым (после SETUP) по значимости конфигурационным файлом пользователя является файл RK.BAS, содержащий полный список всех пользователей, с которыми разрешена связь в рамках системы OMEGA. Отметим, что в этом файле указывается только список пользователей, с которыми разрешена связь, но не указывается, каким конкретно способом она будет осуществляться: напрямую по электронной почте или через администратора. Эта информация содержится в конфигурационных файлах INTERNET.BAS, MAIL.BAS, SPRINT.BAS и VMAIL.BAS системы электронной почты.

Файл RK.BAS состоит из строк-записей, каждая запись обязательно должна начинаться с идентификатора пользователя. После идентификатора может (необязательно) быть указано название пользователя, отделенное от идентификатора как минимум одним пробелом.

Ниже приводится пример файла RK.BAS для пользователя `ALFDIR`:

```
ALFDIR    директор фирмы "Альфа"
ALFBGT    бухгалтерия
ALFMRK    отдел маркетинга
ALFDEV    отдел научных разработок
BETDIR    директор фирмы "Бета"
GAMDIR    директор фирмы "Гамма"
BNKPAY    отдел платежных поручений банка
```

Отметим, что файл RK.BAS должен быть индивидуальным для каждого пользователя.

Файл RK.BAS, аналогично файлу SETUP, может быть защищен электронной подписью Главного менеджера, и таким образом определяются права пользователя по связям с другими пользователями. Добавить себе возможность связи с другими пользователями без изменения файла RK.BAS пользователь не может.

У администрации файла RK.BAS нет, там есть два различных конфигурационных файла: MEMBER.AED и TRANSIT.BAS.

В файле MEMBER.AED расписано все о пользователях администрации: идентификаторы, почтовые ящики для исходящей и входящей почты, тип

базы данных документов. Этот файл начинается со строки =ADM=, содержащей 3-символьный идентификатор администрации, а затем содержит несколько разделов =ITEM= {...} по количеству пользователей, обслуживаемых данной администрацией.

Внутри каждого раздела присутствуют такие строки:

- =NAME= — название пользователя (необязательный параметр);
- =USER= — 3-символьный идентификатор пользователя (без указания администрации), обязательный параметр;
- =FAR= – признак удаленного пользователя (=FAR= TRUE) или прямого (строка отсутствует или =FAR= FALSE);
- =IN_BOX= — почтовый ящик для входящей почты — обязательный параметр. Если пользователь не является удаленным, то данная строка должна совпадать со строкой =IN_BOX= в файле SETUP пользователя;
- =OUT_BOX= — почтовый ящик для исходящей почты — обязательный. Если пользователь не является удаленным, то данная строка должна совпадать со строкой =OUT_BOX= в файле SETUP пользователя;
- =DDB_TYPE= — тип базы данных документов — INTERNAL (внутренняя) или EXTERNAL (внешняя). Параметр необязательный, по умолчанию — INTERNAL. Если пользователь удаленный, то обязательно =DDB_TYPE= INTERNAL, если нет — то значение должно совпадать с аналогичным значением в файле SETUP пользователя;
- =DDB= — база данных документов. Если =DDB_TYPE= EXTERNAL, то значение строки =DDB= должно совпадать с аналогичным значением в файле SETUP пользователя;
- =MASK= — маска разбиения базы данных документов. Если =DDB_TYPE= EXTERNAL, то значение строки =MASK= должно совпадать с аналогичным значением в файле SETUP пользователя. Необязательный параметр.

Пора дать пояснения ко всему этому.

Итак, администрация обслуживает группу пользователей. Обслуживание заключается в следующем.

- Отправка исходящей почты пользователей. Просматриваются исходящие почтовые ящики, из них забираются документы на отправку, из которых готовятся и отправляются исходящие пакеты.
- Прием входящей почты. Принимаются и распаковываются входящие пакеты из других администраций, их содержимое разносится по входящим почтовым ящикам пользователей.

Во время прохождения почты осуществляется повторная отправка документов пользователей и пакетов, у которых истек срок получения подтверждения приема.

Пользователи каким-либо образом должны связываться с администрацией. Способов связи два: прямой, когда пользователь и администратор имеют доступ к общим каталогам в локальной сети, и удаленный, когда пользователь связывается с администратором по внешним каналам связи — Internet, Sprint-Net, по телефону через модем. Какой тип связи будет использоваться (прямой или удаленный) — указывается в строке =FAR=. Отметим, что способ организации удаленной связи (через Internet, Sprint-Net или модем) здесь не описывается.

Далее описываются два почтовых ящика: =IN_BOX= и =OUT_BOX=. Если пользователь прямой, то это просто каталоги в сети, к которым имеют доступ как пользователь, так и администратор. Если удаленный, то эти каталоги должны быть настроены на последующую отправку и прием из них почты пользователю по внешнему каналу связи.

Следующие три строки в разделе файла MEMBER.AED относятся к способу создания базы данных документов пользователя. Несколько предварительных замечаний.

Зачем администратору нужна база данных документов пользователя? Все дело в том, что, получив документ от пользователя, администратор берет на себя ответственность за его дальнейшее прохождение. В частности, документ попадает под систему контроля администратора, которая будет осуществлять его повторные отправки до тех пор, пока не придет подтверждение приема получателем. Для осуществления повторных отправок необходима база данных документов, содержащая документы пользователя. Администрация может вести собственную базу данных документов пользователя, а может, для экономии дискового пространства, воспользоваться той базой данных, которую ведет сам пользователь. В последнем случае, как администратор, так и пользователь должны иметь к ней доступ по локальной сети. В случае, если параметр =DDB_TYPE= INTERNAL, то базы данных документов у администрации и пользователя собственные (хотя документы там могут быть одни и те же), а если =DDB_TYPE= EXTERNAL — общая. Если пользователь удаленный, то он не может иметь доступа к одному и тому же каталогу вместе с администратором в рамках локальной сети, поэтому для удаленного пользователя всегда =DDB_TYPE= INTERNAL.

Для всякой базы данных в системе OMEGA предусмотрено ее разбиение. Это разбиение обязательно осуществляется по времени и при желании может осуществляться по адресатам. Управляет разбиением базы данных документов пользователя строка =MASK=.

Аналогом файла RK.BAS у администрации является файл TRANSIT.BAS, содержащий полный список всех администраций, с которыми разрешена связь в рамках системы OMEGA. Отметим, что в этом файле указывается только список администраций, с которыми разрешена связь, но не указывается, каким конкретно способом она будет осуществляться. Эта информация

содержится в конфигурационных файлах INTERNET.BAS, MAIL.BAS, SPRINT.BAS и VMAIL.BAS системы электронной почты.

Файл TRANSIT.BAS состоит из строк-записей, каждая запись обязательно должна начинаться с идентификатора администрации. После идентификатора может (необязательно) быть указано название администрации, отделенное от идентификатора как минимум одним пробелом.

Ниже приводится пример файла TRANSIT.BAS для администрации `ALF`:

```
ALF   фирма "Альфа"
BET   фирма "Бета"
GAM   фирма "Гамма"
```

Отметим, что файл TRANSIT.BAS должен быть индивидуальным для каждой администрации.

Как у пользователя, так и у администрации обязательно должен быть файл PUBLKEYS с открытыми ключами участников системы ЭД. Этот файл индивидуален для каждого участника, поскольку в нем содержатся только требуемые открытые ключи и никаких лишних.

Про открытые ключи и связанные с ними проблемы мы уже подробно говорили в *главе 9*. Напомним, что:

- каждый открытый ключ имеет дату ввода в эксплуатацию;
- каждый открытый ключ имеет электронную подпись менеджера системы безопасности.

Сам файл PUBLKEYS целиком не подписывается. В *главе 29*, посвященной менеджеру системы безопасности, мы еще вернемся к вопросам, связанным с изготовлением и рассылкой файлов PUBLKEYS.

Одно из основных назначений системы OMEGA, как уже отмечалось ранее, является рассылка электронных документов в автоматическом режиме. Что это означает на практике? То, что система OMEGA в этом случае стыкуется с некоторой внешней системой (например, с системой типа "Банковский операционный день"), выдающей файлы, которые необходимо гарантированно доставить по назначению. В месте назначения отправленные файлы требуется принять и подать на вход аналогичной системы. Для такой стыковки необходимо иметь некоторые каталоги, через которые будет осуществляться подобный обмен, причем как исходящими, так и входящими файлами. Эти каталоги задаются в конфигурационных файлах OUT.BAS и IN.BAS. Смысл формирования отдельных файлов остается прежним — подписать и гарантировать тем самым их сохранность.

Файл OUT.BAS, также как и файл SETUP, начинается со строки `=ID=`, содержащей идентификатор пользователя. После нее в файле OUT.BAS записывается необходимое число разделов, каждый из которых выполняет одну процедуру отправки почты (транзакцию). Каждый раздел начинается с заголовка

=ITEM=, и его содержание заключено в фигурные скобки {...}. В составе каждого раздела должны присутствовать следующие строки.

- =TARGET= — получатель документа. В качестве получателя может быть указан точный идентификатор одного единственного адресата, а может быть указан идентификатор, содержащий неопределенные символы &&& вместо символов администрации, и ### вместо символов пользователя. Единственное ограничение — получатель должен быть прописан в файле RK.BAS (если задан явно), или по явно заданным символам (отличным от & и #) в RK.BAS можно найти по меньшей мере одного адресата. Смысл введения неопределенных символов будет ясен чуть позже.
- =DOC_TYPE= — тип подготавливаемого документа — большая латинская буква.
- =PREP_MASK= — маска подготовки файлов к отправке. Электронный документ состоит из набора файлов. Строка =PREP_MASK= определяет операции, которые необходимо проделать с каждым файлом перед включением его в документ. К таким операциям относятся подпись файла отправителем и проверка некоторого количества уже имеющихся в файле подписей.

Состав строки:

<TestSign><Sign><TestSignID1TestSignID2....>

где:

- <TestSign> — индикатор проверки подписей (T — проверку подписей проводить, N — не проводить);
- <Sign> — индикатор подписи (S — подписать файл, N — не подписывать);
- <TestSignID1TestSignID2....> – строка идентификаторов для проверки подписей.

Примеры строки:

- =PREP_MASK= TSALFBGTALFDEV — перед включением в документ для каждого файла осуществляется проверка подписей ALFBGT и ALFDEV, и сам файл после этого подписывается;
- =PREP_MASK= NS — перед включением в документ файл подписывается;
- =PREP_MASK= TNALFBGT — перед включением в документ проверяется подпись ALFBGT, файл не подписывается;
- =PREP_MASK= NN — файл, безусловно, включается в документ и не подписывается;
- =CRYPT_MASK= — признак шифрования документа. Если =CRYPT_MASK= C, то подготовленный документ будет зашифрован, если =CRYPT_MASK= N — нет. Заметим, что шифруется весь документ целиком, а не отдельные входящие в него файлы;

- =FILE_MASK= — маска поиска файлов, подлежащих включению в документ. В маске поиска могут присутствовать символы &&& и ###, упоминавшиеся в строке =TARGET=. При обработке раздела они будут последовательно заменяться на символы идентификаторов администраций и пользователей соответственно.
- =MAX_WAIT= и =MAX_ATTEMPT= — с ними мы уже встречались при описании файла SETUP, их назначение здесь — точно такое же. Это единственные необязательные параметры в разделе, при их отсутствии значения берутся из файла SETUP.

Отметим, что после успешного осуществления отправки, все файлы, задаваемые строкой =FILE_MASK=, удаляются.

Ниже приводится пример файла OUT.BAS пользователя ALFDIR, и далее с помощью табл. 24.1 дается описание процессов, которые будут происходить при его обработке.

```
=ID= ALFDIR
Управление фирмой
=ITEM= {
=TARGET= ALF###
=DOC_TYPE= U  (управление фирмой)
=PREP_MASK= NS
=CRYPT_MASK= N
=FILE_MASK= C:\OMEGA\OUT\UPRAVL\ALF###\*.*
=MAX_WAIT= 12 hour
=MAX_ATTEMPT= 3
}

Договора
=ITEM= {
=TARGET= &&&DIR
=DOC_TYPE= D  (договора)
=PREP_MASK= TSALFBGTALFDEV
=CRYPT_MASK= C
=FILE_MASK= C:\OMEGA\OUT\DOGOVOR\&&&\*.DOC
}

Платежные поручения
=ITEM= {
=TARGET= BNKPAY
=DOC_TYPE= P  (платежные поручения)
```

```
=PREP_MASK= TSALFBGT
=CRYPT_MASK= C
=FILE_MASK= C:\OMEGA\OUT\PAY\*.*
=MAX_WAIT= 1 day
=MAX_ATTEMPT= 5
}
```

Предположим, что у пользователя ALFDIR на компьютере имеются файлы, представленные в табл. 24.1.

Таблица 24.1. *Пример отправки в автоматическом режиме*

Файл	Описание
C:\OMEGA\OUT\UPRAVL\ALFBGT\ PRIKAZ1B.DOC	Файл с приказом о назначении главного бухгалтера фирмы "Альфа"
C:\OMEGA\OUT\UPRAVL\ALFBGT\ PRIKAZ2B.DOC	Файл с приказом о назначении кассира фирмы "Альфа"
C:\OMEGA\OUT\UPRAVL\ALFMRK\ EXEBIT98.TXT	Файл с перечнем выставок, в которых планирует принять участие фирма "Альфа"
C:\OMEGA\OUT\UPRAVL\ALFDEV\ PLAN98.RTF	План работ отдела научных исследований
C:\OMEGA\OUT\DOGOVOR\BET\ DOG98.DOC	Договор с фирмой "Бета"
C:\OMEGA\OUT\DOGOVOR\GAM\ DOG98.DOC	Договор с фирмой "Гамма"
C:\OMEGA\OUT\PAY\01-12-98.DBF	Платежные поручения за день, направляемые в банк

Последовательность операций, которая будет проделана в этом случае системой OMEGA при отправке почты в автоматическом режиме, следующая.

1. Считывается первый раздел файла OUT.BAS (Управление фирмой). В нем сначала определяется строка =TARGET=.
2. Для строки =TARGET= (в данном случае ALF###) открывается файл RK.BAS и в нем отыскиваются все адресаты, удовлетворяющие этой строке. В данном случае такими адресатами будут ALFBGT, ALFMRK и ALFDEV.
3. Для каждого найденного адресата просматривается соответствующий каталог с файлами на отправку, строится исходящий документ и протокол к нему. В данном случае для адресата ALFBGT будет построен документ, содержащий файлы C:\OMEGA\OUT\UPRAVL\ALFBGT\PRIKAZ1B.DOC и C:\OMEGA\OUT\UPRAVL\ALFBGT\PRIKAZ2B.DOC, для адресата ALFMRK — документ, содержащий файл C:\OMEGA\OUT\UPRAVL\

ALFMRK\EXEBIT98.TXT, для адресата ALFDEV — документ, содержащий файл C:\OMEGA\OUT\UPRAVL\ALFDEV\PLAN98.RTF.

Каждый из этих файлов перед включением в документ будет подписан `ALFDIR`.

Остальные разделы файла OUT.BAS обрабатываются аналогично.

Остановимся теперь на процессе автоматической обработки входящих документов. Этот процесс состоит из двух этапов:

- проверка входящих документов и внесение их по ее результатам в архив;
- сортировка файлов из принятых документов по различным каталогам.

Первый из этих этапов является обязательным и включает в себя выполнение типовых процедур проверки входящей документации, заложенных в алгоритме работы системы OMEGA: проверка на просроченность, проверка подлинности подписей документа и протокола, проверка соответствия реальной подписи документа той, которая указана в протоколе, проверка соответствия реального содержания документа его реестру и т. п. Сейчас, при описании конфигурационных файлов, остановимся подробнее на втором этапе обработки входящих документов — сортировке принятых файлов. Сразу же отметим, что этот этап не является обязательным при осуществлении обработки входящей почты: если в конфигурационных файлах отсутствуют соответствующие параметры, то сортировка выполняться не будет, а принятая почта останется только в архиве.

Об одном из способов сортировки мы вкратце упоминали при описании файла SETUP, когда рассказывали о строках `=CORRECT.IN=` и `=CORRECT.IN_MASK=`. Сейчас остановимся на нем подробнее.

В системе OMEGA каталог, указанный в строке `=CORRECT.IN=`, выполняет функции некоторого "электронного шкафа с полочками", а `=CORRECT.IN_MASK=` определяет количество таких полок и порядок расстановки на них входящих файлов. При наличии строки `=CORRECT.IN=` в файле SETUP все файлы из входящих документов будут поступать в некоторые подкаталоги этого каталога в зависимости от даты поступления и идентификатора отправителя в соответствии с маской `=CORRECT.IN_MASK=`. Эта маска состоит из двух частей, разделенных символом '\'. Первая часть определяет разбиение по времени (по годам, месяцам или дням), а вторая — по отправителям. Разбиение по времени является обязательным, разбиение по отправителям — нет. Синтаксис этих частей маски следующий:

- разбиение по датам, возможны только три варианта:
  - `YY` — разбиение только по годам;
  - `YY-MM` — разбиение по месяцам;
  - `YY-MM-DD` — разбиение по дням.

- разбиение по отправителям, возможны следующие варианты:
  - `******` — разбиения по отправителям нет;
  - `??????` — разбиение по администрациям;
  - `ADM***` — выделенная администрация;
  - `ADM???` — разбиение по пользователям выделенной администрации;
  - `ADMUSR` — выделенный пользователь;
  - `??????` — разбиение по пользователям.

Необходимые пояснения лучше всего сделать на примерах.

Пусть для традиционно упоминаемого нами в примерах директора фирмы "Альфа" `ALFDIR` в файле SETUP есть строка `=CORRECT.IN= C:\OMEGA\CORRECT.IN`. Предположим, что он ведет интенсивную переписку с фирмами "Бета" и "Гамма", и рассмотрим различные варианты сортировки его входящей почты.

- `=CORRECT.IN_MASK= YY\******`

  В этом случае при сортировке входящей почты в каталоге C:\OMEGA\CORRECT.IN создаются только подкаталоги по годам:

  97

  98

  99

- `=CORRECT.IN_MASK= YY-MM\******`

  Сортировка по месяцам. Пример подкаталогов:

  98-11

  98-12

  99-01

  99-02

- `=CORRECT.IN_MASK= YY-MM-DD\******`

  Сортировка по дням. Пример подкаталогов:

  98-12-29

  98-12-30

  98-12-31

  99-01-01

  99-01-02

- `=CORRECT.IN-MASK= YY\???***`

  Сортировка по годам и администрациями. Пример подкаталогов:

  - `98\BET` — файлы от всех пользователей фирмы "Бета" за 1998 год;

- 98\GAM — файлы от всех пользователей фирмы "Гамма" за 1998 год;
- 99\BET — файлы от всех пользователей фирмы "Бета" за 1999 год;
- 99\GAM — файлы от всех пользователей фирмы "Гамма" за 1999 год.

❒ `=CORRECT.IN_MASK= YY-MM\???***`

Сортировка по годам, месяцам и администрациями. Пример подкаталогов:

- `98-11\BET`
- `98-12\BET`
- `99-01\BET`
- `98-11\GAM`
- `98-12\GAM`
- `99-01\GAM`

❑ `=CORRECT.IN_MASK= YY\BET***`

Выделенная фирма "Бета":

- 98\BET — файлы от всех пользователей фирмы "Бета" за 1998 год;
- 99\BET — файлы от всех пользователей фирмы "Бета" за 1999 год;
- 98\$$$ — файлы от всех других пользователей за 1998 год;
- 99\$$$ — файлы от всех других пользователей за 1999 год.

На этом примере видно, что при выделенной администрации все файлы от остальных пользователей заносятся в специальный каталог, содержащий вместо идентификатора символы '`$`'. Этот же принцип (создание подкаталога "для всего остального" с символами '`$`') используется и везде далее.

❒ `=CORRECT.IN_MASK= YY\BET???`

Разбиение по пользователям выделенной администрации:

- 98\BETDIR — файлы от директора фирмы "Бета" за 1998 год;
- 98\BETBGT — файлы от бухгалтерии фирмы "Бета" за 1998 год;
- 98\$$$$$$ — файлы от всех других пользователей за 1998 год.

❒ `=CORRECT.IN_MASK= YY\BETDIR`

Выделенный пользователь `BETDIR`

- 98\BETDIR — файлы от директора фирмы "Бета" за 1998 год;
- 98\$$$$$$ — файлы от всех других пользователей за 1998 год.

❒ `=CORRECT.IN_MASK= YY\??????`

Разбиение по всем пользователям:

- 98\BETDIR — файлы от директора фирмы "Бета" за 1998 год;
- 98\BETBGT — файлы от бухгалтерии фирмы "Бета" за 1998 год;

- 98\GAMDIR — файлы от директора фирмы "Гамма" за 1998 год;
- 98\GAMBGT — файлы от бухгалтерии фирмы "Гамма" за 1998 год.

С "электронным шкафом" `=CORRECT.IN=` разобрались. А теперь вспомним про упоминавшийся ранее конфигурационный файл IN.BAS — это целая коллекция подобных шкафов и полок.

Конфигурационный файл IN.BAS, так же как и файл SETUP, начинается со строки `=ID=`, содержащей идентификатор пользователя. После нее в файле IN.BAS записывается необходимое число разделов, каждый из которых выполняет одну процедуру сортировки принятой почты. Каждый раздел начинается с заголовка `=ITEM=`, и его содержание заключено в фигурные скобки {...}. В составе каждого раздела должны присутствовать следующие строки:

- `=SOURCE=` отправитель документа. В качестве отправителя может быть указан точный идентификатор одного единственного адресата, а может быть указан идентификатор, содержащий неопределенные символы `&&&` вместо символов администрации, и `###` вместо символов пользователя.
- `=DOC_TYPE=` — тип входящего документа — большая латинская буква или символ '`*`' — любой тип.
- `=FILE_MASK=` — маска поиска входящих файлов, подлежащих сортировке (в отличие от файла OUT.BAS здесь указывается только маска файлов без каких-либо каталогов).
- `=TARGET_DIR=` — каталог, в который следует поместить отсортированные файлы, возможно, что в ней присутствуют символы '`&`' и '`#`'.
- `=DATE_SEPARATE=` — признак сортировки по времени внутри `=TARGET_DIR=`. Параметр необязательный, возможны три значения: `NONE`, `CURRENT`, `SOURCE`. Единственная необязательная строка раздела, по умолчанию предполагается значение `NONE`.

Ранее уже упоминались конфигурационные файлы INTERNET.BAS, MAIL.BAS, VMAIL.BAS, SPRINT.BAS, определяющие канал связи, используемый для доставки электронных документов по назначению. В *главе 26*, посвященной вопросам доставки электронных документов, мы поговорим об этих файлах подробнее, сейчас же отметим, что эти файлы также строго индивидуальны и могут быть подписаны Главным менеджером системы OMEGA.

Ну и теперь, наконец, рассмотрим заключительный этап наладки системы OMEGA — сценарий ее работы. Специализированные программные модули — это строительный материал, который содержит отдельные процедуры работы с системой. Сценарий работы — это конечный продукт, построенный из отдельных процедур и определяющий те операции, которые можно будет осуществлять на рабочем месте пользователя или администрации.

Работа в системе OMEGA начинается с основного меню — это окно с кнопками и, возможно, картинкой (рис. 24.1).

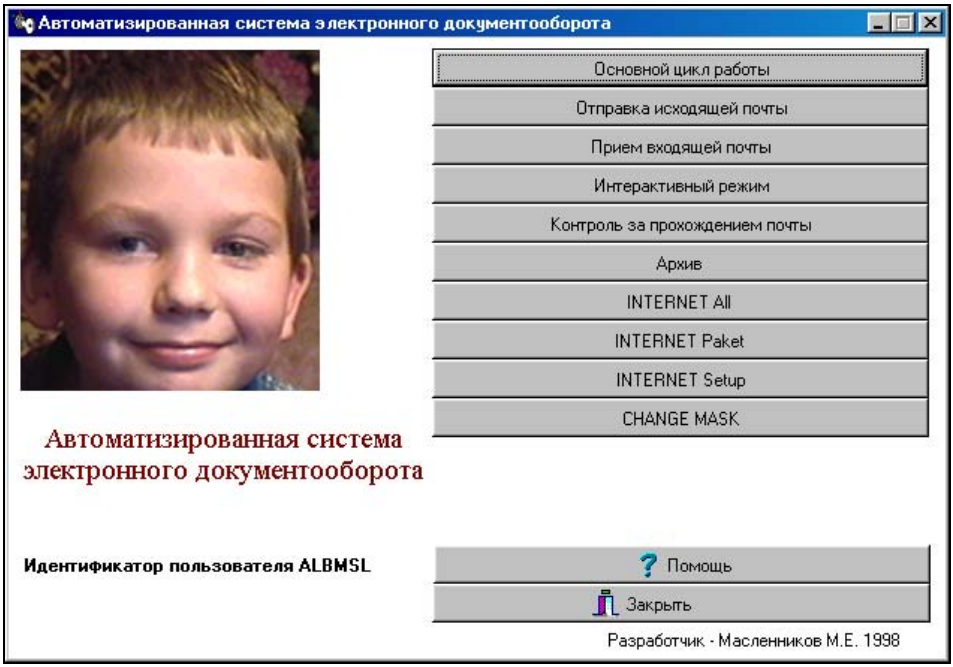

Рис. 24.1. Основное меню системы OMEGA

Картинка задается в файле SETUP (строка `=IMAGENAME=`), а в сценарии работы описывается, что будет происходить при нажатии на кнопки, а также их наклейки. Общее количество программируемых кнопок в сценарии не должно превосходить 12. Сценарий, т. е. мини-программы, описывающие действия системы OMEGA при нажатии на ту или иную кнопку, записываются в конфигурационном файле WINMAIN.BAS. Рассмотрим его.

Файл WINMAIN.BAS состоит из разделов, каждый раздел заключен в фигурные скобки {...}.

Внутри каждого раздела могут быть следующие строки:

- `=NAME=` — название кнопки, обязательный параметр, без него кнопка восприниматься не будет;
- `=IMAGE=` — имя файла-наклейки на кнопку; файл должен быть формата BMP типа тех, которые используются в BORLAND C++ BUILDER для наклеек на клавиши. Параметр необязательный;
- `=BREAK_AFTER=` — количество циклов, выполняемых при нажатии на кнопку (необязат.);

- `=DELAY=` — время задержки между циклами (необязат.);
- `[Procedure]` — выполняемая процедура системы OMEGA или выполняемый файл.

Отметим, что строк `[Procedure]` в одном разделе может быть несколько — это соответствует тому случаю, когда последовательно выполняются несколько процедур, например: ввод ключа, отправка, прием, контроль. В принципе, в одном разделе может быть любое разумное число процедур, однако в программе менеджера при ведении базы данных, из которой строятся файлы WINMAIN.BAS для всех участников системы ЭД, предусмотрено, что число процедур в одном разделе не может быть больше 10.

Поясним синтаксис строки `[Procedure]`. В ней могут указываться либо внутренние процедуры системы OMEGA (их список приводится ниже в табл. 24.2), либо произвольные выполняемые файлы с расширением EXE, COM или BAT. Внутренние процедуры всегда начинаются с символа _. Некоторые внутренние процедуры допускают возможность циклического повторения, в этом случае в круглых скобках указываются число циклов и время задержки между циклами. Ниже в табл. 24.2 приведен список внутренних процедур для пользователя системы OMEGA. Модуль — название программного модуля, который необходимо иметь для запуска процедуры в каталоге `=SOFT=` пользователя, цикл — допускает ли данная процедура внутреннее циклическое использование (будет пояснено позже), дополнительные параметры — требуются ли при вызове модуля дополнительные параметры.

Таблица 24.2. *Внутренние процедуры файла WINMAIN.BAS пользователя*

	ID процедуры	Название	Модуль	Цикл	Дополнительный параметр
1	`_PrepearSecretKey`	Ввод секретного ключа	MCS.DLL		
2	`_CheckDiskFree`	Проверка наличия свободного дискового пространства	USERMAIN.EXE		
3	`_TestChange`	Проверка наличия изменений конфигурации системы	USERMAIN.EXE		
4	`_Control`	Контроль за прохождением электронных документов	USERMAIN.EXE	X	

Таблица 24.2 *(окончание)*

	ID процедуры	Название	Модуль	Цикл	Дополнительный параметр
5	_Out_auto	Отправка исходящей почты пользователя в автоматическом режиме	USERMAIN.EXE	X	
6	_User_in	Прием входящей почты пользователем	USERMAIN.EXE	X	
7	_Manual	Интерактивный режим пользователя	USERMAIN.EXE		
8	_Archiv	Интерактивная работа с архивом	ARCH_USR.EXE		
9	_Legal	Проверка сохранности системы OMEGA	USERMAIN.EXE		
10	_Vypiska	Автоматическое создание выписок из архива	VYPISKA.DLL		X
11	_Mail	Почта MAIL	EMAIL.DLL		X
12	_Sprint	Почта Sprint	EMAIL.DLL		X
13	_Spr_cont	Контроль за почтой Sprint Mail	EMAIL.DLL		X
14	_Vmail	Почта Vmail	EMAIL.DLL		X
15	_Internet	Прямая почта через Internet	INTERNET.EXE		X
16	_InputInternet Password	Ввод пароля для хоста Internet	USERMAIN.EXE		

Для некоторых из этих процедур поясним, что они делают и для чего нужны.

- _PrepearSecretKey — ввод ключа в оперативную память. По строке =KEYPATH= осуществляется чтение файла с ключом, запрос пароля и формирование на их основе в оперативной памяти секретных ключей для шифрования и подписи. При неправильном выполнении процедуры (файл с ключом не найден или ошибка при вводе пароля) работа всего раздела (группы процедур) прерывается. Следует отметить, что наличие

ключа в оперативной памяти проверяется всякий раз перед выполнением процедур шифрования или подписи, и если ключ не обнаружен, то процедура _PrepearSecretKey запускается автоматически.

- _CheckDiskFree — проверка наличия свободного дискового пространства. Для работы этой процедуры в файле SETUP необходимо задать одну или несколько строк =MIN_FREE=, в которых описаны проверяемые логические диски и минимально допустимое свободное место на них. При отсутствии места хотя бы на одном из этих дисков работа всего раздела прерывается, а на экран выдается сообщение о диске, на котором отсутствует свободное дисковое пространство.
- _TestChange — проверка наличия изменений конфигурации системы. Для работы этой процедуры в файле SETUP необходимо задать строку =CHANGE_IN=, при наличии в этом каталоге каких-либо файлов подается звуковой сигнал =SOUND= и на экран выдается сообщение-предупреждение. Работа раздела не прерывается.
- _Legal — проверка сохранности системы. Осуществляется проверка электронной подписи менеджера на конфигурационных файлах и специализированных программных модулях.

Отметим некоторые особенности построения файла сценария WINMAIN.BAS.

О цикличности. В каждом разделе файла WINMAIN.BAS может быть два типа циклов: общий цикл всего раздела и отдельный цикл какой-нибудь процедуры, допускающей циклическое использование (у таких процедур стоит отметка в графе "Цикл"). Для задания цикличности отдельной процедуры, как отмечалось выше, необходимо в круглых скобках указать два параметра: число циклов и время задержки между ними в минутах или секундах. Общий цикл осуществляется в том случае, если в разделе заданы строки =BREAK_AFTER= и =DELAY=, причем циклически будут повторяться не все процедуры раздела, а лишь те, у которых в строке указана спецификация CKL. Без такой спецификации процедура выполняется только в первом цикле.

Теперь рассмотрим некоторые примеры.

Предположим, что необходимо задать циклически отправку исходящей и затем прием входящей почты через каждые 5 минут. Ниже приводится соответствующий раздел в файле WINMAIN.BAS.

```
{
=NAME= Циклическая отправка и прием
=IMAGE= C:\OMEGA\IMAGES\MAIN.BMP
=BREAK_AFTER= 2000
=DELAY= 5min.
[_PrepearSecretKey]
[_out_auto,CKL]
[_user_in,CKL]
}
```

При нажатии на эту клавишу осуществится:

- ввод секретных ключей в оперативную память (только в первом цикле);
- отправка;
- прием;
- задержка 5 минут;
- если количество выполненных циклов менее 2000, то новый цикл.

Предположим, что в вашей организации отправки происходят чаще, чем прием. В этом случае можно модернизировать приведенный ранее раздел.

```
{
=NAME= Циклическая отправка и прием (преобладание отправок)
=IMAGE= C:\OMEGA\IMAGES\MAIN.BMP
=BREAK_AFTER= 2000
=DELAY= 5min.
[_PrepearSecretKey]
[_out_auto(3min,50),CKL]
[_user_in,CKL]
}
```

При нажатии на эту клавишу осуществится:

```
общий цикл {
  ввод секретных ключей в оперативную память (только в первом общем цикле)
  счетчик отправок =0
  цикл отправки{
    отправка
    задержка 3 минуты
    счетчик отправок +1
    если счетчик отправок <50, то новый цикл отправки
  }
  прием
  задержка 5 минут
  если количество выполненных общих циклов <2000, то новый общий цикл
}
```

Несколько примеров с подключением почтовых процедур.

Отправка и прием по Sprint Mail.

```
{
=NAME= Цикл работы с использованием Sprint Mail
=IMAGE= C:\OMEGA\IMAGES\MAINCKL.BMP
=BREAK_AFTER= 2000
=DELAY= 5min.
[_PrepearSecretKey]          Ввод секретного ключа в ОП
```

```
[_Legal,CKL]                  Проверка сохранности системы
[_out_auto(1min,10),CKL]      Отправка в автоматическом режиме
[_control,CKL]                Контроль за прохождением документов
[_sprint(A),CKL]              Отправка и прием по Sprint-Net
[_user_in,CKL]                Обработка входящей почты
}
```

Цикл работы с использованием нескольких почтовых систем.

```
{
=NAME= Цикл работы MultiPost
=IMAGE= C:\OMEGA\IMAGES\MULTICKL.BMP
=BREAK_AFTER= 2000
=DELAY= 5min.
[_PrepearSecretKey]       Ввод секретного ключа в ОП
[_Legal,CKL]              Проверка сохранности системы
[_out_auto(1min,10),CKL] Отправка в автоматическом режиме
[_control,CKL]            Контроль за прохождением документов
[_Internet(A),CKL]        Прямая отправка по INTERNET
[_sprint(A),CKL]          Отправка и прием по Sprint-Net
[_spr_cont,CKL]           Проверка результатов отправки по Sprint-Net
[_mail(A),CKL]            Отправка и прием по альтернативной почте
[_user_in,CKL]            Обработка входящей почты
}
```

Подготовка баланса в MS Word:

```
{
=NAME= Подготовка баланса в WINWORD
=IMAGE= C:\OMEGA\IMAGES\BALANCE.BMP
[C:\MSOFFICE\WINWORD\WINWORD.EXE(C:\BALANCE\BALANCE.DOC) ]
}
```

Настройка файла INTERNET.BAS.

```
{
=NAME= Настройка INTERNET.BAS
=IMAGE= C:\OMEGA\IMAGES\I_CONFIG.BMP
[C:\OMEGA\SOFT\I_CONFIG.EXE(C:\OMEGA) ]
}
```

С помощью создания нужных конфигурационных файлов мы в конечном итоге можем добиться периодической отправки и приема файлов из заданных каталогов и автоматически проконтролировать процесс их доставки по назначению. А это и есть типовой производственный процесс.

Глава 25

Подготовка электронных документов

В этой главе мы поговорим подробнее о подготовке электронных документов в системе OMEGA.

Подготовка любого электронного документа включает в себя:

- создание файла с электронным документом;
- создание файла с протоколом к электронному документу;
- создание файла-директивы;
- подготовка объединенных файлов с документом, протоколами и файлом-директивой.

Начнем с создания файла с электронным документом пользователем. Как уже отмечалось ранее в *главе 23*, любой электронный документ — это объединение отдельных файлов, предназначенных к отправке. Выбор таких файлов может быть осуществлен в автоматическом или интерактивном режиме.

В автоматическом режиме пользователю достаточно просто скопировать предназначенный для отправки файл в определенный каталог (этот каталог указывается в конфигурационном файле OUT.BAS), после чего файл автоматически включается в документ в ходе выполнения процедуры отправки в автоматическом режиме. Адресаты, которым предназначен этот документ, также задаются в файле OUT.BAS. Сразу же отметим, что в этом случае после включения в документ файл из каталога удаляется.

В интерактивном режиме пользователь выбирает файлы, предназначенные к отправке, и адресатов вручную. В этом режиме после включения в документ файл не удаляется.

Перед включением в документ как в автоматическом, так и в интерактивном режиме можно осуществить следующие операции с отдельными файлами.

- Проверить наличие в них определенных подписей и их подлинность. Таких подписей может быть несколько, и они могут принадлежать совершенно произвольным лицам, возможно даже не участвующим в OMEGA

в качестве пользователей или администраторов. Однако в файле PUBLKEYS должны быть их открытые ключи.

- Подписать отдельный файл перед включением его в документ.

Для самого документа можно установить тип документа и определить следующие параметры:

MaxWait — время ожидания подтверждения его приема;

MaxAttempt — максимальное количество повторных отправок.Отметим, что параметры MaxWait и MaxAttempt обязательно определяются для каждого документа, они устанавливаются или непосредственно пользователем в интерактивном режиме, либо в настройках конфигурационного файла OUT.BAS, либо выбираются по умолчанию из файла SETUP.

Документ также можно зашифровать, т. е. сделать его недоступным даже для администратора. Как уже отмечалось ранее в *главе 23*, отдельные файлы включаются в документ, помощью архиватора типа WINRAR95.EXE и автоматически в документ добавляется файл-реестр. Файл-реестр используется при проверке правильности приема: по нему проверяется перечень реально полученных файлов. Далее, если документ необходимо зашифровать, то он шифруется с использованием системы с открытым распределением ключей. В выработке ключа для шифрования принимают участие секретный ключ отправителя и открытый ключ получателя документа. Последний этап подготовки документа — его подпись отправителем и переименование файла с документом по первым 8 символам этой электронной подписи.

Отметим теперь некоторые особенности, связанные с адресацией документа. Документ может быть адресован как одному, так и нескольким пользователям (т. н. циркулярная рассылка), получатель документа задается в специальном файле-протоколе к документу. Для каждого адресата к документу прилагается отдельный протокол, поэтому по системе документооборота циркулируют не отдельные документы, а пары "документ-протокол". Если получателей несколько, а документ — нешифрованный, то отправитель готовит один файл с документом и несколько протоколов к нему. Если документ шифрованный и получателей несколько, то отправитель готовит для каждого получателя отдельную пару "документ-протокол", поскольку документ шифруется на ключе, зависящем от открытого ключа получателя. Всюду далее документы, к которым прилагаются несколько (больше одного) протоколов, мы будем называть *циркулярными*. Из рассуждений видно, что шифрованный документ не может быть в этом понимании циркулярным, хотя, конечно, послать нескольким пользователям одинаковые по составу входящих в них файлов шифрованные документы вполне возможно.

Отметим, что протокол жестко привязан к определенному документу тем, что в него входит электронная подпись документа. Протокол также включает в себя получателя и тип документа. Заметим, что тип документа — большая

латинская буква — может использоваться получателем для сортировки принятой почты. Каждый протокол имеет собственную электронную подпись отправителя.

В зависимости от настроек конфигурационных файлов, каждый документ пользователь может направить получателю или через своего администратора, или же напрямую. Если документ направляется через администратора, то к нему, помимо протокола, прилагается также и *файл-директива*.

Как уже отмечалось ранее, для каждого документа устанавливаются MaxWait и MaxAttempt, которые необходимо сообщить администратору при пересылке документа через администрацию, поскольку дальнейшая ответственность за документ ложится на администратора и он осуществляет повторные отправки. Для передачи MaxWait и MaxAttempt администратору и предназначен файл-директива. Однако в некоторых случаях в него включается и другая информация.

В процессе подготовки электронного документа и протоколов к нему все данные о документе и протоколах автоматически заносятся в специализированные архивы пользователя. Как уже отмечалось в предыдущих главах, одной из составных частей архива является база данных документов (DDB, Document Data Base), в которую заносится содержание каждого документа. У администратора ведется своя DDB, в которую автоматически заносятся документы, поступающие от пользователей. При большом объеме документооборота размеры DDB могут быть весьма велики, поэтому для экономии дискового пространства в системе OMEGA пользователь и администратор могут иметь общую DDB. В этом случае в файл-директиву включаются координаты документа в общей DDB.

Файл-директива также подписывается пользователем, имя файла-директивы — `<Sign>.DIR`, где `<Sign>` — первые 8 символов электронной подписи документа. Отметим также, что файл-директива необходим только своему администратору и никуда далее по системе документооборота не пересылается.

Таким образом, пользователь отправляет администратору файл с документом, протоколы к нему и файл-директиву. Если же осуществляется прямая отправка другому пользователю, то файл-директива ему не направляется.

Наконец, последний этап подготовки к отправке — это составление объединенного файла, включающего в себя файлы с документом, протоколами и файл-директиву. Если осуществляется отправка через администратора, то готовится один объединенный файл и в него включаются все подготовленные к отправке через администратора файлы с документами, а также прилагающиеся к ним протоколы и файлы-директивы. Если же осуществляется прямая отправка другому пользователю, то отдельный объединенный файл готовится для каждой пары "документ-протокол". Как в первом, так и во втором случаях объединенный файл пользователя помещается в `=OUT_BOX=` — индивидуальный почтовый ящик для исходящей почты.

Имя объединенного файла при отправке через администратора — это наименьший свободный четырехзначный номер в `=OUT_BOX=`. Например, если в `=OUT_BOX=` ранее подготовленных объединенных файлов нет (все уже отправлены), то имя будет 0001, если есть ровно один подготовленный ранее, но не отправленный 0001, то имя будет 0002 и т. д.

При прямой отправке имя объединенного файла будет `<USR><Sign>.<ADM>`, где `<ADMUSR>` — идентификатор получателя, `<Sign>` — первые 5 символов электронной подписи документа.

Подготовка пользователем к отправке электронных документов в автоматическом режиме практически не имеет интерфейса: нужно лишь положить отправляемые файлы в соответствующие каталоги. Все заранее настраивается с помощью описанных в прошлой главе конфигурационных файлов OUT.BAS и WINMAIN.BAS, процедура отправки циклически повторяется, сама подхватывает и отправляет по назначению эти файлы. Режим специально проектировался под типовые банковские операции: рассылку платежных документов, балансов, статистических отчетностей, бухгалтерских форм и т. п. Но часто возникает необходимость отправить нетиповой, разовый документ, например, разослать по филиалам некоторый приказ руководства, инструкцию, предложения, служебную записку и т. п. Для этих целей в системе OMEGA предусмотрен интерактивный режим подготовки электронных документов пользователем (рис. 25.1).

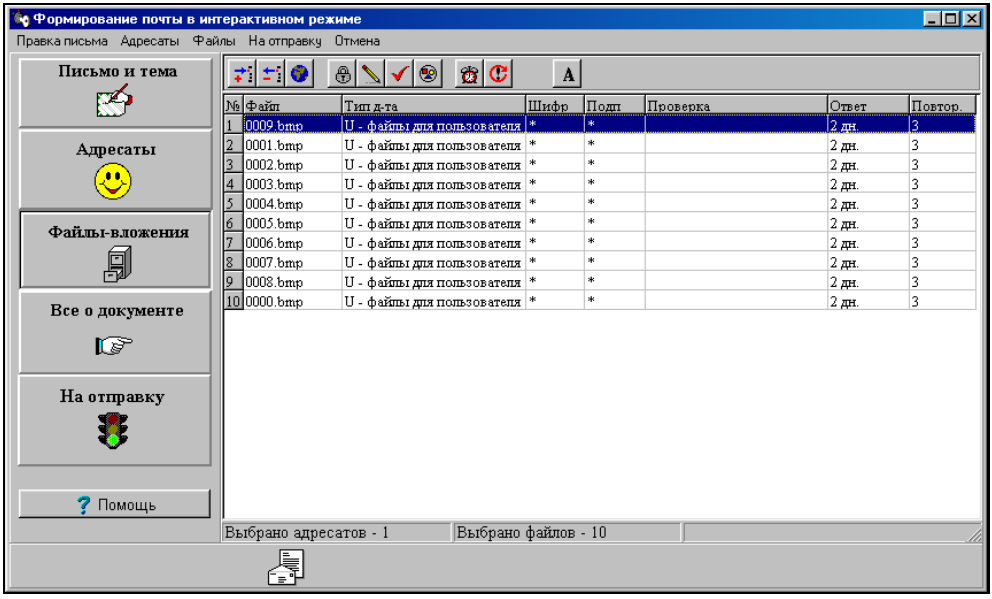

Рис. 25.1. Подготовка электронного документа в интерактивном режиме

Все параметры, характеризующие электронный документ, пользователь задает в этом случае самостоятельно. К таким параметрам относятся:

- файлы, входящие в документ;
- получатели документа;
- письмо и тема к документу;
- тип документа;
- нужно ли документ шифровать;
- нужно ли подписывать отдельные файлы, входящие в документ;
- нужно ли проверять подписи отдельных файлов перед их включением в документ;
- время, через которое должно прийти подтверждение получения документа (MaxWait);
- максимальное количество повторных отправок документа (MaxAttempt).

Из этих параметров специально для интерактивного режима предназначены письмо и тема. Они представляют из себя отдельные файлы с заранее оговоренными именами. Для ввода письма и темы предусмотрен отдельный интерфейс, а в остальном работа с ними происходит точно так же, как и с обычными файлами, входящими в документ.

Обязательным требованием при подготовке документа в интерактивном режиме является наличие в нем хотя бы одного файла и хотя бы одного адресата. Все остальные параметры — необязательные. По умолчанию тип документа — I (интерактивный режим), MaxWait и MaxAttempt выбираются из файла SETUP.

Рассмотрим теперь подготовку документов типа пакет администратором. У них есть несколько отличий от документов пользователей:

- документ типа пакет всегда имеет ровно одного адресата (и, следовательно, ровно один протокол) и не может быть циркулярным;
- для документа типа пакет не создается файл-директива (она не нужна);
- объединенный файл администратора всегда готовится так же, как объединенный файл при прямой отправке пользователем. Имя объединенного файла — `<TRG><Sign>.<SRC>`, где `<TRG>` — идентификатор администрации-получателя, `<SRC>` — идентификатор администрации-отправителя, `<Sign>` — первые 5 символов подписи документа типа пакет;
- после подготовки объединенный файл обязательно шифруется;
- у администратора нет интерактивного режима подготовки документов типа пакет.

Отметим, что вся подготовка электронных документов осуществляется в предварительном режиме. Подготовка и рассылка — это совершенно

разные процедуры. Пользователь при желании может так запрограммировать свой файл WINMAIN.BAS, что подготовка и рассылка будут осуществляться с помощью разных кнопок основного меню. Особенности, связанные с пересылкой подготовленных документов по тому или иному каналу связи, будут подробно рассмотрены в *главе 26* этой книги, а сейчас мы рассмотрим некоторые особенности, связанные с приемом электронных документов, вернее файлов, входящих в электронный документ. При этом предполагаем, что объединенный пакет с документами и протоколами дошел до получателя и находится в его `=IN_BOX=`.

Вот перечень действий, автоматически выполняемых при приеме электронного документа пользователем.

1. Распаковка объединенного пакета в специальном временном каталоге.
2. Для каждого найденного документа поиск протокола к нему.
3. Проверка отправителя, указанного в протоколе, по файлу RK.BAS.
4. Проверка документа на просроченность (15 суток с момента его подписи).
5. Проверка подписи отправителя на документе.
6. Проверка подписи отправителя на протоколе.
7. Проверка подписи своего администратора на протоколе (если была пересылка через администрацию).
8. Проверка соответствия реальной подписи документа той, которая указана в протоколе.
9. Поиск файла-директивы и проверка его подписи, если он существует.
10. Проверка документа на повторяемость по базе данных документов (DDB). Если точно такой же документ обнаружен, то восстанавливается и направляется отправителю протокол его приема, второй раз документ в DDB не заносится, и его обработка на этом заканчивается.
11. Внесение документа в DDB (если не обнаружено повтора).
12. Расшифрование документа, если он был зашифрованный.
13. Разархивирование документа.
14. Проверка реального списка разархивированных файлов по файлу-реестру, обязанному присутствовать в документе.
15. Подпись протокола к документу.
16. Внесение данных о принятых файлах в базу данных реестров (RDB).
17. Внесение записи в базу данных подписей (SDB).
18. Аннулирование в протоколе подписи администратора (если она была).
19. Копирование протокола в специальный временный каталог для отправки его отправителю.

20. Сортировка принятых файлов (если предусмотрена).

21. Удаление принятого документа и протокола из временного каталога.

Процесс осуществления всех этих проверок может быть достаточно длительным, в ходе выполнения отдельных операций может произойти сбой в работе компьютера и программа зависнет или прекратит работу. В этом случае при повторном запуске процесс будет продолжен с того места, на котором прервался.

Для пользователя системы OMEGA предусмотрены два варианта приема входящей информации из `=IN_BOX=`: с помощью основного программного модуля пользователя (процедура `_User_in`) и с помощью специального модуля IN_POSTE.EXE, предназначенного для контроля за состоянием почтового ящика `=IN_BOX=`. Рассмотрим подробнее работу модуля IN_POSTE.EXE.

Начнем с описания алгоритма его работы. Периодически осуществляется просмотр почтового ящика пользователя `=IN_BOX=`, а также, при прямом использовании INTERNET, хоста, задаваемого в файле INTERNET.BAS, и проверяется наличие входящей почты. Периодичность проверки `=IN_BOX=` определяется параметром `=CHECK_INBOX=` файла SETUP, а периодичность проверки хоста — параметром `=SCANNING=` файла INTERNET.BAS *(см. главу 26)*. При этом параметр `=CHECK_INBOX=` задает периодичность проверки `=IN_BOX=` в секундах или минутах, а `=SCANNING=` — через сколько проверок `=IN_BOX=` осуществлять проверку хоста. При этом:

- `=SCANNING= 0` — хост не проверяется вообще;
- `=SCANNING= 1` — хост проверяется при каждой проверке `=IN_BOX=`;
- `=SCANNING= 2` — хост проверяется через одну проверку `=IN_BOX=`;
- `=SCANNING= 3` — хост проверяется через две проверки `=IN_BOX=` и т. д.

При проверке хоста осуществляется поиск на нем входящей почты, удовлетворяющей признакам почты OMEGA. Алгоритм поиска и проверки этих признаков — в точности такой же, что и при выполнении команды `_Internet(R)`, о которой пойдет речь в *главе 26*. При этом найденная своя почта безусловно принимается в `=IN_BOX=`. Если значение `=SCANNING=` отлично от 0, то при запуске модуля IN_POSTE.EXE пользователь должен ввести пароль для доступа к INTERNET (если в INTERNET.BAS строка `=INTERNET_PSW=` принимает значение '*') и пароль для приема почты (если в INTERNET.BAS строка `=RECEIVE_PSW=` принимает значение '*').

Программа IN_POSTE.EXE может работать в многозадачном режиме и осуществлять контроль за поступлением входящей почты, в то время как сам пользователь в другом окне решает другие задачи. Как только в `=IN_BOX=` пользователя будет обнаружена почта, то на экран, поверх всех других окон, будет выдано сообщение, аналогичное приведенному на рис. 25.2.

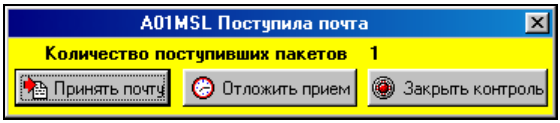

Рис. 25.2. Сообщение о поступлении входящей почты

Кроме того, если в файле SETUP прописана строка =SOUND=, то вместе с этим окном проигрывается мелодия, заданная в файле WAV из этой строки.

При нажатии на кнопку **Принять почту** осуществляется прием поступившей входящей почты из =IN_BOX=. Кнопка **Отложить прием** откладывает принятие решения о приеме на 5 минут. После приема почты на экране появляется сообщение о результатах приема входящей почты, аналогичное приведенному на рис. 25.3.

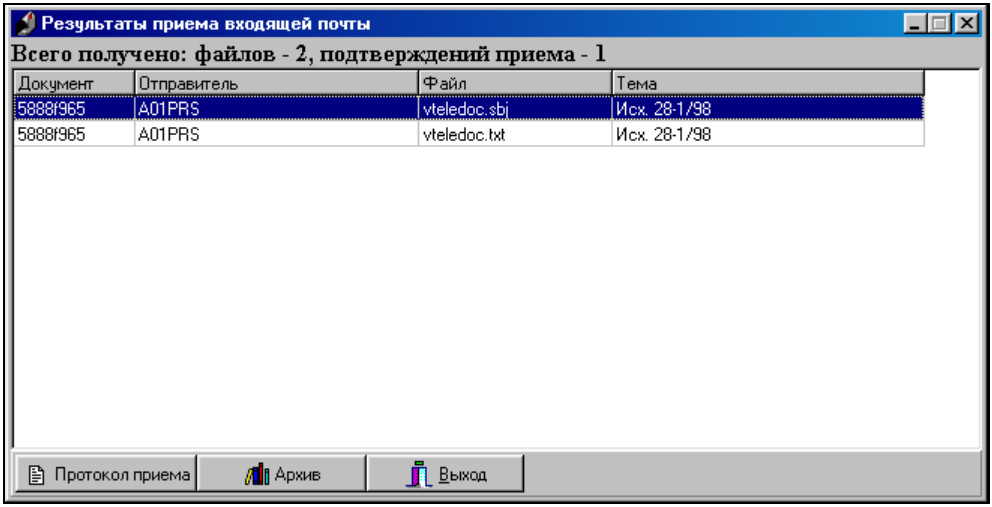

Рис. 25.3. Сообщение о результатах приема почты

Щелкнув два раза мышью по строке с документом, можно получить на экране текст сопроводительного письма (рис. 25.4).

Отметим еще одну особенность приема входящей почты с помощью модуля IN_POSTE.EXE. При получении входящих документов отправителю направляется протокол с подписью получателя. Этот протокол в объединенном файле поступает в =OUT_BOX= пользователя, а если предусмотрено прямое использование Internet, то он сразу же направляется отправителю по Internet.

Таким образом, автоматизированная система OMEGA может, наряду с автоматическим режимом отсылки и приема типовых документов, использоваться в качестве закрытой почты для деловой переписки, обмена конфиденциальной

информацией между руководителями компаний. С помощью АС OMEGA возможно создание удаленных рабочих мест сотрудников компании, например, программистов. Автоматическое подтверждение получения электронного документа, заверенное электронной подписью получателя, является гарантией взаимного контроля между отправителем и получателем.

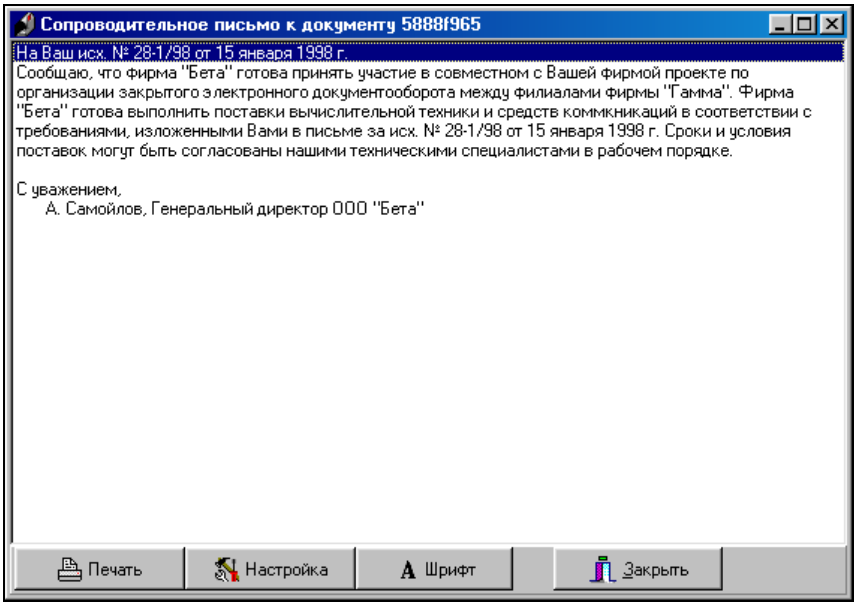

Рис. 25.4. Текст сопроводительного письма, принятого с помощью модуля IN_POSTE.EXE

Глава 26

Пересылка электронных документов

Электронные документы к отправке готовы. Это означает, что созданы объединенные файлы, которые находятся в почтовом ящике для исходящей почты. Далее необходимо осуществить их пересылку, под которой мы будем понимать некоторый способ перемещения объединенного файла из =OUT_BOX= отправителя в соответствующий почтовый ящик =IN_BOX= для входящей почты получателя. Заметим, что почтовые ящики для исходящей и входящей почты =OUT_BOX= и =IN_BOX= обязательно есть у каждого участника системы электронного документооборота, они задаются в основном конфигурационном файле SETUP.

Каким образом осуществляется пересылка? Способов два: либо по локальной сети, либо с использованием электронной почты. В системе OMEGA для доставки электронных документов предусмотрено использование следующих видов электронной почты:

- прямая доставка по Internet с использованием протоколов SMTP и POP3;
- Pegasus Mail;
- Sprint Mail;
- альтернативная почта.

Отметим следующие особенности. Пользователь может связываться со своей администрацией как по локальной сети (должны быть общие каталоги =OUT_BOX= и =IN_BOX=, к которым имеют доступ как пользователь, так и администратор), так и по электронной почте. В последнем случае такого пользователя будем называть удаленным. Удаленный пользователь имеет доступ только к своим индивидуальным =OUT_BOX= и =IN_BOX=, а в администрации исключительно для этого пользователя также зарезервированы другие =OUT_BOX= и =IN_BOX=, к которым имеет доступ только администратор. Электронная почта в этом случае переносит исходящую почту из =OUT_BOX=, доступного пользователю, в =OUT_BOX=, доступный администратору, а также

входящую почту из =IN_BOX=, доступного администратору, в =IN_BOX=, доступный пользователю.

Таким образом, для каждого своего пользователя в администрации имеется индивидуальная пара почтовых ящиков =OUT_BOX= и =IN_BOX=. Кроме того, в администрации также имеется пара почтовых ящиков (ссылка на них задается строкой =POSTE= в файле SETUP), предназначенных для отправки и приема внешних пакетов. К этим почтовым ящикам имеет доступ только сам администратор. Задача электронной почты при пересылке внешних пакетов — доставить пакет из исходящего почтового ящика администрации-отправителя во входящий почтовый ящик администрации-получателя.

Отметим некоторые общие моменты, связанные с использованием электронной почты в системе OMEGA.

Характерным признаком почтовых отправлений, подготовленных с помощью OMEGA, является наличие в них в качестве параметра Subject строки OMEGA. По этому признаку, в частности, программа, осуществляющая доставку почты с хоста Internet, отличает свои сообщения от чужих, происходит разделение на свою — чужую при разборке почты, полученной с помощью Pegasus Mail и Sprint Mail.

Прямая доставка по Internet с использованием протоколов SMTP и POP3

Для осуществления прямой доставки по Internet с использованием протоколов SMTP и POP3 необходимо, чтобы компьютер, на котором работает пользователь или администратор, имел доступ к Internet. Это может быть либо доступ через локальную сеть, либо удаленный доступ через модем. Параметры доступа (host, login, password и т. п.) задаются в конфигурационном файле INTERNET.BAS, состав которого мы рассмотрим чуть ниже. Адреса, необходимые для отправки, выбираются из конфигурационного файла VMAIL.BAS, содержащего адреса участников системы документооборота при использовании Internet или Pegasus Mail. Отметим некоторые особенности процедуры прямого использования Internet в системе OMEGA.

- При использовании удаленного доступа к сети через модем имеется возможность организовать сценарий автоматического дозвона до сервера, а также отключаться от него по окончании сеанса. Это позволяет организовать полностью автоматические циклы отправки и приема электронных документов без участия оператора и избегать лишних затрат, оптимизируя время доступа к сети.
- При приеме с хоста забирается только своя входящая почта, т. е. имеющая в качестве параметра Subject значение OMEGA. Чужая почта, в зависимости от настройки конфигурационного файла INTERNET.BAS, либо остается на хосте без изменений, либо удаляется.

- При приеме осуществляется автоматическое распознавание способа подготовки своей почты: либо с помощью модулей прямого использования Internet, либо с помощью Pegasus Mail. Тем самым обеспечивается совместимость почты Pegasus Mail с прямым использованием Internet.

Pegasus Mail

Для использования Pegasus Mail в файле SETUP пользователя или администрации необходимо указать параметры `=PMAIL.EXE=` и `=PMAIL.IN=`, задающие ссылку на исполняемый файл PMAIL.EXE почты Pegasus Mail и каталог для входящих файлов с расширением CNM, присутствующих при использовании почты Pegasus Mail. Адреса выбираются из файла VMAIL.BAS.

Sprint Mail

Почта Sprint Mail имеет ряд специфических особенностей. Для ее функционирования используются несколько собственных почтовых ящиков, являющихся подкаталогами в каталоге, задаваемом строкой `=SPRINT=` файла SETUP. Для того чтобы отправить сообщение по Sprint Mail, необходимо в таком почтовом ящике OUT.BOX создать файл-конверт с расширением ASC и прописать в нем имя отправляемого файла-вложения. В системе OMEGA всегда файл-вложение, перед отправкой его по Sprint Mail, копируется в подкаталог OMEGA.BOX каталога `=SPRINT=` и ссылка в конверте дается уже на файл в этом подкаталоге. Отправленные конверты переносятся в SENT.BOX, неотправленные остаются в OUT.BOX или переносятся в REJECTED.BOX. На этом построена система контроля за отправкой документов по Sprint Mail: осуществляется просмотр OMEGA.BOX, для каждого найденного в нем файла пытаемся найти соответствующий ему конверт. Если конверт найден в SENT.BOX, то документ считается отправленным, в противном случае документ считается неотправленным.

Адреса получателей при использовании Sprint Mail выбираются из конфигурационного файла SPRINT.BAS.

Альтернативная почта

Все перечисленные виды электронной почты имеют одно общее свойство: в них для пересылки сообщения к нему добавляется специальная служебная часть, содержащая адреса отправителя и получателя, тему сообщения и другую подобную информацию. Под альтернативной почтой мы будем понимать такой способ пересылки сообщения, когда никакой дополнительной служебной части к нему на этапе подготовки к отправке не добавляется.

Действия по отправке сообщения по альтернативной почте сводятся к следующей схеме: положить сообщение на отправку в определенный каталог и вызвать некоторую процедуру отправки (например, программу пересылки по модему). Выбор каталога осуществляется в зависимости от получателя сообщения, в системе OMEGA перечень таких каталогов задается в конфигурационном файле MAIL.BAS.

Таким образом, для каждого участника системы электронного документооборота возможности его связи с другими участниками по электронной почте задаются с помощью конфигурационных файлов VMAIL.BAS, SPRINT.BAS и MAIL.BAS, в которых указываются адреса (или каталоги) для связи. Однако для каждого пользователя существует еще конфигурационный файл RK.BAS, в котором перечислены все другие пользователи, с которыми ему разрешена связь по системе OMEGA. Если какой-либо из пользователей, перечисленных в RK.BAS, присутствует еще и в VMAIL.BAS, SPRINT.BAS или MAIL.BAS, то считается, что связь с этим пользователем — прямая, минуя администратора. В противном случае связь поддерживается через администратора.

В каждой администрации также существует конфигурационный файл TRANSIT.BAS, в котором перечислены все администрации, с которыми разрешена связь. Однако, в отличие от пользователя, каждая из администраций, перечисленных в TRANSIT.BAS, должна присутствовать хотя бы в одном из конфигурационных файлов VMAIL.BAS, SPRINT.BAS или MAIL.BAS, в противном случае переслать ей внешние пакеты будет невозможно.

Отметим, что путем настройки сценария работы OMEGA можно использовать несколько попыток отправки по различным типам электронной почты. При приеме каждого входящего документа по базе данных документов осуществляется его проверка на повторяемость, поэтому повторные отправки не приведут в этом случае к нежелательным последствиям (например, к приему одного и того же платежного поручения дважды).

Рассмотрим подробнее упоминавшиеся конфигурационные файлы MAIL.BAS, SPRINT.BAS и VMAIL.BAS. Они могут быть как у пользователя, так и у администрации. Рассмотрим эти файлы у пользователя, поскольку у администрации они строятся аналогично.

- Файл MAIL.BAS содержит каталоги пользователей, необходимые при использовании альтернативной почты (почта, в которой не требуется создания конверта);
- Файл SPRINT.BAS содержит адреса участников системы электронного документооборота в Sprint Mail и необходим при использовании Sprint Mail;
- Файл VMAIL.BAS содержит адреса участников системы электронного документооборота в Internet и необходим при использовании Internet напрямую или через Pegasus Mail.

У всех этих файлов есть некоторые общие особенности.

- Эти файлы индивидуальны для каждого пользователя, первым в них всегда указывается идентификатор их владельца.
- С каждым пользователем, прописанным хотя бы в одном из этих файлов, связь осуществляется напрямую, минуя администрацию. Все пользователи, указанные в этих файлах, должны быть прописаны в файле RK.BAS. Связь с пользователями, не прописанными ни в одном из этих файлов, но присутствующими в файое RK.BAS, осуществляется через администрацию.
- В некоторых из этих файлов может быть прописана собственная администрация. В этом случае пользователь считается удаленным и связь с администрацией осуществляется по электронной почте.

Перейдем теперь к содержанию этих файлов. Начнем с файла MAIL.BAS, предназначенного для альтернативной почты. Напомним, что под альтернативной почтой понимается пересылка сообщения адресату без создания служебной части (специального конверта, как это сделано в Sprint Mail, или заголовка, как при пересылке по Internet). Для отправления сообщения нужному адресату по альтернативной почте необходимо поместить это сообщение в специальный каталог, предназначенный для исходящих сообщений только для данного адресата, и вызвать некоторую программу, осуществляющую пересылку (например, программу пересылки по телефонному каналу с помощью модема). Предполагается, что при осуществлении приема входящей почты с помощью аналогичной программы, она будет рассортирована по специальным каталогам в зависимости от ее отправителя. Конфигурационный файл MAIL.BAS содержит перечень таких каталогов для отправки и приема сообщений по альтернативной почте.

В первой строке КФ MAIL.BAS всегда должен быть указан идентификатор его владельца, в последующих строках указываются адресаты, с которыми осуществляется связь по альтернативной почте, и соответствующие каталоги. К идентификатору адресата добавляется признак `_OUT:`, если это каталог для исходящей почты, `_IN:` — для входящей. Не менее чем через один пробел от идентификатора с признаком следует полное имя самого каталога.

- Пример файла MAIL.BAS.

```
ALFDIR:
BETDIR_OUT: C:\OMEGA\MAIL\BETDIR\OUT
BETDIR_IN: C:\OMEGA\MAIL\BETDIR\IN
GAMDIR_OUT: C:\OMEGA\MAIL\GAMDIR\OUT
GAMDIR_IN: C:\OMEGA\MAIL\GAMDIR\IN
BNKPAY_OUT: C:\OMEGA\MAIL\BNKPAY\OUT
```

  Из этого примера видно, что для некоторых пользователей в файле MAIL.BAS может быть указан только один каталог на отправку (или

только один каталог на прием). В данном примере для пользователя BNKPAY указан только каталог на отправку, это означает, что отправка документов к нему может осуществляться по альтернативной почте, а прием документов от него происходит другим способом (через администрацию, Sprint Mail или Internet).

Для удаленного пользователя в файле MAIL.BAS также может быть указана его администрация, если связь с ней осуществляется по альтернативной почте.

- Пример удаленного пользователя BETDIR.

```
BETDIR:
ALFDIR_OUT: C:\OMEGA\MAIL\ALFDIR\OUT
ALFDIR_IN: C:\OMEGA\MAIL\ALFDIR\IN
BET_OUT: C:\OMEGA\MAIL\BET\OUT
BET_IN: C:\OMEGA\MAIL\BET\IN
```

Рассмотрим теперь КФ SPRINT.BAS. В нем содержатся адреса участников системы ЭД, с которыми осуществляется связь по Sprint Mail. Файл состоит из строк, в начале строки указывается идентификатор участника системы ЭД, далее — символ :, затем, не менее чем через 1 пробел, — сам адрес. В первой строке должен быть указан владелец и его адрес в Sprint Mail. Для удаленного пользователя в SPRINT.BAS может быть указана его администрация, если связь с ней осуществляется по Sprint Mail. Примеры.

- SPRINT.BAS для пользователя `ALFDIR`:

```
ALFDIR: ALFA.SPRINT
BETDIR: B-DIRECTOR.SPRINT
BNKPAY: BANK.SPRINT
```

- SPRINT.BAS для удаленного пользователя `BETDIR`:

```
BETDIR: B-DIRECTOR.SPRINT
ALFDIR: ALFA.SPRINT
BET: BETA.SPRINT
```

Наконец, последний из этих трех КФ — VMAIL.BAS. В нем указаны адреса для пересылки по Internet. Правила построения — те же, что и SPRINT.BAS. Примеры.

- VMAIL.BAS для пользователя `ALFDIR`:

```
ALFDIR: alfa@alfahost.ru
BETDIR: betadir@betahost.ru
BNKPAY: bank@bankhost.ru
```

- VMAIL.BAS для удаленного пользователя `BETDIR`:

```
BETDIR: betadir@betahost.ru
```

```
ALFDIR: alfa@alfahost.ru
BET: beta@betahost.ru
```

Отметим, что только включения адресата в некоторые из файлов MAIL.BAS, SPRINT.BAS и VMAIL.BAS еще недостаточно для осуществления связи с ним. Помимо этого в сценарии работы необходимо прописать вызов процедуры осуществления отправки, приема (или отправки и приема вместе) по соответствующей почте.

Остановимся теперь подробнее на прямой доставке по Internet с использованием протоколов SMTP и POP3. Начнем с конфигурационного файла INTERNET.BAS.

Конфигурационный файл INTERNET.BAS предназначен для задания параметров, необходимых для осуществления прямого выхода в Internet при отправке и приеме электронных документов с помощью типовых протоколов SMTP и POP3 соответственно. Этот конфигурационный файл, как правило, создается и редактируется самим пользователем, для чего в комплекте программных модулей имеется модуль I_CONFIG.EXE, позволяющий самостоятельно задавать параметры этого файла.

Ниже в табл. 26.1 приводится состав и назначение параметров файла INTERNET.BAS.

Таблица 26.1. *Состав и назначение параметров конфигурационного файла INTERNET.BAS*

	Название параметра	Назначение параметра	Значение по умолчанию
1	=ID=	Идентификатор пользователя	
2	=INTERNET_IN=	Каталог для входящих Internet-пакетов	<SetupDir>\INTERNET.IN
3	=INTERNET_OUT=	Каталог для исходящих Internet-пакетов	<SetupDir>\INTERNET.OUT
4	=HOST=	Адрес хоста Internet. Допускается либо явное задание IP-адреса (типа 102.219.31.65), либо его имя (типа hotmail.ru)	Часть адреса пользователя, прописанного в первой строке файла VMAIL.BAS, после символа '@'
5	=RECEIVE_HOST=	Адрес хоста Internet для приема входящей почты	Совпадает с =HOST=
6	=SMTP_PORT=	Номер TCP-порта сервера, на который устанавливается SMTP-сервис	25

Таблица 26.1 *(продолжение)*

	Название параметра	Назначение параметра	Значение по умолчанию
7	=POP3_PORT=	Номер TCP-порта сервера, на который устанавливается POP3-сервис	110
8	=INTERNET_ID=	Идентификатор пользователя в сети Internet (его login)	Часть адреса пользователя, прописанного в первой строке файла VMAIL.BAS, до символа @
9	=RECEIVE_ID=	Идентификатор почтового ящика пользователя на хосте для приема почты (его login для входящей почты)	Совпадает со значением строки =INTERNET_ID=
10	=INTERNET_NAME=	Имя пользователя в сети Internet (включается в служебную часть отправляемых по Internet сообщений)	Unknown user OMEGA
11	=INTERNET_PSW=	Пароль для доступа к хосту Internet. Задается или явно, или одним символом '*'. В этом случае пароль вводится с клавиатуры	*
12	=RECEIVE_PSW=	Пароль для приема почты по Internet. Задается явно или одним символом '*'. В этом случае пароль вводится с клавиатуры	Совпадает со значением строки =INTERNET_PSW=
13	=TIME_OUT=	Время (в миллисекундах), отпущенное на установление связи. При значении 0 время не ограничено	0
15	=DEL_HOST=	Индикатор удаления чужой почты с хоста (TRUE — чужая почта удаляется, FALSE — нет)	FALSE
15	=SCANNING=	Показатель сканирования хоста программой VTD.EXE контроля за поступлением входящей почты. При значении 0 хост не сканируется	0

Таблица 26.1 *(окончание)*

	Название параметра	Назначение параметра	Значение по умолчанию
16	=REMOTE_ ACCESS=	Название учетной записи почты, используемой для удаленного доступа к Internet	
17	=MAX_ATTEMPT=	Максимальное количество попыток дозвониться до хоста при удаленном доступе	
18	=WAIT=	Время задержки между попытками дозвона при удаленном доступе (в секундах)	
19	=DISCONNECT_ LINE=	Индикатор разрыва соединения после окончания сеанса связи при удаленном доступе. TRUE — соединение разрывается, FALSE — остается	TRUE
20	=PASSWORD_ CONTROL=	Контрольная комбинация для проверки правильности ввода пароля для обращения к хосту	
21	=RECEIVE_ PASSWORD_ CONTROL=	Контрольная комбинация для проверки правильности ввода пароля для приема почты	

Необходимые условия использования пользователем прямого выхода в Internet в системе OMEGA:

- В SoftDir должен быть специализированный программный модуль INTERNET.EXE;
- В SetupDir должен быть конфигурационный файл INTERNET.BAS с параметрами для прямого выхода в Internet, а также файл VMAIL.BAS с адресами электронной почты;
- В сценарии работы должна быть задана команда `_Internet(param)`, где `param` — параметр — латинская буква, определяющая операции, выполняемые почтовой программой. Перечень всевозможных параметров задается в приводимой ниже табл. 26.2.

Таблица 26.2. *Значения параметров при использовании команды* `_Internet`

	Значение параметра	Формирование пакетов на отправку в =INTERNET_OUT=	Отправка	Прием	Распаковка принятых в =INTERNET_IN= пакетов
1	O	X			
2	S	X	X		
3	R			X	X
4	P				X
5	A	X	X	X	X

Разницы между большими и малыми буквами в параметрах нет.

Здесь необходимо дать пояснения о том, как в системе OMEGA работает механизм прямого выхода в Internet. Начнем с отправки.

Просмотр =OUT_BOX= пользователя и формирование из находящихся в нем объединенных файлов специальных Internet-пакетов в каталоге =INTERNET_OUT=. При этом:

- Объединенные файлы, предназначенные для администрации, переименовываются, к 4-значному имени добавляется приставка <SS><HH>.<MM>, где MMSSHH — текущее время: минуты, секунды, сотые доли секунды. Это делается для того, чтобы избежать затирания файлов при приеме;
- Все объединенные файлы, предназначенные одному и тому же адресату, складываются в один Internet-пакет;
- Internet-пакет, предназначенный для своей администрации, имеет имя <ADM>.<USR>, где ADMUSR — идентификатор пользователя, Internet-пакет, предназначенный для прямой отправки другому пользователю, имеет имя <TARGET>, где TARGET — идентификатор получателя;
- Объединенные файлы, включенные в Internet-пакет, из =OUT_BOX= пользователя удаляются.

Отметим, что для формирования Internet-пакетов для администрации необходимо, чтобы в файле VMAIL.BAS был указан адрес администрации (строка <ADM>:), а для формирования Internet-пакетов для пользователя TARGET в файле VMAIL.BAS должен быть указан его адрес (строка <TARGET>:).

При параметре O работа команды `_Internet(O)` на этом заканчивается.

- Соединение с Internet-хостом.
- Отправка подготовленных Internet-пакетов из каталога =INTERNET_OUT=. Правила составления Internet-сообщения для отправки — отдельные строки заголовка (header) и тела (body) сообщения — следующие:
  - **header: FromAdress** — адрес пользователя — первая строка из файла VMAIL.BAS;

- **header: FromName** — строка =INTERNET_NAME= из файла INTERNET.BAS;
- **header: ToAdress** — адрес получателя (администрации или пользователя) из файла VMAIL.BAS;
- **header: Subject** — OMEGA <ADMUSR> (для отправки в администрацию) и просто OMEGA (для отправки другому пользователю);
- **body:** OMEGA.

В Internet-сообщении всегда ровно один файл-вложение (Attachments). Им является Internet-пакет, предназначенный для данного получателя. Используемая кодировка вложения — MIME.

При параметре S работа команды _Internet(S) на этом заканчивается.

Заканчивая рассмотрение отправки, отметим еще такие особенности:

- при параметре A (all — все операции) отправка осуществляется первой, до приема;
- при отсутствии Internet-пакетов на отправку соединение с Internet-хостом и вызов процедуры отправки не осуществляются.

Перейдем теперь к рассмотрению приема.

1. Соединение с Internet-хостом.
2. Просмотр сообщений на хосте и поиск среди них тех, в которых в заголовке (header) строка **Subject** начинается с OMEGA. Если это не так, то сообщение считается чужим, его прием не осуществляется. Чужие сообщения остаются на хосте, если в качестве параметра =DEL_HOST= в файле INTERNET.BAS задано значение FALSE. Если же =DEL_HOST= TRUE, то чужие сообщения с хоста удаляются.
3. В найденном сообщении, помимо проверки строки **Subject**, осуществляется проверка его возможной отправки с помощью Pegasus Mail. Для этого в заголовке осуществляется поиск строки **X-mailer: Pegasus Mail**. При наличии такой строки считается, что сообщение было подготовлено с помощью Pegasus Mail, его тело (body) записывается в каталог =INTERNET_IN= под именем PEGASUS и обрабатывается в соответствии со специальным алгоритмом обработки почты Pegasus Mail.
4. Если сообщение не было подготовлено с помощью Pegasus Mail, то проверяется число вложений (Attachments). Если оно не равно 1, то сообщение считается чужим, его прием не осуществляется, оно обрабатывается так же, как и в п. 2. Если число вложений равно 1, то сообщение считается своим и его тело принимается в каталог =INTERNET_IN=. Отметим, что различные сообщения от одного и того же адресата не затирают друг друга в каталоге =INTERNET_IN=: при приеме, если файл с точно таким же именем уже существует, то к его имени автоматически добавляются служебные символы.

5. После приема всех сообщений в каталог =INTERNET_IN= осуществляется их разборка. Проверяется имя файла с Internet-пакетом и специальные признаки Internet-пакета. Имя файла должно либо начинаться с <ADMUSR> и не иметь расширения (Internet-пакеты от других пользователей), либо начинаться с <ADM> и иметь расширение USR (Internet-пакеты от администрации для удаленного пользователя), где ADMUSR — идентификатор пользователя. Если имя файла другое, но признаки Internet-пакета обнаружены, то такой пакет считается транзитным и просто перекладывается в каталог =INTERNET_OUT=. Если признаков Internet-пакета не обнаружено, то вне зависимости от имени файла он считается мусором и переносится (если такая существует) в каталог =BASKET= (корзина для мусора, задаваемая в файле SETUP), а из =INTERNET_IN= удаляется. Свои Internet-пакеты раскрываются, и их содержание переносится в =IN_BOX= пользователя. После раскрытия пакеты из =INTERNET_IN= удаляются.

Отметим, что при параметре P (paket) соединения с хостом не происходит, а работа команды _Internet(P) осуществляется только по приведенному выше п. 5.

Остановимся теперь на некоторых особенностях прямого использования Internet в системе OMEGA.

Организация дозвона и минимизация времени доступа к хосту

Во многих случаях пользователи используют удаленный доступ к сети по модему через провайдеров Internet. В этой ситуации для отправки и приема почты по Internet необходимо осуществлять операции дозвона до хоста, а с учетом того, что система OMEGA в основном ориентирована на автоматические режимы отправки и приема, операции дозвона также должны выполняться в автоматическом режиме без вмешательства оператора. Для этой цели в файле INTERNET.BAS предусмотрена возможность организации сценария дозвона, в котором задаются максимальное число попыток дозвона (=MAX_ATTEMPT=) и время задержки между попытками (=WAIT=). В случае, если при доступе к хосту требуется ввод дополнительных данных с клавиатуры (login, password), то для этого можно использовать типовые SCRIPT-файлы. Ниже приводится пример простого SCRIPT-файла.

```
proc main

   waitfor "login:" until 10
   if $SUCCESS then
   transmit $USERID,raw
   transmit "^M"
   endif
```

```
waitfor "password:" until 10

if $SUCCESS then
transmit $PASSWORD,raw
transmit "^M"
endif

endproc
```

Подключение SCRIPT-файла осуществляется при настройке параметров удаленного доступа к сети.

Для оптимизации времени обращения к сети (и, следовательно, затрат по оплате услуг провайдера) в системе OMEGA предусмотрена возможность автоматического отключения от сети после выполнения операций по отправке и приему. Для этого в файле INTERNET.BAS необходимо в качестве параметра `=DISCONNECT_LINE=` задать значение `TRUE`.

Пароли для доступа к хосту и для приема почты

Пароли для доступа к хосту и для приема почты могут быть заданы явно в файле INTERNET.BAS. В этом случае в строках `=INTERNET_PSW=` и `=RECEIVE_PSW=` необходимо в явном виде прописать пароль, тогда каждый раз при обращении к Internet пароль будет считываться из этих строк и никаких дополнительных операций по вводу пароля не требуется. В случае, если в строке `=INTERNET_PSW=` или `=RECEIVE_PSW=` указан символ `*`, то пароль необходимо вводить с клавиатуры. Однако и в этом случае пароли можно ввести один раз в начале цикла, если в сценарии работы задать команду `_InputInternetPassword`. При этом пароли вводятся в оперативную память и используются автоматически при каждом обращении к Internet. Проверку правильности ввода пароля можно осуществить заранее, до обращения к хосту, если в файле INTERNET.BAS задать строки `=PASSWORD_CONTROL=` и `=RECEIVE_PASSWORD_CONTROL=`.

Если строка `=INTERNET_PSW=` в файле INTERNET.BAS отсутствует и команда `_InputInternetPassword` в сценарии не задавалась, то обращение к Internet-хосту осуществляется без пароля.

Особенности использования Pegasus Mail

Необходимые условия использования почты Pegasus Mail:

- В файле SETUP должны быть заданы строки `=PMAIL.EXE=` со ссылкой на исполняемый файл PMAIL.EXE почты Pegasus Mail и `=PMAIL.IN=`

со ссылкой на почтовый ящик для входящей почты в системе Pegasus Mail (каталог, в который поступают файлы с расширением CNM).

- В SetupDir должен быть конфигурационный файл VMAIL.BAS с адресами электронной почты. Если с помощью Pegasus Mail будет осуществляться прием сообщений, отправленных путем прямого выхода в Internet, то в SetupDir также должен быть конфигурационный файл INTERNET.BAS, в котором прописана строка `=INTERNET_IN=`.
- В сценарии работы должна быть задана команда `_Vmail(param)`, где `param` — параметр — латинская буква, определяющая операции, выполняемые почтовой программой. Перечень всевозможных параметров задается в приводимой ниже табл. 26.3.

Таблица 26.3. *Значения параметров при использовании команды* `_Vmail`

	Значение параметра	Отправка	Код успешного выполнения отправки	Прием	Удаление чужой входящей почты в `=PMAIL.IN=`
1	s	x	0		
2	S	x	1		
3	r			x	
4	R			x	x
5	a	x	0	x	
6	A	x	1	x	
7	f	x	0	x	x
8	F	x	1	x	x

Как видно из этой таблицы, есть различие между большими и малыми буквами в параметре.

Рассмотрим теперь механизм работы Pegasus Mail у пользователя OMEGA. Начнем с отправки. Если осуществляется отправка объединенных файлов от удаленного пользователя своему администратору, то к четырехзначному имени объединенного файла добавляется `<SS><HH>.<MM>`, где `MMSSHH` — текущее время: минуты, секунды, сотые доли секунды. Для каждого найденного в `=OUT_BOX=` объединенного файла поиск его адресата в VMAIL.BAS. Если адресат найден, то объединенный файл сразу же отправляется ему с помощью вызова исполняемого файла `=PMAIL.EXE=` со следующей строкой параметров:

```
<=PMAIL.EXE=> -t {ADRESS} -f {FILENAME} -s {SUBJECT} -b
```

Здесь в фигурных скобках указаны переменные величины, где:

- {ADRESS} — адрес получателя, выбираемый из файла VMAIL.BAS;
- {FILENAME} — имя отправляемого объединенного файла;
- {SUBJECT} — строка **Subject:** в заголовке сообщения имеет значение OMEGA при прямой отправке другому пользователю и OMEGA <ADMUSR>, где ADMUSR — идентификатор пользователя, при отправке в свою администрацию.

При успешно выполненной отправке (определяется по коду успешного выполнения) объединенный файл из =OUT_BOX= пользователя удаляется. При параметрах a, A, f, F, когда осуществляется отправка и прием, отправка всегда происходит первой.

Отметим одно существенное отличие почты Pegasus Mail от прямого использования Internet в системе OMEGA: в Pegasus Mail осуществляется отправка каждого объединенного файла в отдельности, независимо от других объединенных файлов, предназначенных тому же адресату, в то время как при прямом использовании Internet все объединенные файлы, предназначенные одному адресату, складываются в один Internet-пакет.

Прием входящей почты.

- Просмотр каталога =PMAIL.IN= и поиск в нем файлов с расширением CNM;
- Для каждого найденного файла проверка признаков его отправки с помощью Pegasus Mail по системе OMEGA. Такими признаками являются:
  - наличие в заголовке (header) строки **Subject:**, начинающейся с OMEGA;
  - наличие признаков начала и конца тела (body) сообщения: строка, соответствующая началу, начинается со слова begin, конец — со слова end. Если файл удовлетворяет этим условиям, то его тело, преобразованное в обычный файл, записывается в =IN_BOX= пользователя, а сам файл из =PMAIL.IN= удаляется.
- Если файл не удовлетворяет хотя бы одному из приведенных выше признаков отправки с помощью Pegasus Mail, то проверяются признаки его отправки по системе OMEGA с помощью прямого использования Internet. Такими признаками являются:
  - наличие в заголовке (header) строки **Subject:**, начинающейся с OMEGA;
  - наличие в одной из строк заголовка слова filename, за которым следует имя файла-вложения.

Если эти признаки выполнены, то тело сообщения преобразуется в обычный файл и записывается в каталог =INTERNET_IN=, указанный в файле

INTERNET.BAS. При этом признаком начала тела сообщения считается строка, в которой есть не менее 50 подряд идущих символов без пробелов и нулей. К имени файла в каталоге =INTERNET_IN= добавляются вспомогательные служебные символы (номер файла в каталоге) с тем, чтобы исключить возможность затирания предыдущих файлов от тех же адресатов. Таким образом, в =INTERNET_IN= попадают Internet-пакеты, подготовленные при отправке. После этого для приема их содержания в =IN_BOX= пользователя можно использовать команду _Internet(P). После успешного приема файл из =PMAIL.IN= удаляется. Если файл не был принят, то он считается чужим. Чужие файлы или остаются без изменений, или удаляются из =PMAIL.IN= в зависимости от значения параметра.

Особенности использования Sprint Mail

Необходимые условия использования почты Sprint Mail.

- В файле SETUP должна быть строка =SPRINT= со ссылкой на каталог, содержащий почтовые ящики почты Sprint Mail;
- Должны быть созданы и помещены в SoftDir следующие BAT-файлы, осуществляющие запуск почты Sprint Mail:
  - **SPRINT_S.BAT**, содержащий командную строку запуска почты Sprint в режиме только отправки;
  - **SPRINT_R.BAT**, содержащий командную строку запуска почты Sprint в режиме только приема;
  - **SPRINT_A.BAT**, содержащий командную строку запуска почты Sprint в режиме отправки и приема.
- В сценарии работы должна быть задана команда _Sprint(param), где param — параметр — латинская буква, определяющая операции, выполняемые почтовой программой. Перечень всевозможных параметров задается в приводимой ниже табл. 26.4.

Рассмотрим теперь механизм отправки исходящей почты. Для этого в каталоге =SPRINT= создается специальный подкаталог OMEGA.BOX, в который будут записываться файлы-вложения, созданные системой OMEGA и подлежащие отправке по Sprint Mail. Кроме того, предполагается, что в каталоге =SPRINT= уже существуют подкаталоги, являющиеся стандартными почтовыми ящиками почты Sprint Mail: OUT.BOX (для неотосланной исходящей почты), IN.BOX (для входящей почты), SENT.BOX (для отосланной исходящей почты), REJECTED.BOX (для исходящей почты, при отсылке которой произошли ошибки).

Таблица 26.4. *Значения параметров при использовании команды* `_Sprint`

	Значение параметра	Подготовка к отправке по Sprint-Mail (подготовка конверта и приложения в OMEGA.BOX)	Отправка по Sprint-Mail (вызов BAT-файла)	Прием по Sprint-Mail (вызов BAT-файла)	Перенос из IN.BOX почты Sprint-Mail в IN.BOX OMEGA	Удаление чужой входящей почты из IN.BOX почты Sprint-Mail
1	o	x				
2	O	x				
3	s	x	x			
4	S	x	x			
5	r			x	x	
6	R			x	x	x
7	p				x	
8	P				x	x
9	t	x			x	
10	T	x			x	x
11	a	x	x	x	x	
12	A	x	x	x	x	x

Разница между большими и малыми буквами в параметре есть.

Итак, при отправке почты осуществляются.

- Если осуществляется отправка объединенных файлов от удаленного пользователя своему администратору, то к четырехзначному имени объединенного файла добавляется `<SS><HH>.<MM>`, где `MMSSHH` — текущее время: минуты, секунды, сотые доли секунды.
- Для каждого найденного в `=OUT_BOX=` объединенного файла поиск его адресата в SPRINT.BAS. Если адресат найден, то объединенный файл сразу же переносится в `OMEGA.BOX` почты Sprint Mail и из `=OUT_BOX=` пользователя удаляется.

- Для перенесенного в OMEGA.BOX объединенного файла создание в подкаталоге <=SPRINT=>\OUT.BOX конверта (файла с расширением ASC) для пересылки по почте Sprint Mail. Формат файла с конвертом следующий:
  - ***From:*** адрес отправителя (первая строка в SPRINT.BAS);
  - ***To:*** адрес получателя (определяется из SPRINT.BAS);
  - ***CC:*** (пустая строка);
  - ***Subj:*** OMEGA <ADMUSR>, где ADMUSR — идентификатор пользователя, при отправке своему администратору, OMEGA при отправке другому пользователю напрямую;
  - ***Text:*** (пустая строка);
  - *****BINARY***** (служебная строка);
  - ***<Filename>*** Имя объединенного файла-вложения из каталога <=SPRINT=>\OMEGA.BOX.
- При параметре O или o процедура закончена. При параметре S или s вызов BAT-файла SPRINT_S.BAT, при параметре A или a — SPRINT_A.BAT.

Отметим, что здесь, как и везде ранее в OMEGA, при параметре A или a отправка осуществляется первой.

Заметим еще одну особенность использования почты Sprint Mail. При подготовке к отправке осуществляется перенос объединенного файла из =OUT_BOX= в <=SPRINT=>\OMEGA.BOX, и тем самым этот объединенный файл становится недоступным для отправки по другим почтовым системам. Если по каким-либо причинам отправка по Sprint Mail оказалась неудачной, то этот файл необходимо вернуть на место (в =OUT_BOX= пользователя) с тем, чтобы попытаться отправить его по другим почтовым системам. Эта операция осуществляется с помощью процедуры _Spr_cont, но о ней мы поговорим чуть позже.

Прием по Sprint Mail.

- При параметре R или r запуск BAT-файла SPRINT_R.BAT.
- Проверка каталога <=SPRINT=>\IN.BOX. Для каждого найденного в ней файла-конверта (*.ASC) проверка в нем наличия следующих признаков:
  - строки ***Subj:***, начинающейся с OMEGA;
  - строки ***This is a binary file:***;
  - если имя файла-вложения, прописанного в конверте после строки ***This is a binary file***, имеет длину 12, то оно должно быть

<USR><NNNNN>.<ADM>, где ADMUSR — идентификатор пользователя, NNNNN – произвольные символы;

- в <=SPRINT=>\IN.BOX вместе с файлом-конвертом *.ASC должен быть файл-вложение с тем же именем, но с расширением B01;
- если файл-конверт удовлетворяет этим условиям, то он считается своим. В этом случае его файл-вложение копируется в =IN_BOX= пользователя под именем <USR><NNNNN>.<ADM>, а из <=SPRINT=>\IN.BOX конверт и вложение удаляются. Если нет, то конверт считается чужим, в этом случае он и его возможное вложение остаются без изменений или удаляются в зависимости от буквы в параметре команды _Sprint (см. табл. 26.4 значений параметра).

Отметим, что по аналогии с другими почтами, в Sprint Mail есть возможность выполнить только подготовительные операции без вызова BAT-файла — см. табл. 26.4, значения o, O, p, P, t, T.

Теперь рассмотрим упоминавшуюся ранее процедуру контроля отправки по Sprint Mail, задаваемую командой _Spr_cont(param), где param — параметр — принимает следующие значения, приведенные ниже в табл. 26.5.

Таблица 26.5. *Значения параметров при использовании команды _Spr_cont*

	Значение параметра	Удаление отправленных конвертов из SENT.BOX и приложений из OMEGA.BOX	Удаление неотправленных конвертов из OUT.BOX, перенос приложений из OMEGA.BOX в OUT.BOX системы OMEGA
1	B или b	x	x
2	Любое другое	x	

Работа процедуры:

- осуществляется просмотр <=SPRINT=>\SENT.BOX;
- найденные в нем свои конверты удаляются из <=SPRINT=>\SENT.BOX, а их приложения — из <=SPRINT=>\OMEGA.BOX.

Если указать параметр b или B (back) при запуске процедуры контроля, то можно попытаться использовать другую почту, вернув приложение на прежнее место, а именно:

- оставшиеся в <=SPRINT=>\OMEGA.BOX приложения переносятся обратно в =OUT.BOX= системы OMEGA;
- оставшиеся в <=SPRINT=>\OUT.BOX свои конверты удаляются.

Особенности использования альтернативной почты

Необходимые условия использования пользователем альтернативной почты.

- Должны быть созданы и помещены в SoftDir следующие BAT-файлы, осуществляющие запуск альтернативной почты:
  - **MAIL_S.BAT**, содержащий командную строку запуска альтернативной почты в режиме только отправки;
  - **MAIL_R.BAT**, содержащий командную строку запуска альтернативной почты в режиме только приема;
  - **MAIL_A.BAT**, содержащий командную строку запуска альтернативной почты в режиме отправки и приема.
- В сценарии работы должна быть задана команда `_Mail(param)`, где `param` — параметр — латинская буква, определяющая операции, выполняемые почтовой программой. Перечень всевозможных параметров задается в приводимой ниже табл. 26.6.

Таблица 26.6. *Значения параметров при использовании команды* `_Mail`

	Значение параметра	Отправка	Прием	Удаление чужой входящей почты
1	s	x		
2	S	x		
3	r		x	
4	R		x	x
5	a	x	x	
6	A	x	x	x

Отправка по альтернативной почте.

- Если осуществляется отправка объединенных файлов от удаленного пользователя своему администратору, то к четырехзначному имени объединенного файла добавляется `<SS><HH>.<MM>`, где `MMSSHH` — текущее время: минуты, секунды, сотые доли секунды.
- Для каждого найденного в `=OUT_BOX=` объединенного файла поиск его адресата в MAIL.BAS. Если адресат `<TARGET>` найден, то объединенный файл сразу же переносится в каталог, указанный в строке `<TARGET>_OUT:` файла MAIL.BAS и из `=OUT_BOX=` пользователя удаляется.

- Запуск BAT-файла MAIL_S.BAT (при параметре `s` или `S`) или MAIL_A.BAT (при параметре `a` или `A`).

Здесь, как и везде ранее, при параметре `A` или `a` отправка осуществляется первой.

Прием по альтернативной почте.

- При параметре `R` или `r` запуск BAT-файла MAIL_R.BAT.
- По файлу MAIL.BAS проверка всех каталогов, указанных в строках `<SOURCE>_IN:`, где `<SOURCE>` — всевозможные идентификаторы отправителей.
- В каждом каталоге `<SOURCE>_IN:` поиск входящих файлов и проверка следующих условий:
  - длина имени файла с расширением должна быть равна 11 (для файлов, полученных удаленным пользователем из администрации) или 12 (для файлов, полученных напрямую от других пользователей);
  - если имя файла длины 11, то в него должны входить только цифры, если 12, то оно должно иметь структуру `<USR><NNNNN>.<ADM>`;
  - все файлы, удовлетворяющие этим условиям, считаются своими и переносятся в `=IN_BOX=` пользователя. Прочие файлы считаются чужими и остаются без изменений или удаляются в зависимости от значений параметра.

Заканчивая раздел, посвященный использованию почты, отметим, что помимо описанных в нем способов, в системе OMEGA можно использовать и различные другие, удобные для пользователя возможности. Например, подготовка Internet-пакетов на отправку (команда `_Internet(O)`), но отправка их с помощью Microsoft Outlook, прием по Microsoft Outlook Internet-пакетов с копированием их в `=INTERNET_IN=`, затем запуск `_Internet(P)` для их раскрытия. Подобные способы требуют выполнения ручных операций (запуск Microsoft Outlook, копирование Internet-пакетов), но в некоторых случаях могут быть полезны для пользователей, привыкших работать с той или иной почтовой системой и ведущих переписку также помимо OMEGA. В этом случае роль OMEGA сводится к выполнению всего комплекса подготовительных операций: группированию сообщений, их подписи и шифрованию, архивированию и адресации. Пользователю останется только вложить готовый файл в конверт и отправить его по назначению.

Глава 27

Контроль за доставкой электронных документов

О контроле за доставкой электронных документов мы уже упоминали в *главах 23—25*. Давайте суммируем все, что было сказано о применяющихся в системе OMEGA способах контроля за доставкой электронных документов, и рассмотрим некоторые их особенности подробнее.

- Контроль за доставкой электронных документов имеется как у пользователя, так и у администрации электронного документооборота.
- Контроль за доставкой осуществляется с помощью индивидуальных архивов, автоматически ведущихся как у пользователя, так и у администрации.
- Каждый электронный документ имеет индивидуальные параметры контроля за его доставкой: время, в течение которого отправитель должен получить подтверждение его приема и максимальное число повторных отправок.
- Параметры контроля за доставкой электронного документа с помощью специального файла-директивы передаются от пользователя к администратору.
- Если отправитель в назначенное время не получил подтверждение приема электронного документа, то он с помощью автоматической системы контроля направляется повторно. При исчерпании максимального числа повторных отправок пара "документ-протокол" с контроля снимается и в базе данных подписей, контролирующей прохождение документа, переносится в специальный раздел для документов, на которые не было получено подтверждения приема.
- Автоматическая система контроля запускается с помощью процедуры `_Control`, которая должна быть прописана в сценарии работы системы OMEGA. При отсутствии этой строки система автоматического контроля отключена.

Система контроля предназначена, в первую очередь, для важных денежных документов. Непрохождение или задержка платежа чреваты для банка различными неприятностями, вплоть до выставления клиентом штрафных санкций. А если в день проходят тысячи платежей и система электронного документооборота полностью автоматизирована, то и система контроля за прохождением платежей тоже должна быть автоматизирована. В системе OMEGA получилась универсальная система контроля за доставкой по назначению совершенно произвольных файлов, вне зависимости от их содержания.

Основу системы контроля составляет открытая часть SDB — базы данных подписей. О ней мы уже упоминали в *главе 23*. Напомним, что при построении SDB мы специально физически разделили записи, относящиеся к открытой и закрытой частям SDB. Система контроля фактически сводится к просмотру всех записей, находящихся в открытой части SDB, и нахождению среди них тех, для которых время ожидания подтверждения получения уже истекло. Для них восстанавливается пара "документ-протокол" и затем она автоматически направляется на повторную отправку. Никаких новых подписей не осуществляется.

Из таблицы 23.3, содержащей формат записи в SDB, видно, что специальных полей для повторных отправок в SDB не предусмотрено. Дата и время повторных отправок записываются в поле, предназначенное для подписи получателя. После получения подтверждения приема все эти данные затираются подписью получателя. Одновременно с этим запись переносится из открытой в закрытую часть SDB и тем самым снимается с контроля.

Для того чтобы система контроля работала быстро и эффективно, необходимо, чтобы объем открытой части SDB оставался ограниченным. Если какой-либо получатель по каким-то причинам не принимает направляемые ему документы, то это приводит к росту объема открытой части SDB у отправителя и тем самым общая скорость работы системы документооборота снижается. Оператор, обслуживающий рабочее место пользователя системы OMEGA, в интерактивном режиме всегда имеет возможность провести мониторинг своего архива и выявить подобные ситуации.

Отметим еще одну особенность системы контроля, работающей у администратора. Каждый документ, принятый от пользователя, автоматически ставится на контроль в администрации. Одновременно на контроль ставится и документ типа пакет, включающий в себя этот документ пользователя. По истечении срока контроля документа пользователя он будет повторно включен уже в новый документ типа пакет, который также ставится на контроль. Таким образом, при неудачном выборе параметров контроля у пользователя и администратора возможна значительная перегрузка канала связи за счет многократной повторной отправки и документов пользователя и документов типа пакет администратора, а также поступление подтверждения приема на документ, снятый с контроля.

Из приведенных рассуждений можно сделать вывод о том, что пользователю, пересылающему документы через администрацию, не следует устанавливать излишне жесткие параметры контроля, превышающие возможности канала связи, используемого администратором: эффект в этом случае может получиться противоположный ожидаемому.

Отметим, что как у пользователя, так и у администратора процедуры контроля за доставкой электронных документов не требуют наличия в оперативной памяти секретных ключей. Все подписи повторно отправляемых документов восстанавливаются по SDB. Следовательно, процедура контроля может быть выделена в основном меню (сценарии работы) отдельной кнопкой. Запускать процедуру контроля сможет в этом случае лицо, вообще не имеющее доступа к секретным ключам системы OMEGA. Однако направляемые повторно электронные документы восстанавливаются в промежуточном буфере, для их отправки адресату необходимо выполнить еще одну из процедур, переносящих данные из промежуточного буфера в пакеты. У пользователя это может быть отправка в автоматическом или интерактивном режиме. Отправка в автоматическом режиме у пользователя, если нет новых документов, также не требует секретных ключей. Однако у администрации все отправляемые пакеты обязательно шифруются, поэтому отправить восстановленные документы без секретного ключа нельзя. Положить в промежуточный буфер — можно, а отправить — нельзя.

Отметим теперь особенность системы контроля у администрации. Как уже отмечалось выше, там фактически существует две независимые системы контроля: за документами пользователей и за внешними пакетами, созданными самим администратором. Сами внешние пакеты в архиве администрации не хранятся, их содержание восстанавливается по базе данных реестров (RDB) и базе данных документов пользователей (DDB). И здесь появляется одна важная особенность системы контроля за доставкой внешних пакетов: пакет может быть восстановлен и отправлен повторно только в том случае, когда все входящие в него документы являются открытыми. Если хотя бы один из документов, входящих в пакет, является закрытым, то соответствующая запись в SDB перенесена в закрытую часть SDB, а ее объем может быть весьма велик. Восстановление пакета в этом случае может быть связано со значительными временными затратами.

Рассмотрим теперь дополнение автоматической системы контроля за доставкой электронных документов интерактивными средствами контроля за состоянием архива пользователя или администратора.

В интерактивном модуле для просмотра архива системы OMEGA имеется возможность устанавливать различные фильтры, в том числе и фильтр "Открытые документы" (рис. 27.1).

При задании такого фильтра на экране отображаются только открытые документы. Как видно из рис. 27.1, можно установить несколько типов фильтров

для открытых документов: просто "Открытые", на которые нет подтверждения их получения конечным адресатом, "Открытые АЭД", на которые нет подтверждения их получения своей администрацией, и "Открытые, получатель", на которые есть подтверждение их получения своей администрацией, но нет подтверждения от конечного адресата. При фильтре "Закрытые" на экране отображаются только закрытые документы, а при фильтре "Ошибки" — те, максимальное число повторных отправок которых использовано, а подтверждение приема не получено. Сразу же отметим, что в раздел "Ошибки" пользователь имеет возможность перенести любой открытый документ, если, например, выяснилось, что документ подготовлен ошибочно и нужно остановить его дальнейшее прохождение.

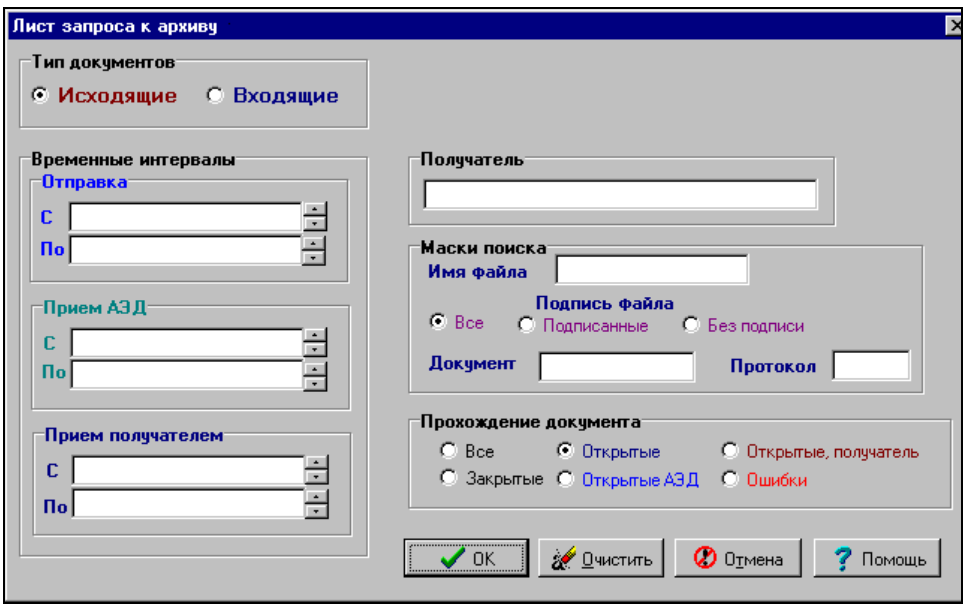

Рис. 27.1. Фильтры для работы с архивом системы OMEGA

На рис. 27.1 также видно, что у пользователя имеются и другие возможности фильтрации документов архива. В фильтре могут присутствовать различные временные интервалы (время отправки, приема АЭД, приема получателем), признаки получателя (символы его ID), признаки файлов, входящих в документы. На рис. 27.2 приведен пример фильтра для отслеживания платежных поручений, отправленных 17 июля 2002 года в банк. Имена файлов с платежными поручениями начинаются с PAY, через две позиции указан месяц (07) и имеют расширение DOC.

На рис. 27.3 приведен результат запроса к архиву по этому фильтру.

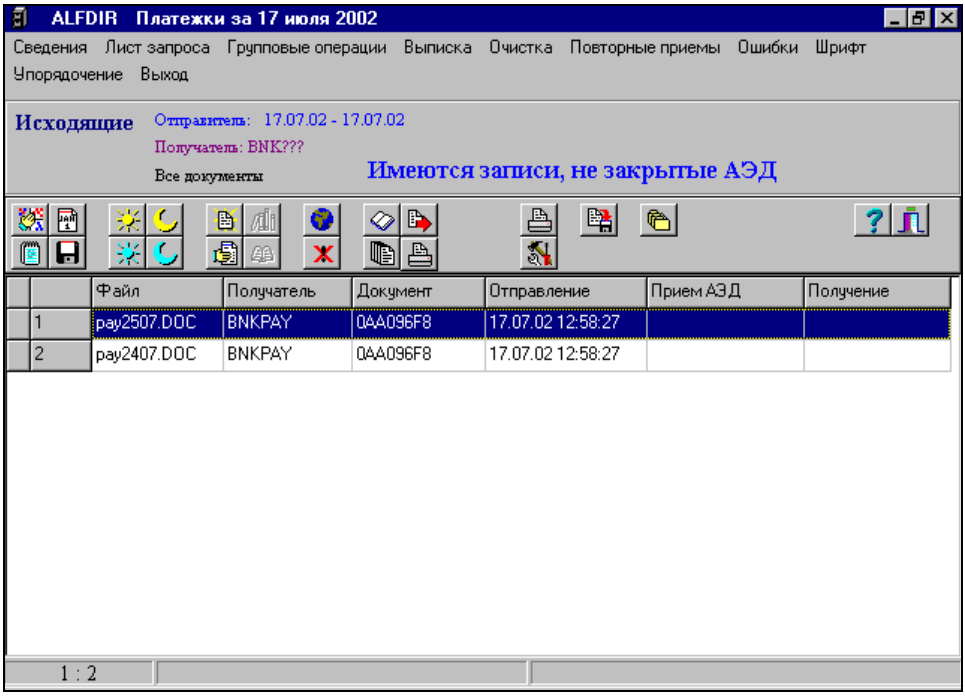

Рис. 27.2 Пример фильтра для платежных поручений, отправленных 17 июля 2002 г.

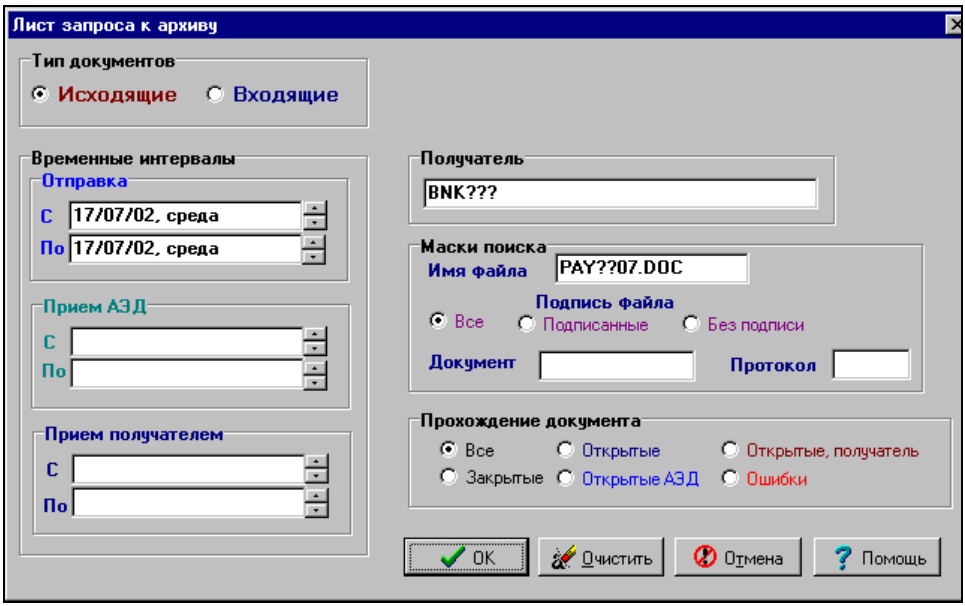

Рис. 27.3. Результат запроса к архиву

Как видно на рис. 27.3, результатом запроса к архиву является перечень файлов и соответствующих документов, удовлетворяющих всем условиям фильтра. Щелкнув два раза мышью по интересующему пользователя файлу, он может получить более подробную информацию о нем, а также получить возможность его восстановления (рис 27.4).

Кнопка **Полная информация** позволяет также получить информацию о времени ожидания подтверждения получения и числе повторных отправок (рис. 27.5).

Как видно из рис. 27.1—27.5, у пользователя при работе с архивом в интерактивном режиме имеется достаточно широкий набор возможностей по контролю за прохождением электронных документов. В частности, в любой момент он имеет возможность осуществить повторную отправку документа, пометить его как ошибочный (если был открытый) или сделать открытым, если был помечен как ошибочный, восстановить его содержание или целиком, или только отдельный файл, посмотреть файл или письмо к документу, распечатать файл на принтере и т. п. Существующая система фильтрации позволяет сравнительно легко осуществлять поиск требуемых документов и файлов, сохранять заданные фильтры, загружать типовые фильтры (Исходящие/входящие за текущий день/месяц), а также делать выписки из архива. Выписка сохраняет всю информацию о выбранных файлах и документах, позволяет их восстанавливать, однако она фиксирует текущее состояние документа, и в дальнейшем получение различных подтверждений получения не влияет на выписку. Пользователи могут обмениваться друг с другом выписками из своих архивов, выписку одного пользователя можно загружать в архив другого. Выписка представляет из себя отдельный файл, содержащий всю требуемую информацию о выбранных файлах и документах архива, ее можно в свою очередь пересылать по системе электронного документооборота как обычный файл. В случае возникновения недоразумений между участниками системы электронного документооборота они могут разрешить их путем направления друг другу соответствующих выписок из своих архивов. На рис. 27.6 приведен пример выписки пользователя BNKPAY, загруженной у пользователя ALFDIR.

Завершая главу, посвященную контролю за прохождением электронных документов, кратко суммируем основные особенности используемых в системе OMEGA способов контроля.

- Имеется автоматическая система контроля, которая обеспечивает повторные отправки электронных документов по истечении контрольного срока получения подтверждения доставки. Контрольный срок доставки может быть индивидуальным для различных типов электронных документов.

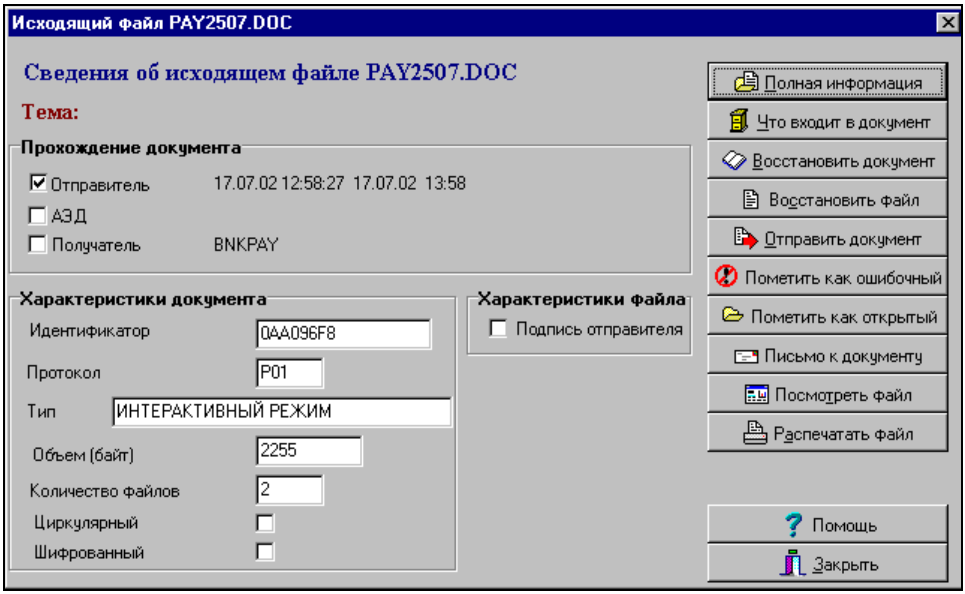

Рис. 27.4. Информация о файле из архива

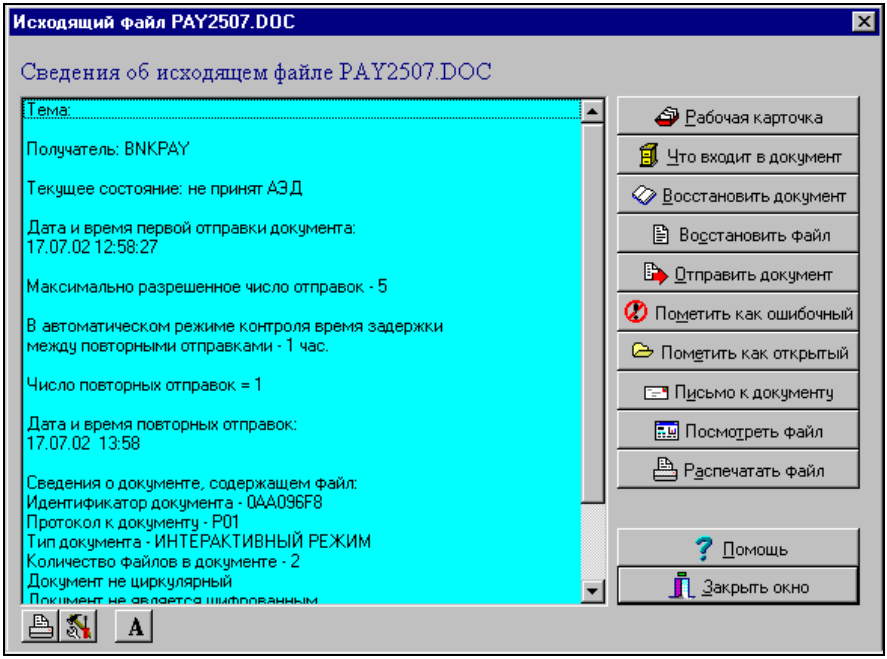

Рис. 27.5. Полная информация о файле из архива

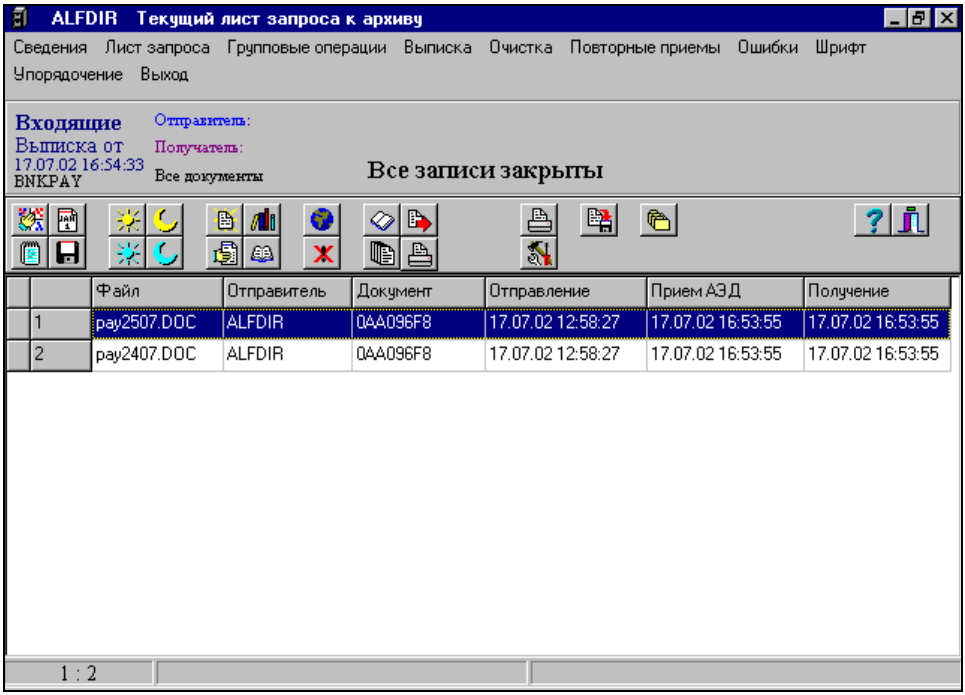

Рис. 27.6. Выписка пользователя BNKPAY, загруженная у ALFDIR

- С помощью интерактивной системы контроля за состоянием архива оператор имеет возможность осуществлять визуальный контроль за прохождением различных электронных документов. Оператор также имеет возможность активно вмешиваться в процесс их прохождения, осуществлять в любой момент времени повторные отправки, отсеивать ошибочные документы, делать и отсылать различные выписки из архива.
- Для облегчения поиска требуемых электронных документов у оператора имеется возможность установить различные фильтры поиска. Эти фильтры могут быть сохранены для последующих сеансов работы с архивом.

Эти системы контроля позволяют обеспечивать гарантированную доставку по назначению достаточно большого количества типовых электронных документов. Объем ежедневно обрабатываемых с помощью системы OMEGA файлов может доходить до нескольких десятков тысяч.

Глава 28

Главный менеджер системы

Главный менеджер — это основное лицо в системе электронного документооборота OMEGA. Изготовление и подпись конфигурационных файлов, определяющих права и возможности участников системы электронного документооборота, рассылка их по назначению, прием отчетов об их получении и установке на рабочих местах, ввод новых пользователей и подготовка для них инсталляционных пакетов — вот перечень основных вопросов, входящих в компетенцию Главного менеджера системы OMEGA. Для успешного функционирования системы OMEGA в крупной организации Главному менеджеру необходимо иметь специализированное программное обеспечение, обеспечивающее оперативное управление этой системой. В нем предусматривается осуществление подписи Главного менеджера на подготовленных конфигурационных файлах, поэтому необходимо заранее выработать секретный ключ Главного менеджера, с помощью которого осуществляются такие подписи.

Давайте посмотрим на те проблемы, с которыми придется иметь дело человеку, назначенному на должность Главного менеджера системы OMEGA. Эти проблемы разобьем на две части: одноразовые, возникающие при наладке и запуске в эксплуатацию системы, и повседневные, возникающие в процессе ее эксплуатации.

Очевидно, что для АРМ Главного менеджера необходимо иметь несколько баз данных. В отличие от специализированных баз данных пользователя и администрации, базы данных Главного менеджера — это обычные, традиционные базы данных, построенные с использованием, например, механизма BDE — Borland Database Engine. Имеющиеся в языке программирования Borland C++ Builder возможности позволяют сравнительно легко сделать программные средства для работы с такими базами данных. Заполнение баз данных Главного менеджера осуществляется один раз на этапе подготовки к эксплуатации системы OMEGA. Давайте рассмотрим эту процедуру подробнее.

В базах данных Главного менеджера будет храниться необходимая информация о пользователях и администрациях, участвующих в системе документооборота: их идентификаторы, название, а также индивидуальные параметры, входящие в их конфигурационные файлы. И здесь необходимо сделать одно замечание. Конфигурационные файлы у участников индивидуальны, но некоторые параметры в них могут быть одинаковыми для многих. Например, каталог `=SOFT=` для специализированных программных модулей, каталоги `=IN_BOX=` и `=OUT_BOX=` для входящей и исходящей почты пользователей и т. п. Число участников с одинаковыми параметрами может быть весьма большим, поэтому многократное повторение одних и тех же каталогов в базах данных Главного менеджера значительно увеличит их размер. Поэтому у Главного менеджера имеется *база данных шаблонов*, содержащая типовые параметры конфигурационных файлов, с помощью которой удается значительно сократить общий объем баз данных и уменьшить трудоемкость заполнения баз данных пользователей и администраций. Оператор, работающий на АРМ Главного менеджера, имеет возможность заранее заготовить набор типовых шаблонов и значительно облегчить свою дальнейшую работу по составлению конфигурационных файлов (рис. 28.1).

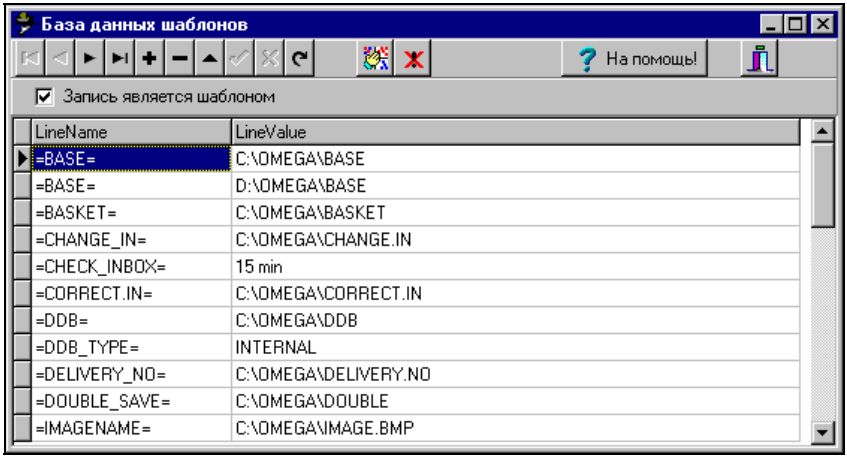

Рис. 28.1. База данных шаблонов

Человек, ответственный за наладку и запуск системы OMEGA, начинает свою работу с того, что составляет общий перечень всех пользователей и администраций системы, определяет их идентификаторы и названия, после чего заносит их в специализированные базы данных. Эта процедура осуществляется одновременно с заданием основного конфигурационного файла SETUP. Специально для этой цели в программе для АРМ Главного менеджера имеется отдельный интерфейс (рис. 28.2).

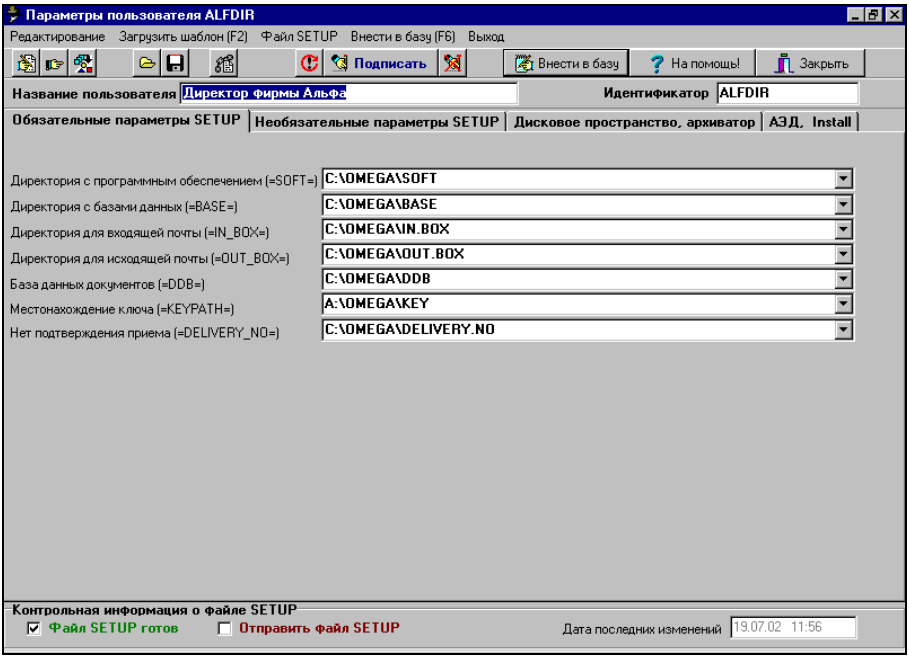

Рис. 28.2. Подготовка основного конфигурационного файла SETUP пользователя

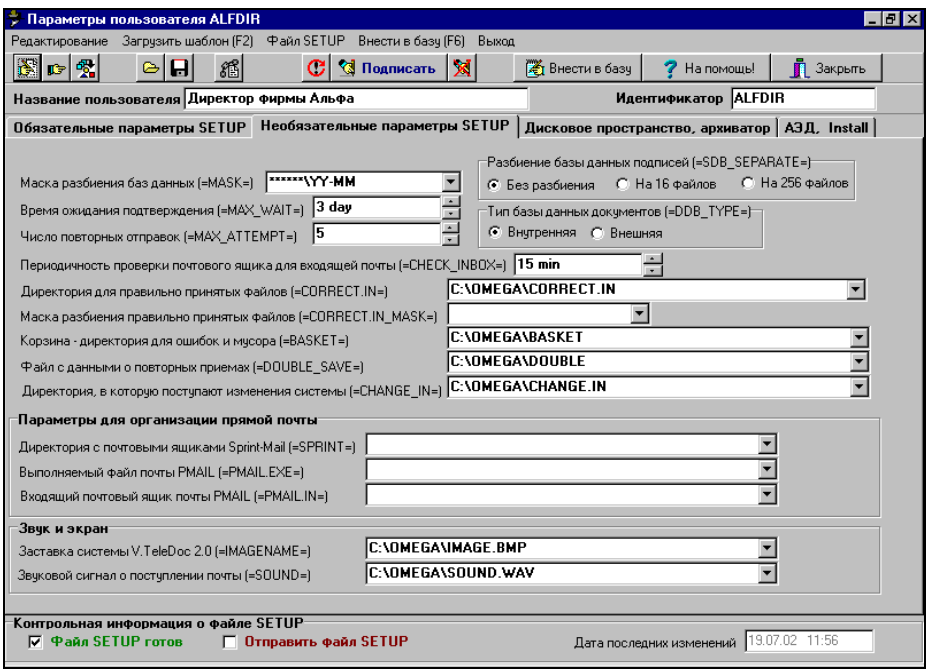

Рис. 28.3. Необязательные параметры файла SETUP

Из рис. 28.2 видно, что параметры файла SETUP пользователя разбиты на несколько групп. В первую группу входят обязательные параметры, без которых создание файла невозможно. Ко второй группе относятся необязательные параметры, которые создают дополнительные возможности при работе пользователя (рис. 28.3).

К третьей группе относятся параметры контроля за допустимым объемом дискового пространства, а также тип используемого архиватора (рис. 28.4).

И, наконец, к последней группе параметров относятся те, которые определяют взаимодействие пользователя и администрации, а также каталог для самого файла SETUP, который используется при построении инсталляционного пакета (рис. 28.5).

В последнем случае задаются параметры, не входящие в файл SETUP пользователя. База данных документов, маска разбиения и почтовые ящики присутствуют в файле MEMBER.AED администрации.

При заполнении некоторых позиций файла SETUP оператор имеет возможность использовать заранее заготовленные шаблоны. Каждая внесенная запись сама, в свою очередь, вносится в базу данных шаблонов и может быть затем использована при построении файла SETUP для других пользователей или администраций.

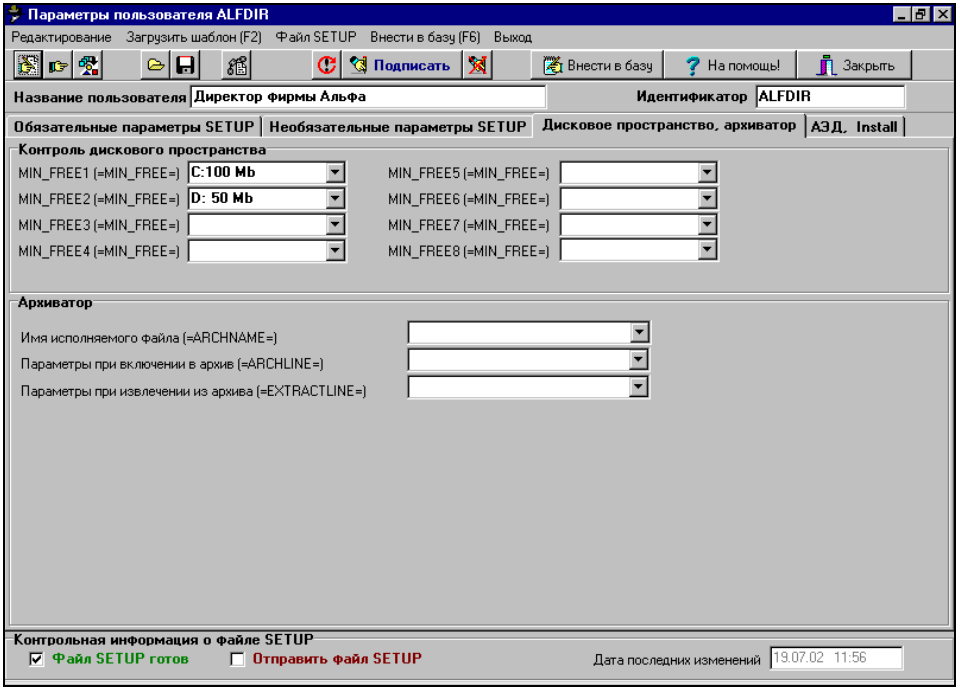

Рис. 28.4. Параметры контроля дискового пространства и архиватор в файле SETUP

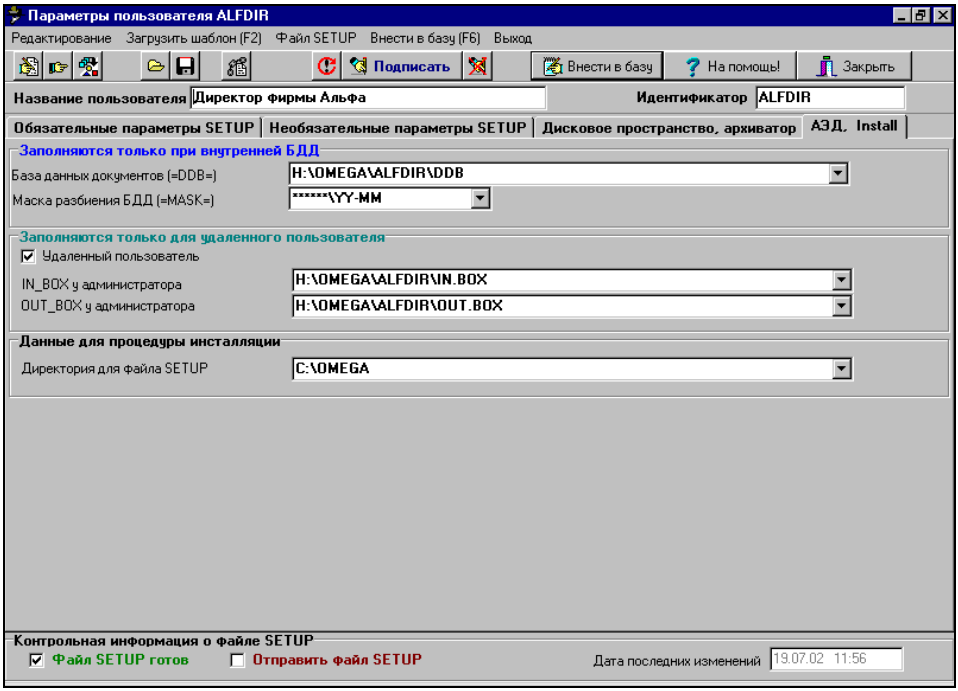

Рис. 28.5. Параметры, определяющие взаимодействие пользователя и администрации, а также положение самого файла SETUP

После заполнения всех необходимых параметров файла SETUP, оператор осуществляет его подпись. Подпись файла SETUP сохраняется в базе данных вместе с другими параметрами.

Пользователь или администрация считаются внесенными в базу данных Главного менеджера, если задан их основной конфигурационный файл SETUP. Перечень всех внесенных в базу данных пользователей или администраций оператор имеет возможность увидеть на экране (рис. 28.6).

После составления базы данных пользователей и администраций, оператор осуществляет для каждого пользователя и администрации определение их связей и способов связи. При этом для каждого пользователя определяются сразу несколько конфигурационных файлов: RK.BAS, MAIL.BAS, VMAIL.BAS, SPRINT.BAS (для администрации — TRANSIT.BAS вместо RK.BAS) — рис. 28.7.

При определении связей пользователя и администраций Главный менеджер должен заранее знать всевозможные адреса электронной почты участников системы электронного документооборота.

Имеется возможность часть связей пользователя сделать прямыми, минуя АЭД. Как видно из рис. 28.7, связь ALFDIR с BETDIR осуществляется напрямую через Internet, а с GAMDIR — по альтернативной почте.

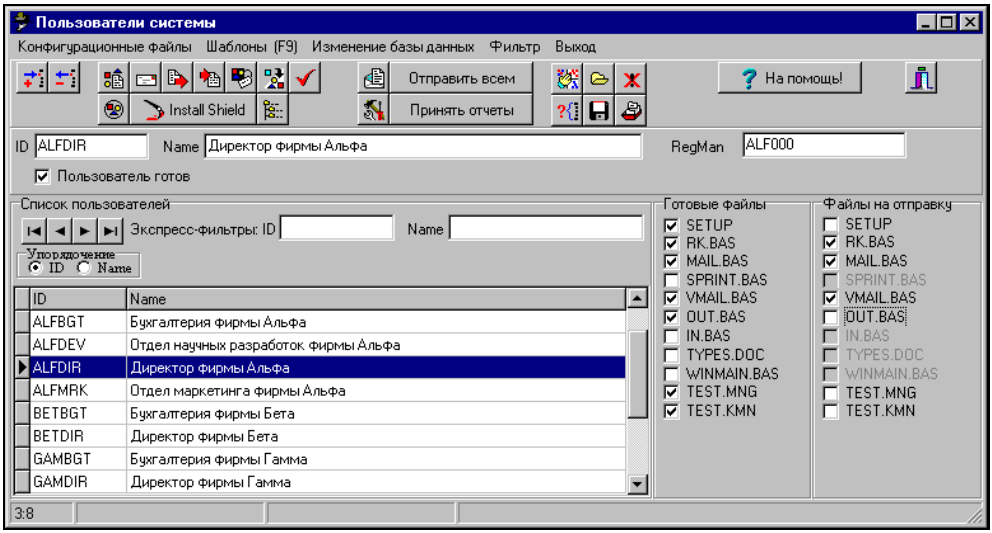

Рис. 28.6. Перечень внесенных в базу данных пользователей

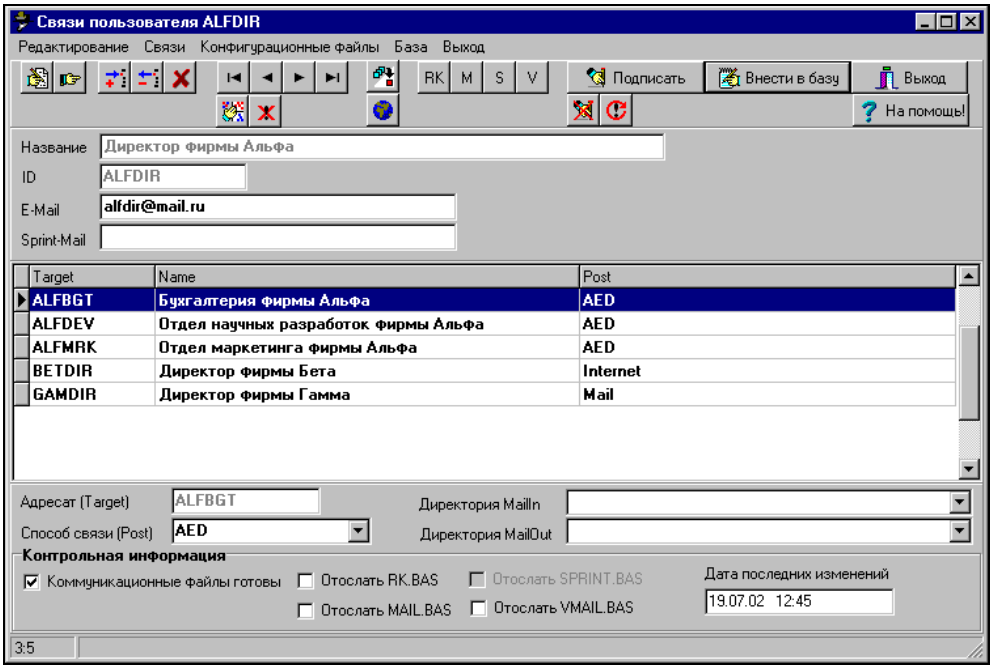

Рис. 28.7. Связи пользователя

Заполнение базы данных пользователей и администраций и определение их связей — это основная часть подготовительной работы, осуществляемой

Главным менеджером при подготовке к запуску системы OMEGA. Дальнейшие действия осуществить уже проще. Для пользователя — это определение индивидуальных конфигурационных файлов OUT.BAS, IN.BAS, TYPES.DOC, WINMAIN.BAS, TEST.MNG и TEST.KMN. Перечень этих файлов виден на рис. 28.6. При их подготовке используются интерфейсы, похожие на те, которые использовались при подготовке файла SETUP. На рис. 28.8 приведен пример интерфейса для подготовки файла OUT.BAS.

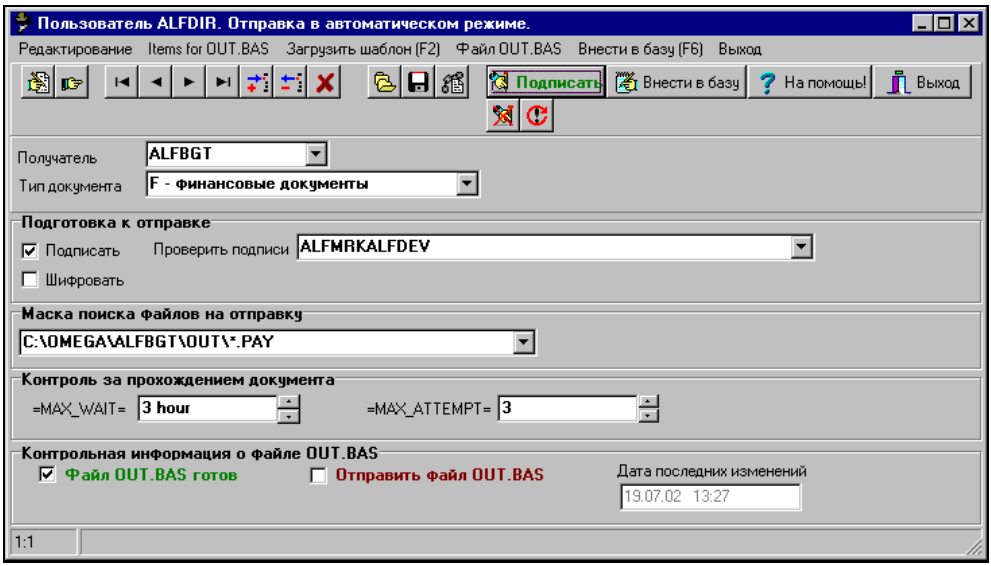

Рис. 28.8. Интерфейс для подготовки файла OUT.BAS

С точки зрения обеспечения безопасности системы OMEGA, наибольший интерес представляют файлы TEST.MNG и TEST.KMN, задающие перечень файлов, в которых будет проверяться электронная подпись Главного менеджера и менеджера системы безопасности соответственно. Давайте остановимся на них подробнее.

При запуске основного модуля системы OMEGA как у пользователя, так и у администрации автоматически проверяется подлинность электронных подписей Главного менеджера на основном конфигурационном файле SETUP и на файлах TEST.MNG и TEST.KMN. После этого проверяются подлинность подписи Главного менеджера на всех файлах, перечисленных в TEST.MNG, и подлинность подписи менеджера СБ на всех файлах, перечисленных в TEST.KMN. Для файла PUBLKEYS, содержащего открытые ключи, проверяется подпись каждого открытого ключа в отдельности.

Задавая определенные файлы в TEST.MNG и TEST.KMN, Главный менеджер определяет тем самым для каждого участника системы электронного документооборота "степень его свободы". Те файлы, которые не войдут ни

в один из этих перечней, могут изменяться самим пользователем. Например, если Главный менеджер хочет дать пользователю возможность редактировать сценарий работы системы OMEGA, определяемый файлом WINMAIN.BAS, то он не включает его ни в один из перечней TEST.MNG и TEST.KMN. При этом он имеет возможность подготовить WINMAIN.BAS и отослать его пользователю, но подпись в нем при запуске системы проверяться не будет. Те файлы, которые вошли хотя бы в один из перечней TEST.MNG или TEST.KMN, изменять самостоятельно пользователь не может.

На рис. 28.9 приведен пример интерфейса для задания файлов TEST.MNG и TEST.KMN.

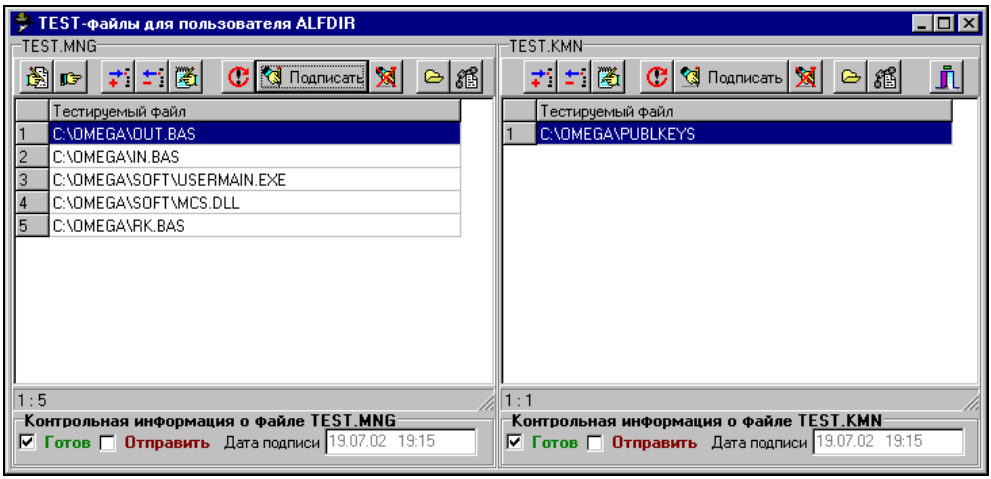

Рис. 28.9. Интерфейс для задания файлов TEST.MNG и TEST.KMN

Итак, индивидуальные конфигурационные файлы готовы. Теперь Главный менеджер имеет возможность подготовить индивидуальный инсталляционный пакет для каждого пользователя и администрации. Но сначала — одно важное замечание.

В инсталляционный пакет должен также входить файл PUBLKEYS, содержащий необходимые открытые ключи. Этот файл готовит менеджер системы безопасности. О нем подробнее мы поговорим в следующей главе, а сейчас же отметим механизм взаимодействия между Главным менеджером и менеджером СБ.

У менеджера СБ, аналогично Главному менеджеру, имеются базы данных, на основе которых он осуществляет, в частности, построение индивидуальных файлов PUBLKEYS, содержащих только требуемые открытые ключи и никаких лишних. Как определить, какие открытые ключи требуются пользователю или администрации? Естественно, что по перечню разрешенных связей. А такой перечень определяется Главным менеджером —

см. рис. 28.7. Кроме того, менеджер СБ — это живой человек, было бы негуманно заставлять его заново проделывать ту огромную работу, которую уже проделал Главный менеджер, создавая базу данных всех пользователей и администраций. Поэтому у Главного менеджера предусмотрено создание *выписки для системы выработки ключей*, с помощью которой затем менеджер СБ формирует свои "ключевые" базы данных. Как взаимная любезность, менеджер СБ снабжает Главного менеджера набором индивидуальных файлов PUBLKEYS, содержащих требуемые открытые ключи.

И еще одно замечание, прежде чем приступить к составлению инсталляционного пакета. Помимо индивидуальных конфигурационных файлов, в инсталляционный пакет входят также общие файлы: требуемые программные модули, файлы для запуска некоторых программных модулей общего пользования, некоторые общие конфигурационные файлы, как, например, файл ASKLIST, определяющий типовой перечень листов запроса к архиву, и т. п. Такие файлы готовятся один раз для всех пользователей и помещаются в определенный каталог АРМ Главного менеджера. Некоторые из них заранее Главный менеджер может подписать.

Для составления инсталляционных пакетов используется механизм широко известного пакета Install Shield Express. На рис. 28.6 можно найти кнопку **Install Shield**, с помощью которой осуществляется построение инсталляционного пакета.

При нажатии на эту кнопку программа запрашивает перечень общих файлов, которые необходимо добавить в инсталляционный пакет (рис. 28.10).

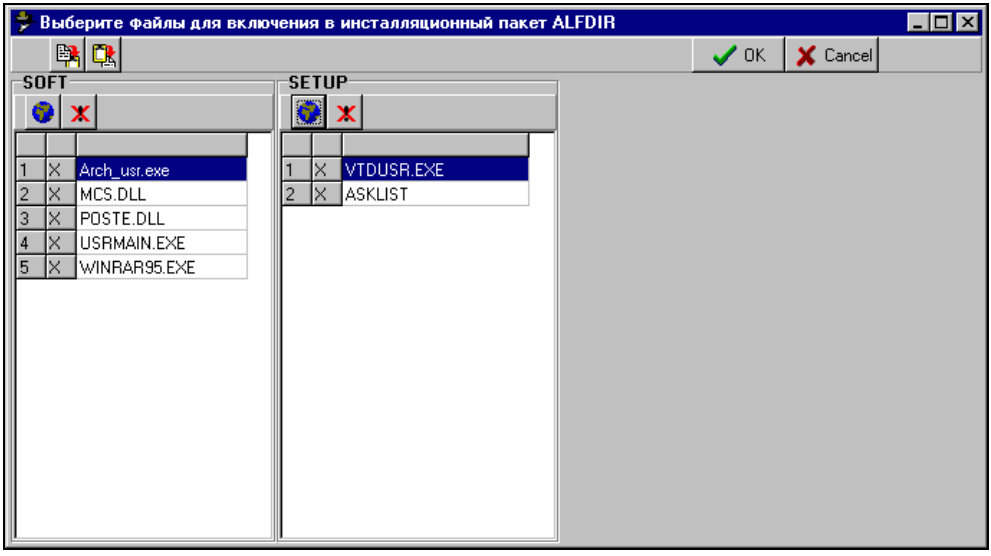

Рис. 28.10. Постоянные файлы, добавляемые в инсталляционный пакет

Механизм составления инсталляционного пакета следующий. Берется некоторый заранее заготовленный типовой проект (файл IWZ), в который программно подставляются имена готовых индивидуальных конфигурационных файлов и указанных постоянных файлов. После этого запускается пакет Install Shield Express.

Составление инсталляционных пакетов — это разовая процедура. Но очень часто в процессе эксплуатации системы OMEGA Главному менеджеру приходится заниматься изменением уже готовых индивидуальных конфигурационных файлов и рассылкой их по назначению. Кто-то хочет добавить себе возможности связей с другими пользователями, изменить тип рассылаемых и получаемых документов, сменить каналы связи — для этого необходимо изменение конфигурационных файлов. Кроме того, исполняемые модули системы OMEGA также не остаются неизменными: разработчик часто вносит в них различные усовершенствования, готовит новые версии, добавляет новые возможности, устраняет выявленные ошибки. Для всего этого необходимо иметь механизм обновления системы OMEGA, т. е. механизм замены программных модулей и индивидуальных конфигурационных файлов в большой и широко разветвленной сети. Для этих целей, как уже упоминалось в *главе 22*, используются региональные менеджеры. Региональный менеджер — это фактически специализированное рабочее место, основными задачами которого является посредничество между Главным менеджером и менеджером СБ и определенной (и сравнительно небольшой) группой пользователей и администраций системы OMEGA. Обслуживать АРМ регионального менеджера может отдельно выделенный для этой цели человек, не принимающий участия в самом документообороте. Он сам не имеет доступа к секретным ключам обслуживаемых им пользователей и администраторов, но имеет свой собственный секретный ключ, с помощью которого подписывает отчеты о проделанных заменах, которые отсылаются в Центр Главному менеджеру и менеджеру СБ.

На рис. 28.6 региональный менеджер задается в поле **RegMan**. По умолчанию, идентификатор регионального менеджера — это идентификатор администрации и символы "000". У регионального менеджера имеются специализированные программные модули и индивидуальные конфигурационные файлы, подготовку которых также осуществляет Главный менеджер. Для этого в его программном обеспечении имеется специальный интерфейс (рис. 28.11).

Вернемся теперь к рассылке изменений конфигурации системы OMEGA. Она осуществляется через региональных менеджеров и может быть индивидуальная и общая. У Главного менеджера ведется база данных отправок, в которую автоматически заносятся все направленные изменения, а также результаты их приема.

Для просмотра базы данных отправок, а также для индивидуальной отправки предусмотрен специальный интерфейс, пример которого приведен на рис. 28.12.

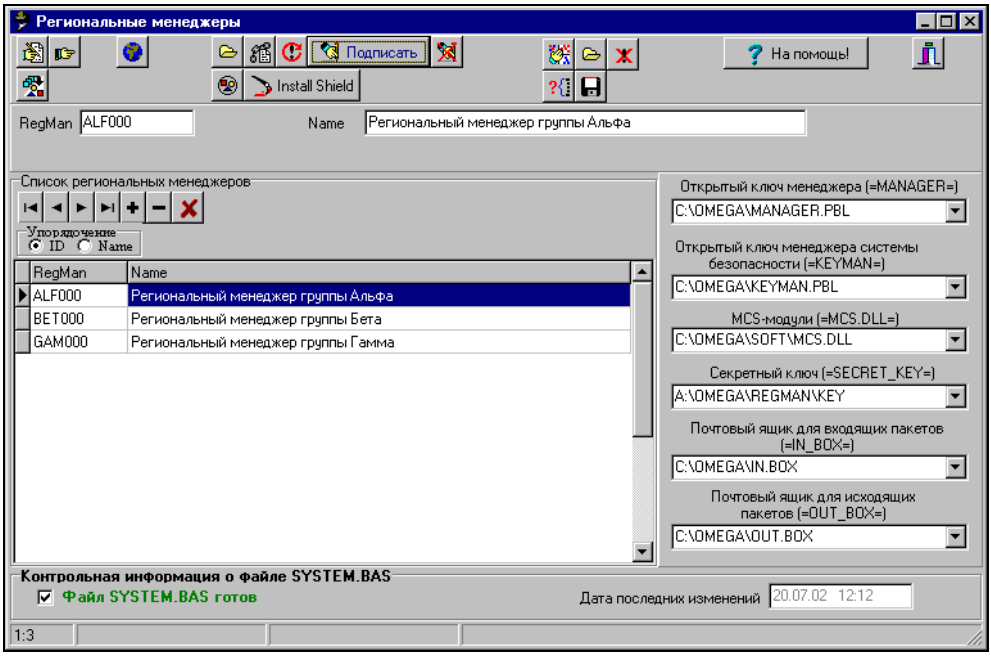

Рис. 28.11. Интерфейс для подготовки конфигурационных файлов региональных менеджеров

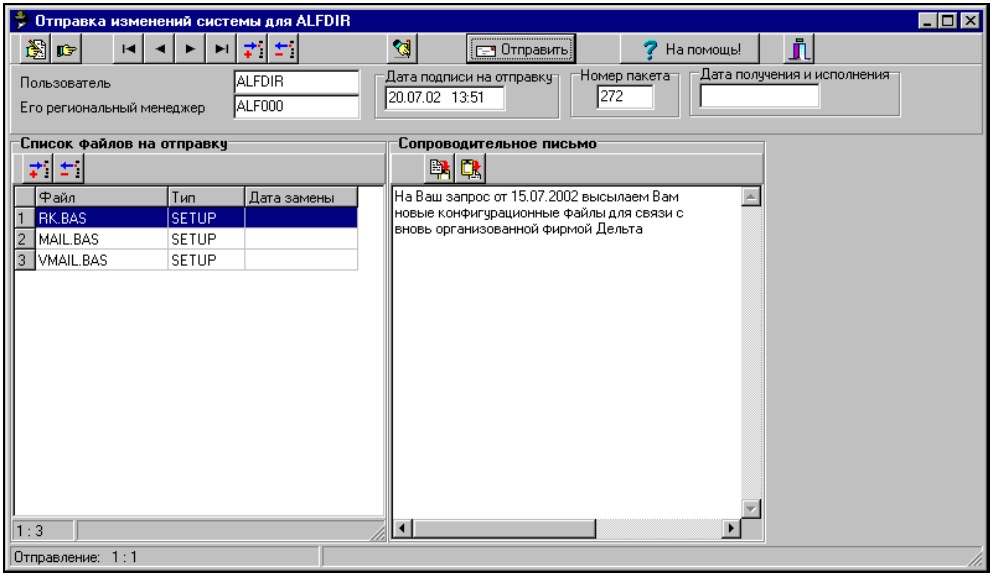

Рис. 28.12. Интерфейс для просмотра базы данных отправок и индивидуальной отправки

Вместе с индивидуальной отправкой предусмотрена также, как это видно на рис. 28.6, кнопка **Отправить всем**, с помощью которой осуществляется автоматическая отправка всем пользователям сразу. Конфигурационные файлы отправляются в пакетах, номер пакета присваивается автоматически (рис. 28.12). Региональный менеджер, получив от Главного менеджера пакет с измененными конфигурационными файлами, осуществляет их установку на рабочих местах пользователей и администраций. После этого Главному менеджеру направляется отчет о проведенных заменах, который принимается с помощью кнопки **Принять отчеты** (см. рис. 28.6). После приема отчетов в базе данных отправок заполняется поле **Дата получения и исполнения** (рис. 28.13).

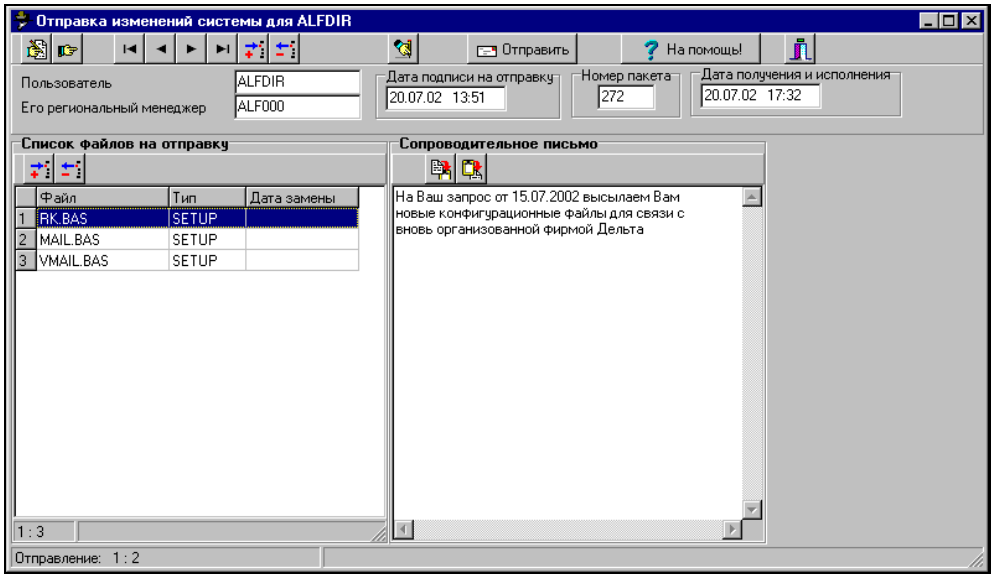

Рис. 28.13. Запись в базе данных отправок с отметкой о получении и исполнении

Завершая главу, посвященную Главному менеджеру системы OMEGA, еще раз отметим, что это главная рабочая сила всей системы. Ни один из других участников системы документооборота не заполняет вручную базы данных, содержащие всех пользователей, администраторов и региональных менеджеров. Главный менеджер затем стойко противостоит всей этой армаде, направляя их деятельность по строго определенному курсу. Высокую зарплату Главному менеджеру!

Глава 29

Менеджер системы безопасности

Слышен звон цепей и грохот кандалов... Со связкой ключей появляется менеджер системы безопасности. Давайте познакомимся с ним.

Обеспечение всех участников системы электронного документооборота секретными и открытыми ключами — вот главная задача менеджера СБ. Но это не обязательно должен быть виртуоз в компьютерных играх: основная часть требуемых секретных ключей вырабатывается из соответствующих базовых секретных ключей. О таком принципе оснащения большой организации секретными ключами мы уже упоминали ранее в *главе 8*. Здесь же имеем воплощение этого принципа на практике.

Итак, прежде чем приступить к составлению собственных баз данных, менеджер СБ должен создать себе несколько базовых секретных ключей, а также свой личный секретный ключ, с помощью которого будет осуществляться сертификация всех открытых ключей. Какие конкретно базовые секретные ключи нужны менеджеру СБ — об этом будет сказано далее.

Следующая задача менеджера СБ — познакомиться с Главным менеджером, установить с ним рабочий контакт, можно даже неформальный. После знакомства попросить у него выписку для системы выработки ключей. Всю основную работу — заполнение баз данных пользователей и администраций — уже проделал Главный менеджер, а менеджер СБ просто использует результаты этой работы в виде выписки.

Теперь можно приступать к заполнению собственных баз данных менеджера СБ. Заполнение — это громко сказано. Просто в основном окне нажимается кнопка **Принять выписку от менеджера** и почти все готово (рис. 29.1).

Вся остальная работа менеджера СБ, в основном, сводится к последовательному нажатию трех кнопок, приведенных на рис. 29.1: **Создать все секретные ключи**, **Создать все открытые ключи**, **Осуществить общую рассылку**. Ну и затем еще кнопки **Принять отчеты**. Все остальное за человека делает программа.

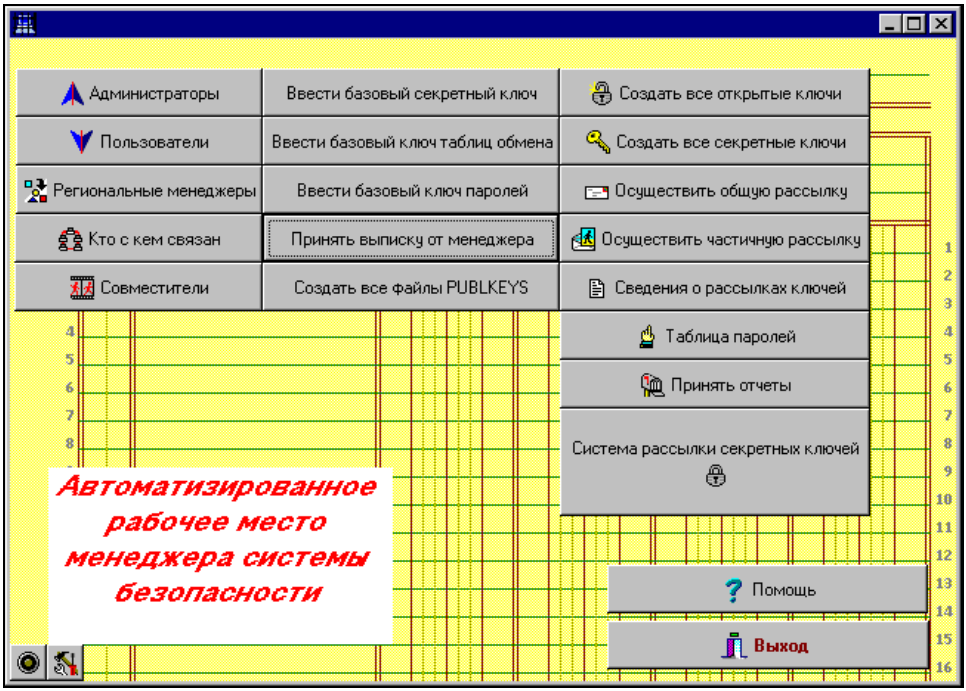

Рис. 29.1. Основное меню АРМ менеджера системы безопасности

Рассмотрим действие и некоторых других кнопок из приведенных на рис. 29.1. Они тоже могут оказаться полезными менеджеру СБ.

У менеджера СБ ведутся свои базы данных пользователей, администраторов и региональных менеджеров. Более того, если у Главного менеджера велась только база данных администраций, то у менеджера СБ ведется база данных администраторов. Как ранее отмечалось в *главе 22*, администрация — это автоматизированное рабочее место с набором специализированных программных модулей и конфигурационных файлов, а администратор — это человек, работающий на АРМ администрации. Причем таких людей может быть один или два, каждый из которых имеет свой индивидуальный секретный ключ. Поэтому менеджер СБ как ответственный за выработку секретных и открытых ключей при заполнении базы данных администраторов должен определить, сколько там будет администраторов: один или два (рис. 29.2).

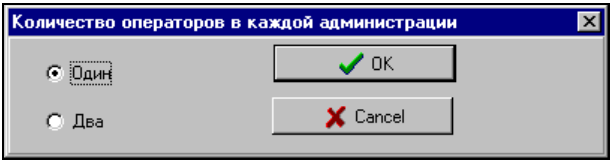

Рис. 29.2. Определение количества администраторов

Менеджер СБ имеет возможность посмотреть свои базы данных пользователей и администраторов (рис. 29.3).

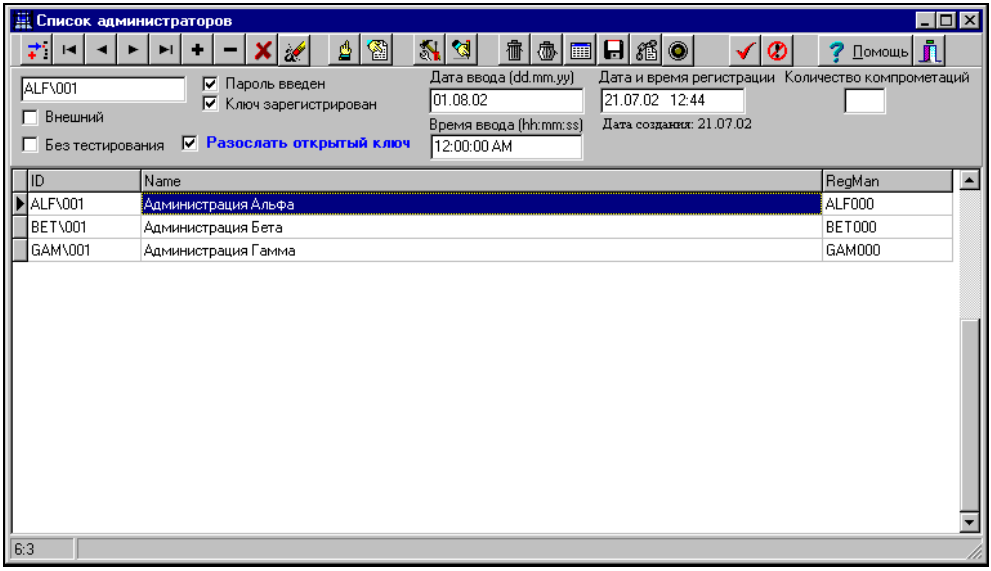

Рис. 29.3. База данных администраторов у менеджера СБ

К этому рисунку необходимо сделать несколько пояснений, показывающих некоторые особенности работы программы менеджера СБ. В *главе 9* настоящей книги мы уже очень много говорили о тех проблемах, которые возникают при использовании открытых ключей. Одна из основных — смена ключей. В открытом ключе должен присутствовать "час X", т. е. момент, с которого он вступает в действие. На рис. 29.3 такой "час X" задан в строках **Дата ввода** и **Время ввода**. Этот "час X" задается менеджером СБ и защищается его электронной подписью. Регистрация означает, что открытый ключ подписан менеджером СБ. Дата и время регистрации — дата и время подписи открытого ключа менеджером СБ.

Наличие отметки **Разослать открытый ключ** означает, что при ближайшей общей рассылке этот открытый ключ будет включен в пакет, предназначенный для соответствующего регионального менеджера. И, наконец, набор кнопок в верхней части окна дает возможность менеджеру СБ выполнять только для данного администратора такие типовые операции, как просмотр и изменение пароля к его секретному ключу, создание дискеты с комплектом секретных ключей, создание файла PUBLKEYS, содержащего открытые ключи, предназначенные для данной администрации, запись секретного ключа в устройство типа Touch Memory и т. п.

Особо отметим, что ни один из секретных ключей не хранится в базах данных менеджера СБ. Они вырабатываются из базовых секретных ключей,

которые менеджер СБ хранит на отдельных дискетах, и индивидуальных идентификаторов пользователей и администраций. Об этом также шла речь ранее в *главе 8* книги. Менеджер вводит в оперативную память какой-то из этих базовых ключей, если возникает потребность в его использовании. Таких базовых ключей здесь три: для выработки индивидуальных секретных ключей, для выработки таблиц обмена и для выработки паролей.

Отметим, что некоторым пользователям или администраторам менеджер СБ может предоставить право вырабатывать свой индивидуальный секретный ключ самостоятельно. В этом случае менеджеру СБ остается только внести соответствующий открытый ключ в свою базу данных и зарегистрировать его так же, как он регистрирует открытые ключи собственного изготовления. Для этой цели у менеджера СБ в базе данных имеется отметка **Внешний** (см. рис. 29.3).

У менеджера СБ имеется возможность оперативно реагировать на факты компрометации секретного ключа. Для этого нужно нажать кнопку выработки ключа, на экране появляется следующее предупреждение (рис. 29.4).

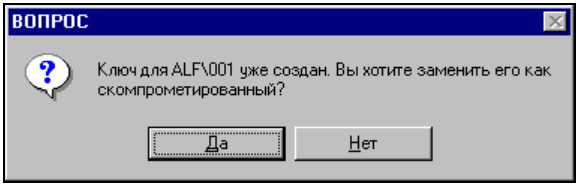

Рис. 29.4. Предупреждение о замене скомпрометированного ключа

После этого автоматически создается новый секретный ключ, соответствующий ему открытый ключ, который также автоматически регистрируется (рис. 29.5).

У менеджера СБ имеется несколько сервисных возможностей, позволяющих ему визуально контролировать состояние своих баз данных. Так, помимо упоминавшегося ранее визуального контроля баз данных администраторов и пользователей, он также имеет возможность увидеть на экране, кто с кем связан. Это очень важный момент, поскольку по связям строится файл PUBLKEYS, содержащий открытые ключи. Связи передаются менеджеру СБ в выписке от Главного менеджера, и в обычной ситуации менеджеру СБ нет необходимости их смотреть. Но при возникновении сбоев в работе системы OMEGA, если какой-нибудь пользователь не получил требуемые ему открытые ключи, менеджер СБ имеет возможность проверить по своей базе данных связи этого пользователя (рис. 29.6).

Как уже отмечалось в прошлой главе, Главному менеджеру для подготовки инсталляционных пакетов необходимо получить от менеджера СБ файлы PUBLKEYS для всех участников системы электронного документооборота. Подготовка всех таких файлов осуществляется с помощью кнопки

PUBLKEYS основного меню менеджера СБ. Подготовленные файлы записываются в виде дерева подкаталогов, соответствующих идентификаторам участников.

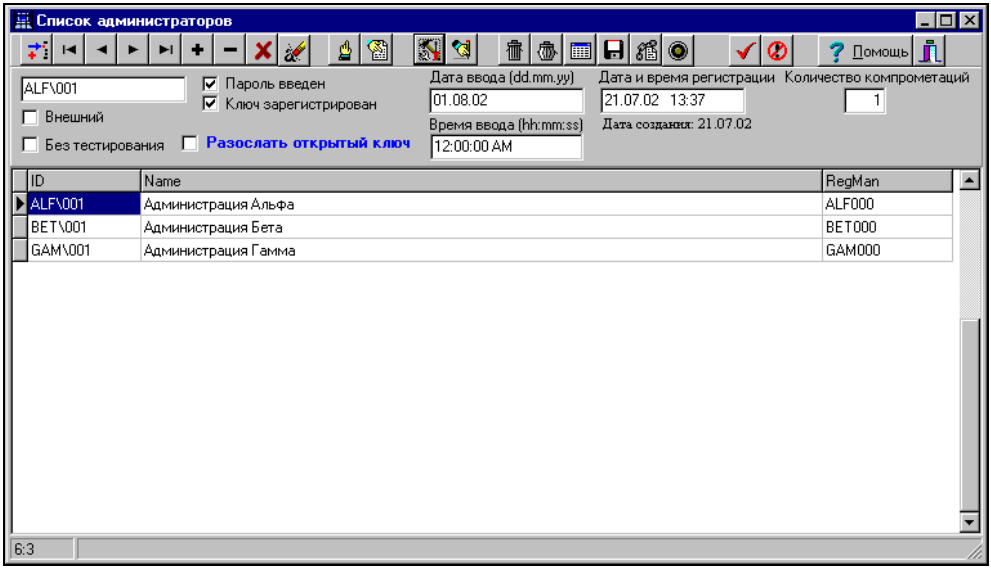

Рис. 29.5. Замена скомпрометированного секретного ключа

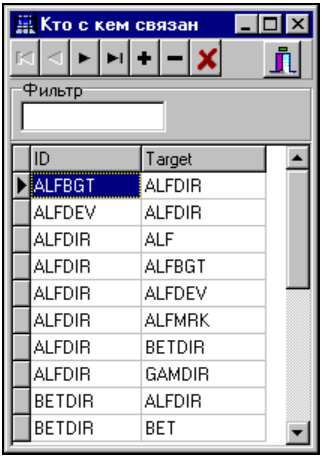

Рис. 29.6. База данных связей, существующая у менеджера СБ

Ну и, наконец, одна из главных задач менеджера СБ — рассылка открытых ключей. Механизм рассылки во многом похож на тот, который использовался у Главного менеджера: рассылка осуществляется в пакетах, предназначенных региональному менеджеру, ведется база данных рассылок, на каждый

пакет менеджер СБ получает подтверждение его приема и обработки, заверенное электронной подписью регионального менеджера.

Рассылка может быть двух типов — выборочная и общая. Выборочная рассылка может осуществляться при введении нового участника системы ЭД или при компрометации ключа у одного из действующих участников. При выборочной рассылке выбирается идентификатор участника, чей открытый ключ необходимо разослать, и адресаты, кому необходимо его направить (рис. 29.7).

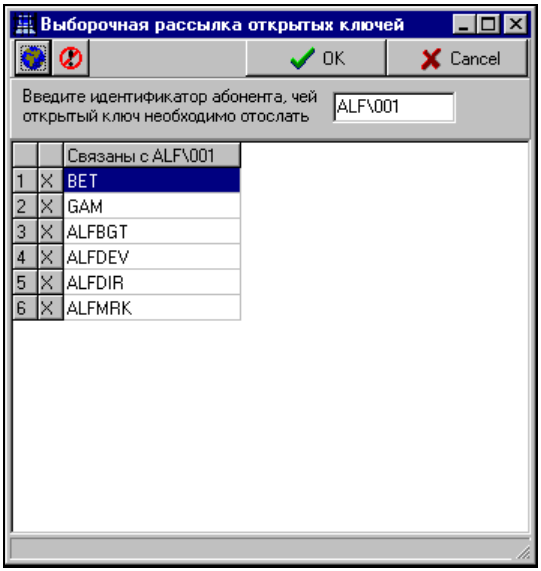

Рис. 29.7. Выборочная рассылка открытых ключей

К отсылаемым пакетам менеджер СБ имеет возможность приложить сопроводительное письмо (рис. 29.8).

На рис. 29.7 и 29.8 видно, что хотя новый открытый ключ нужен 6 участникам системы электронного документооборота, менеджер СБ отправляет его только троим региональным менеджерам, обслуживающим этих участников. Установка полученного открытого ключа нескольким участникам — это уже задача регионального менеджера, о нем мы поговорим в следующей главе. А региональных менеджеров, которым необходимо отправить открытый ключ, программа сама автоматически вычислила по базам данных менеджера СБ.

Региональный менеджер, произведя замену открытых ключей, автоматически посылает менеджеру СБ отчет о проведенных заменах. Этот отчет принимается менеджером СБ с помощью кнопки **Принять отчеты**. Ни рис. 29.9 приведен пример отчета.

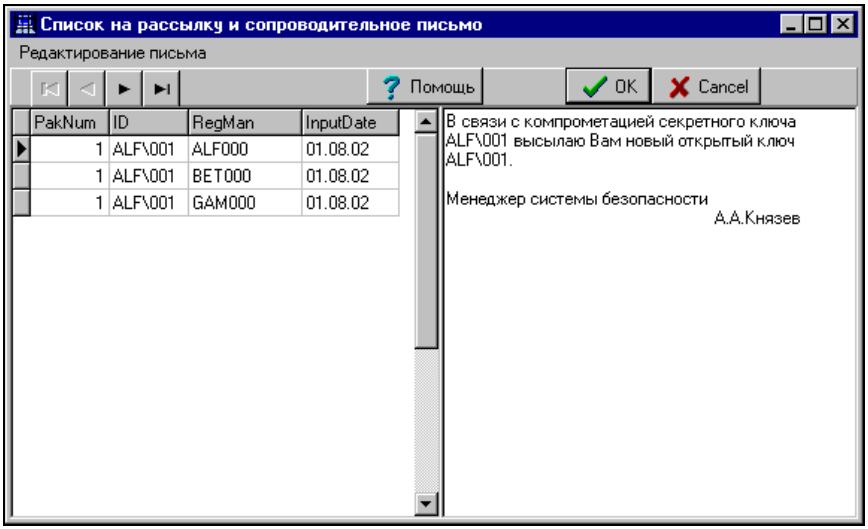

Рис. 29.8. Список на рассылку и сопроводительное письмо

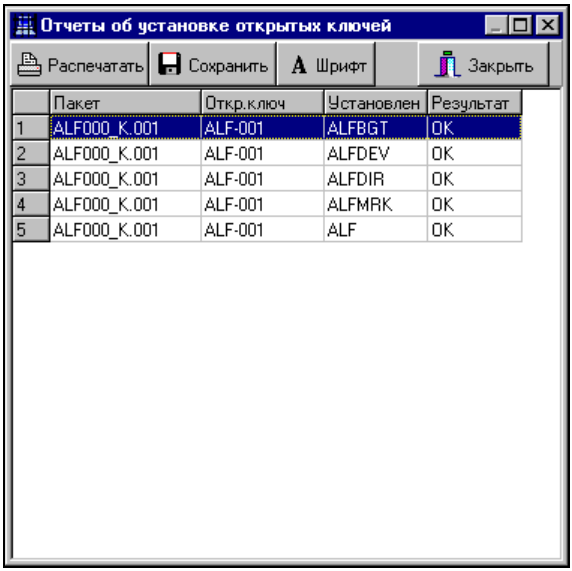

Рис. 29.9. Отчет о проведенных заменах открытых ключей

У менеджера СБ ведется база данных осуществленных рассылок, в которой также хранятся принятые отчеты. Просмотр этой базы данных осуществляется с помощью кнопки **Сведения о рассылках ключей** (рис. 29.10).

Таковы типовые операции, которые выполняет менеджер СБ в процессе эксплуатации автоматизированной системы OMEGA. Большинство действий

менеджера СБ автоматизированы, человеку, в основном, достаются только функции визуального контроля за состоянием баз данных. Более экзотические возможности, которые также предусмотрены в АРМ менеджера СБ, связаны с рассылкой участникам системы документооборота их секретных ключей. Дело в том, что операция плановой смены ключей в большой организации с широкой филиальной сетью является, как правило, достаточно дорогостоящей процедурой. Каждый пользователь должен приехать в Центр и лично получить дискету со своим секретным ключом. Выходов из этой ситуации может быть два: дать возможность участникам системы ЭД самим вырабатывать свои секретные ключи на месте или использовать систему рассылки секретных ключей. А может быть и комбинированный вариант — часть участников вырабатывает ключи самостоятельно, а часть получает по электронной почте.

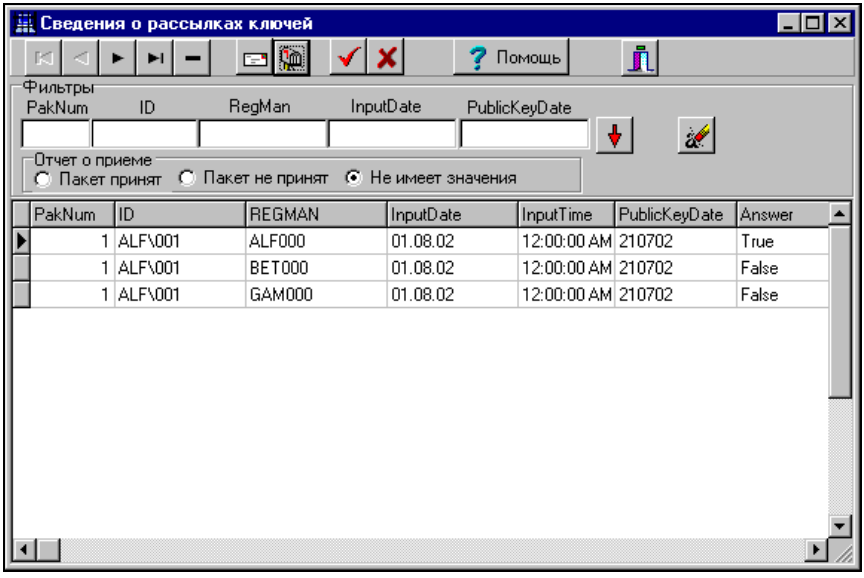

Рис. 29.10. База данных рассылок открытых ключей

Очень часто с системой документооборота работают люди, весьма далекие от проблем криптографии: бухгалтеры, банковские операционистки, сотрудники различных отделов, связанных с денежными переводами, и т. п. Заставлять таких людей самостоятельно вырабатывать свой индивидуальный секретный ключ часто чревато ошибками, сбоями в системе документооборота, вероятностью компрометации выработанного ключа. Поэтому задача рассылки выработанных в Центре секретных ключей является актуальной.

Но как зашифровать пересылаемый секретный ключ? На старом ключе шифровать нельзя — читатель, внимательно читающий эту книгу с самого начала, должен это твердо усвоить. Вернемся к *главе 2*, к мистеру Шеннону,

абсолютно стойкому шифру, получающемуся путем наложения случайной и равновероятной гаммы. Для защиты секретных ключей при их пересылке по электронной почте применяем одноразовую гамму! Одноразовая гамма может быть записана на специальную дискету, после расшифровки использованная часть гаммы автоматически стирается.

А надолго ли хватит гаммы на такой дискете? Давайте прикинем. Секретный ключ, использующийся в системе OMEGA, имеет длину 142 байта. Для пересылки секретных ключей будем, как и прежде, использовать пакеты, отсылаемые региональным менеджерам. Пусть в среднем пакете находится 100 секретных ключей общим объемом 14,2 Кбайт, а плановая смена ключей осуществляется один раз в полгода. Тогда гаммы, записанной на дискете объемом 1,44 Мбайт, хватит на 100 плановых замен, т. е. на 50 лет.

Гаммы хватит надолго, вот только условия для хранения дискеты с такой гаммой должны быть очень жесткие: чтобы, во-первых, исключить возможность ее компрометации, а во-вторых, чтобы не произошло ее порчи. Но в любом случае такой способ рассылки секретных ключей может оказаться значительно дешевле, чем, скажем, перелет человека из Владивостока в Москву за новыми секретными ключами один раз в полгода.

Менеджер системы безопасности — ключевая фигура для обеспечения информационной безопасности электронного документооборота системы OMEGA. Большинство секретных и все открытые ключи проходят через него. Это должен быть надежный, проверенный человек, без вредных привычек. Человек с чистыми руками, горячим сердцем и холодной головой.

Глава 30

Взаимодействие участников документооборота с менеджерами

Все изменения конфигурационных файлов и ключей участники документооборота получают, в основном, от менеджеров: Главного и системы безопасности. Для краткости, мы будем называть далее их Центральными менеджерами. Промежуточным звеном между Центральными менеджерами и региональными участниками являются региональные менеджеры, которые сами не принимают участия в электронном документообороте. Такая схема, как мы видели в предыдущих главах, более удобна как для Центральных менеджеров, так и для самих региональных участников, она позволяет сократить объем пересылаемой из Центра в регионы информации, например, объемы пересылаемых при плановой смене открытых ключей. Региональный пользователь или администрация могут вообще не иметь никаких дел с Центральными менеджерами, а все необходимые для функционирования системы OMEGA файлы получать только от своего регионального менеджера. Он же общается напрямую с Центром и отвечает за обслуживание своего региона.

Но в отличие от Центральных менеджеров, региональный менеджер, как правило, не создает сам конфигурационных файлов или открытых ключей. Его задача — получить пакеты с соответствующими файлами из Центра, обработать их и послать в Центр отчет о проведенных заменах. По возможности, все действия регионального менеджера автоматизированы, от человека здесь уже почти ничего не зависит.

Давайте рассмотрим автоматизированное рабочее место регионального менеджера. Основными его компонентами являются исполняемый модуль SYSTEM.EXE и конфигурационный файл SYSTEM.BAS, который готовится для регионального менеджера Главным менеджером (см. рис. 28.11). Отметим, что помимо приведенных на рис. 28.11 строк в этот файл входит перечень обслуживаемых региональным менеджером участников системы ЭД и их каталога, где расположен файл SETUP. Эта информация хранится в базах данных пользователей и администраций Главного менеджера (см. рис. 28.5).

Обмен пакетами с Центральными менеджерами региональный менеджер осуществляет с помощью собственных почтовых ящиков `=IN_BOX=` и `=OUT_BOX=`, прописанных в файле SYSTEM.BAS. Для физической пересылки пакетов может использоваться сама система OMEGA, а точнее — ее пользователи, у которых в файлах OUT.BAS и IN.BAS автоматической отправки и приема будут прописаны разделы, соответствующие отправке и приему пакетов менеджеров. Более того, в файлах SETUP таких пользователей можно предусмотреть специальный каталог для пакетов Центральных менеджеров, которая прописывается в строке `=CHANGE_IN=` (см. рис. 28.3). При наличии в сценарии работы пользователя строки `[_TestChange]` при поступлении в этот каталог файлов на экран выдается сообщение-предупреждение (рис. 30.1).

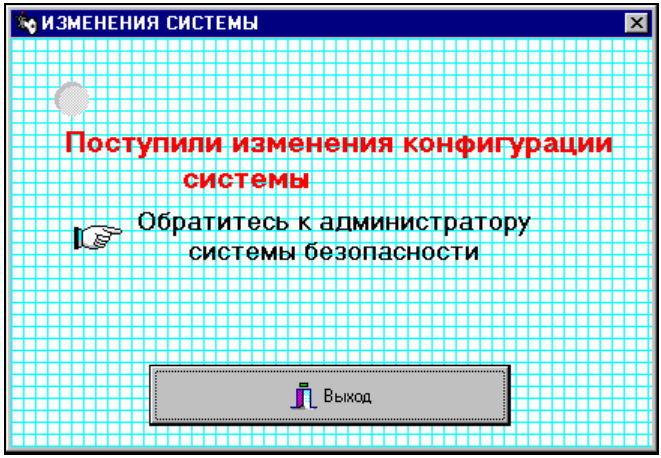

Рис. 30.1. Предупреждение об изменении конфигурации системы

Теперь пора запускать программу SYSTEM.EXE. Поскольку региональный менеджер подписывает отчеты, автоматически отправляемые в Центр, то он должен иметь индивидуальный секретный ключ для подписи отчетов. Поскольку все пакеты из Центра подписаны Центральными менеджерами, то региональный менеджер также должен иметь их открытые ключи для проверки подписи.

При запуске программы SYSTEM.EXE автоматически запрашивается секретный ключ регионального менеджера и, если в `=IN_BOX=` есть входящие пакеты от Центральных менеджеров, то проверяются их подписи. После этого на экране появляется основное меню регионального менеджера (рис. 30.2).

Набор маленьких кнопок внизу окна, изображенного на рис. 30.2, предназначен для выполнения различных операций, связанных с рассылкой секретных ключей. О такой возможности, заложенной в систему OMEGA, мы упоминали в прошлой главе.

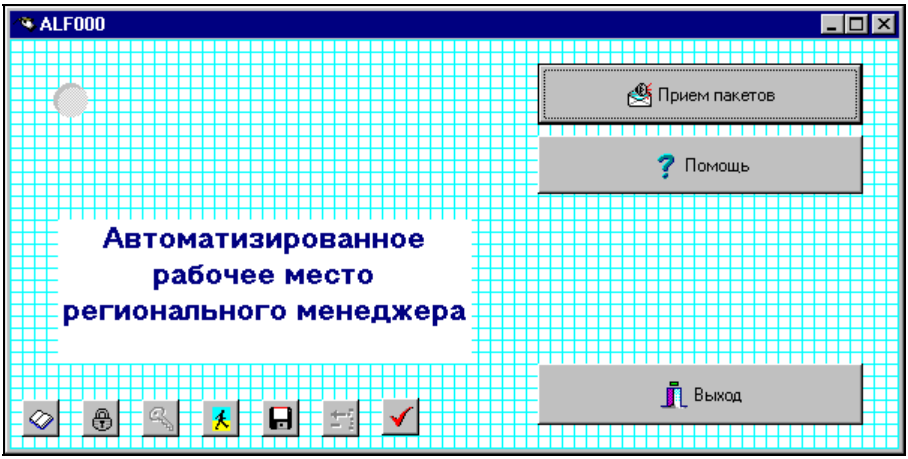

Рис. 30.2. Основное меню регионального менеджера

С помощью кнопки **Прием пакетов** региональный менеджер осуществляет типовые операции по приему пакетов с новыми конфигурационными файлами от Главного менеджера и с новыми открытыми ключами от менеджера СБ. По имени пакета программа сама определяет, от какого Центрального менеджера он поступил, и в зависимости от этого выполняет определенные действия по его обработке.

В пакете, поступившем от Главного менеджера, указано, для какого участника системы предназначены измененные конфигурационные файлы. Эта информация, а также сопроводительное письмо к пакету высвечиваются у регионального менеджера на экране (рис. 30.3).

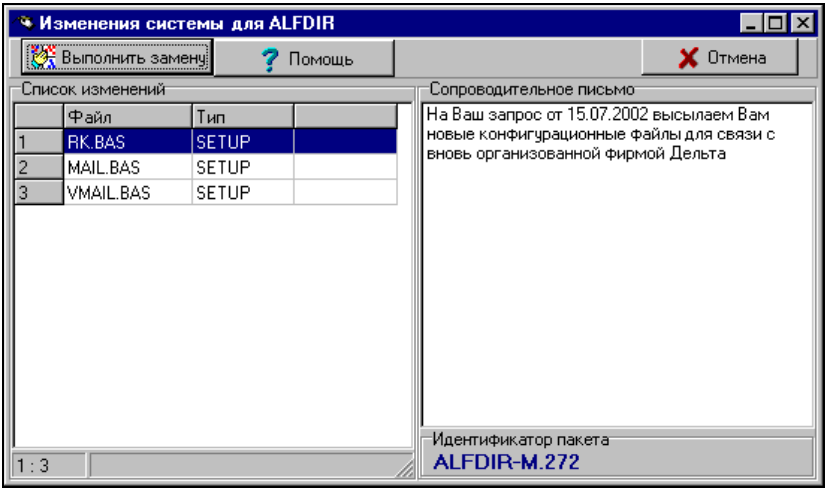

Рис. 30.3. Информация о предстоящей замене конфигурационных файлов

При осуществлении замены старые файлы сохраняются, и всегда есть возможность их восстановить. Сопроводительное письмо также сохраняется вместе со старыми файлами. После выполнения замены в `=OUT_BOX=` регионального менеджера автоматически формируется и подписывается отчет о проведенной замене. Работу человека при замене конфигурационных файлов наверное трудно отнести к разряду тяжелых.

Несколько другой интерфейс при обработке пакетов от менеджера СБ (рис. 30.4).

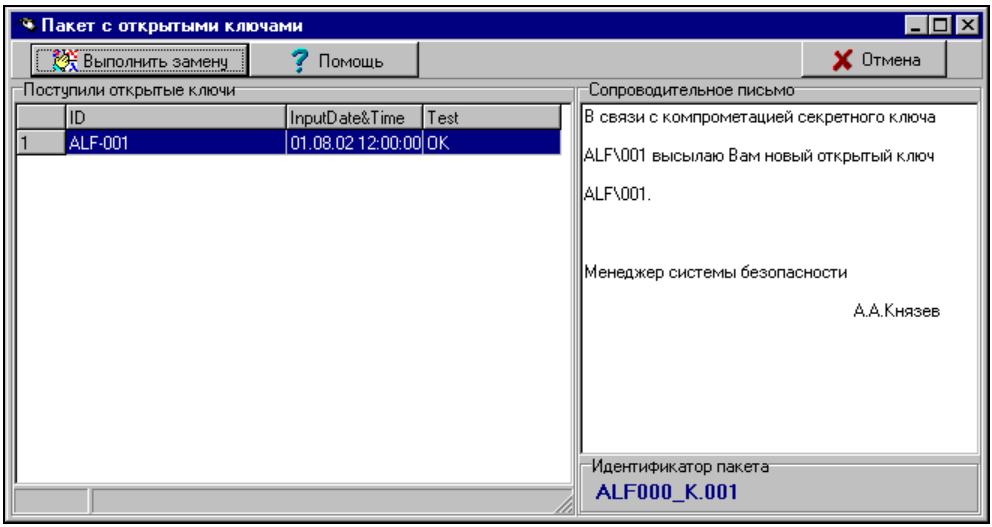

Рис. 30.4. Информация о предстоящей замене открытых ключей

Программа регионального менеджера сама должна определить, кому из обслуживаемых участников системы ЭД необходимо установить тот или иной поступивший открытый ключ. Такое определение осуществляется путем анализа соответствующих конфигурационных файлов, определяющих связи участников системы документооборота, но человек в этом процессе опять же принимает только пассивное участие. Результат выдается на экран (рис. 30.5).

Автоматически формируется и подписывается отчет о проведенных заменах открытых ключей. Старые открытые ключи также сохраняются, и есть возможность их восстановить.

Вот такие механизмы взаимодействия участников системы электронного документооборота с менеджерами. Все, что только можно, в них автоматизировано. На каждое выполненное действие отсылается отчет, который заносится в базу данных соответствующего Центрального менеджера. Жестко централизованная система, в которой однако имеется возможность дать ее

участникам некоторую степень свободы, отменив подписи на некоторых конфигурационных файлах. Нужно это делать или нет — решает сам Заказчик системы OMEGA. Отменить централизацию проще, чем разработать и ввести эффективную систему контроля за пользователями, иногда разбросанными на многие сотни километров по всей стране.

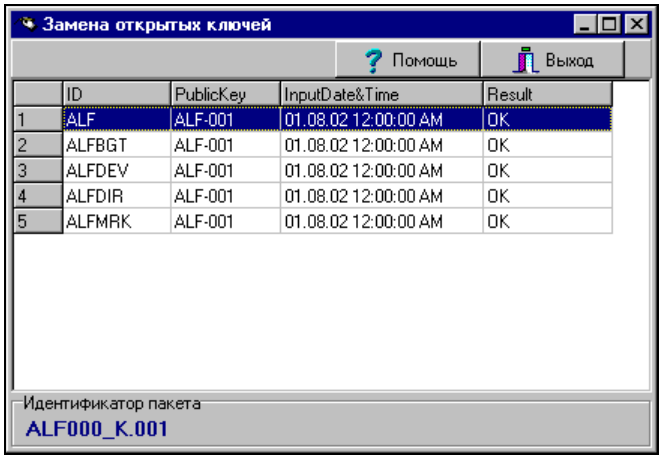

Рис. 30.5. Информация о выполненных заменах открытых ключей

В завершении этой части книги, посвященной автоматизированной системе электронного документооборота OMEGA, позволю себе сказать несколько слов об истории ее создания.

Первая мысль о создании подобной системы появилась в 1992 году при работе с Центральным банком. Ведь введенная там система защиты банковских авизо предполагалась как временная, предназначенная для снятия исключительно напряженной ситуации с фальшивыми авизо в то время. Дальше неизбежно должна была появиться современная программная система защиты банковских платежей, ориентированная на электронную почту, большой объем типовых платежных документов, максимальную автоматизацию. Для ее создания требовалось время и, самое главное, заинтересованность Заказчика. Представление о том, какая это будет система, было весьма общее, но то, что вопросы информационной безопасности должны стоять в ней на одном из первых мест, ни у кого не вызывало сомнений.

Центральный банк — огромная структура. В обычной, не экстремальной ситуации, не склонная к внедрению новых решений. Слава богу, после введения 1 декабря 1992 года системы защиты телеграфных авизо, острота ситуации с фальшивыми авизо спала. Ситуация перестала быть экстремальной, но для внедрения новых решений ЦБ оказался закрыт.

В 1995 году ко мне обратился один весьма крупный банк с предложением автоматизировать и защитить существующий у них электронный документооборот. Что было тогда более актуальным — автоматизация или защита — не могу сказать. По-видимому, и то и другое являлось достаточно актуальным. К этому времени у меня уже были достаточно хорошо отлажены криптографические программные модули, позволяющие осуществлять выработку секретных и открытых ключей, операции шифрования и подписи. Они были сведены в единую систему Криптоцентр, которую сначала предполагалось использовать в качестве основы будущей системы электронного документооборота. Но потребности банка оказались гораздо шире возможностей системы Криптоцентр, в основном требовалось автоматизировать все операции, связанные с пересылкой типовых документов по электронной почте, чего практически не было в Криптоцентре. Там все было ориентировано, в основном, на интерактивное осуществление операций шифрования и подписи. Поэтому было решено, используя отлаженные в Криптоцентре операции шифрования и подписи, создавать систему документооборота заново, учитывая потребности банка. А Криптоцентр оставить как резервное средство для проверки криптографических операций в каких-то нестандартных ситуациях.

Одних усилий разработчиков явно недостаточно для того, чтобы система заработала. Нужно еще, чтобы и у Заказчика нашлись люди, готовые к внедрению новой системы. Это очень хлопотное дело, особенно если речь идет о многофункциональной и достаточно сложной системе, какой получилась описанная здесь система электронного документооборота. Выгоды от ее внедрения появятся потом, а на первом этапе это сплошная неизвестность, нередки сбои, ошибки, доработки, тестирование. Да и к тому же необходимо убедить свое начальство в том, что дело завершится успешно. Первый Главный менеджер этой системы — Владимир Константинович Тяпкин — одновременно был одним из ее теоретиков и практиков, без него ее создание не состоялось бы.

Так в конце 1995 года появилась первая версия системы электронного документооборота. Система заработала.

Известно, как быстро растут запросы Заказчика. Первый раз Главный менеджер создавал и подписывал все конфигурационные файлы вручную, но после этого твердо заявил, что нужно автоматизированное рабочее место Главного менеджера. Естественно, что тут же с таким же запросом выступил и менеджер системы безопасности. И все последующие 5 лет основные мои усилия были направлены только на то, чтобы все в системе электронного документооборота автоматизировать, оптимизировать, добавить новые возможности, провести Upgrade, сделать покрасивее и побыстрее.

Дала ли эта система выгоды банку? Думаю, что да. Количество типовых электронных документов, ежедневно поступающих от всевозможных филиалов и отделений банка и обрабатываемых в Центральном офисе, достигает в настоящее время десятки тысяч, и трудно представить, как справиться с таким потоком вручную. Практически исключены опасные ошибки, применяющиеся средства защиты и контроля позволяют использовать дешевые способы связи по Internet, функции человека сведены к минимуму — экономия на зарплатах сотрудников.

А еще одна особенность системы OMEGA в том, что ее легко адаптировать под любые разумные пожелания Заказчика. Это оригинальная разработка, общий объем исходных текстов всех модулей уже составляет десятки мегабайт. Надеюсь, что это не предел.

Часть VI

Технология отпечатков пальцев

Глава 31. Проблема носителя секретного ключа

Глава 32. Уникальный рисунок отпечатка пальца человека

Глава 33. Формирование секретного ключа по отпечатку пальца

Глава 34. Криптосервер

Глава 35. Генератор случайных паролей по отпечатку пальца

Снять все дисководы — вот мечта управления безопасности практически любого банка. Дисковод — источник всякой компьютерной заразы, возможных утечек информации, окно во внутренний мир компьютера. Надо его заколотить!

Но в то же время дисковод — это наиболее простое устройство для ввода секретного ключа. Записывать файл с секретным ключом на дискету просто и дешево. Если снять дисковод, то как быть с вводом секретного ключа?

Так давайте запишем секретный ключ где-нибудь в укромном местечке на сервере. Засунем там его в какую-нибудь щелочку, чтоб никто не догадался, и сделаем таким образом *криптосервер*. Антикриптографические рассуждения. Все эти щелочки, сетевые протоколы, сетевая защита очень часто становятся легкой добычей для хакеров, которые пишут уйму всяких CRACK для их взлома. Гарантированную защиту может обеспечить только задача перебора всех ключей, а если мы сам ключ пишем на сервер, то вместо такого перебора хакер будет пытаться получить доступ к ключу на сервере. Задача получения доступа к ключу, записанному на сервере, связана с большим перебором? Если бы это было так, то должен был бы существовать ключ для такого перебора, который в свою очередь тоже необходимо где-то хранить. В общем, вся эта затея с криптосервером долгое время казалась мне очень скользкой, чреватой потерей гарантированной стойкости. А для потребителей секретные ключи на сервере удобны: дисководы снимаем, дискеты, которые часто выходят из строя, в качестве носителя секретного ключа не используем.

Но вот в конце 2001 года мне показали устройство типа FingerID, выпускаемое фирмой AuthenTec и предназначенное для ввода отпечатка пальца человека в память компьютера. Что такое отпечаток пальца? Это матрица (в устройстве типа FingerID — размером 128×128), которую практически невозможно точно повторить. Человек — это живое существо, абсолютно одинаково приложить палец к считывателю не может, само устройство имеет допустимые погрешности ввода, в общем, получаемые матрицы всегда различны. А заманчиво было бы использовать их в качестве секретного ключа! Матрицы ведь почти совпадают. Надо как-то нейтрализовать эти погрешности, но таким образом, чтобы сохранить распознавание свой-чужой. И, самое главное, не хранить эталонный отпечаток, а сделать такую функцию, которая по истинному отпечатку вычисляла бы правильный ключ, а наличие ложного отпечатка не давало бы практически никакой дополнительной информации о ключе. А задача вычисления ключа без истинного отпечатка была бы связана с перебором большого числа вариантов, для которого можно было бы оценить его стойкость.

С помощью такого ключа-отпечатка мы, например, шифруем индивидуальные секретные ключи пользователя, после чего помещаем их на сервер. Хакер, даже получив доступ к таким зашифрованным ключам, без истинного отпечатка не сможет их восстановить, не осуществляя трудоемкой задачи перебора. А трудоемкость задачи перебора при восстановлении ключа без отпечатка пальца постараемся оценить и представить оценки Заказчику, который и принимает решение: удовлетворяют ли они его требованиям к стойкости.

Вот основные идеи этой части книги.

Глава 31

Проблема носителя секретного ключа

В последнее время появилась масса устройств, которые подключаются к компьютеру, что позволяет использовать их в качестве потенциальных носителей секретного ключа и вообще любой ключевой информации. Понятие "ключевая информация" может быть шире, чем просто секретный ключ. Так, например, в системе OMEGA к ключевой информации мы отнесли не только индивидуальный секретный ключ пользователя, но и открытые ключи менеджеров, а также программный модуль для проверки подписи, поскольку их подмена представляет угрозу для информационной безопасности системы. При использовании в качестве носителя секретного ключа обычной дискеты туда легко можно поместить всю подобную ключевую информацию, но если мы в качестве носителя будем использовать, например, Touch Memory, то объем имеющейся в этом устройстве памяти, как правило, весьма мал, и нам придется ограничиться записью туда только секретного ключа. Кроме того, любое внешнее устройство типа Touch Memory или Smart Card может быть оставлено пользователем без присмотра или даже потеряно. Проблема компрометации ключа остается по-прежнему актуальной.

Какие еще возможны варианты носителя секретного ключа? Естественно, сам пользователь. Пароль, который он должен помнить. Много раз уже раскритикованная в этой книге система. А что, если сделать пароль сложным, генерируемым из случайных слов и фраз, достаточно длинным, для которого можно было бы более-менее точно оценить сложность его перебора? Эта идея также имеет право на существование. Давайте ее обсудим.

Здесь мы вторгаемся в область языковой лексики и грамматических правил. Для того чтобы человек смог твердо запомнить такой пароль, в нем должны соблюдаться основные и привычные человеку грамматические правила, языковые схемы, например, за подлежащим следовать сказуемое. А сами подлежащее и сказуемое случайно выбираются из некоторого словаря, отдельного для подлежащих, отдельного для сказуемых. Затем можно разукрасить подобную схему еще прилагательными, наречиями и прочими атрибутами

нашего родного и могучего русского языка. Фраза получается действительно случайной, соблюдение в ней грамматических правил способствует ее запоминанию. Вот только останется ли пользователь доволен таким подходом? Ведь для более-менее приличной стойкости (не менее 10^{20}) таких фраз должно быть несколько, и их общий объем будет около 100 букв, которые, помимо того, чтобы их помнить, нужно будет всякий раз по памяти вручную вводить с клавиатуры. Занятие реальное, но малорадостное. И, главное, нет абсолютно никакой гарантии, что пользователь не запишет этот пароль где-нибудь у себя в записной книжке. Для верности, чтобы не забыть. А записную книжку как-нибудь забудет на своем рабочем месте около телефона.

В этом смысле отпечаток пальца явно надежнее. При всем своем желании его не потеряешь, не запишешь в записную книжку, да и помнить его нет надобности. Правда, человек оставляет отпечатки на разных предметах: на дверной ручке, стакане, портфеле и т. д., в точности как Штирлиц. Но, во-первых, выпускаемые устройства для ввода отпечатка пальца в компьютер, в частности, FingerID, требуют "живого" пальца, они реагируют на его температуру, и прикладывание к ним бумажки с картинкой не дает возможности ввода этой картинки. Во-вторых, для вычисления секретного ключа по отпечатку пальца, как это будет показано в дальнейшем, требуется большая точность, и потенциальный злоумышленник, снимая отпечаток пальца со стакана, должен будет снять его с ювелирной точностью, а такое не всегда возможно.

Конечно же, использование отпечатка пальца в качестве секретного ключа — это новое и в настоящее время пока еще экзотическое занятие. Чисто математические задачи тесно переплетаются в нем с особенностями узоров отпечатка, допустимыми углами наклона линий, вероятностью близких отпечатков у двух разных людей и прочими биологическими особенностями строения одной из частей человеческого тела. Постараемся в дальнейшем не вдаваться в эти подробности. Отпечаток пальца — это случайная матрица определенного размера. Она может строиться с помощью различных устройств (помимо FingerID на рынке в настоящее время существует и устройство-мышка BioLink), но выбор устройства определяет только размер матрицы и точность ее изображения. Для дальнейших рассуждений о возможности использования отпечатка пальца в качестве носителя секретного ключа постараемся найти некоторый общий универсальный подход, в котором в качестве аксиомы примем следующее требование:

> Секретный ключ должен вычисляться по отпечатку пальца в условиях, когда никакой эталонный отпечаток в явном или замаскированном виде в компьютере не хранится.

Наличие эталонного отпечатка, пусть даже в замаскированном виде, сразу же перечеркивает возможность достижения гарантированной стойкости. Усилия злоумышленника будут направлены в этом случае не на решение

практически неразрешимой переборной задачи, а на добывание эталонного отпечатка. Если эталонный отпечаток найден, то злоумышленник оказывается практически в тех же условиях, что и законный пользователь.

Если нет эталонного отпечатка, то возможна ли какая-то информация о нем? Вот здесь уже есть о чем поговорить. Давайте вспомним Диффи и Хеллмана и систему с открытым распределением ключей. Содержит ли открытый ключ какую-то информацию о самом секретном ключе? Безусловно, да. Но задача *вычисления* секретного ключа по известному открытому ключу требует перебора достаточно большого числа вариантов, который невозможно осуществить за реальное время. Поэтому открытый ключ посылается в открытом виде по электронной почте с учетом того, что потенциальный злоумышленник сможет его перехватить. Но восстановить по нему секретный ключ все равно не сможет.

А что если для отпечатков пальцев попробовать найти некоторое подобное решение? Найти такую функцию от эталонного отпечатка, в каком-то смысле аналог открытого ключа, с помощью которой при наличии истинного отпечатка можно было бы сравнительно легко вычислить секретный ключ, а при отсутствии истинного отпечатка вычисление ключа потребовало бы трудоемкого перебора, неосуществимого за реальное время. Открытый "отпечаточный" ключ хранить в общедоступном месте в предположении, что потенциальный злоумышленник имеет к нему доступ, а сам эталонный отпечаток нигде не хранить. Тогда истинный пользователь, обладая истинным отпечатком пальца, сумеет вычислить требуемый секретный ключ (конечно, не сам пользователь: за него это сделает программа), а потенциальный злоумышленник, не имея истинного отпечатка пальца, столкнется с неосуществимой задачей перебора. Вот примерный круг вопросов, которые возникают при попытке использовать отпечаток пальца в качестве носителя секретного ключа.

Один подобный подход к решению этой задачи и будет изложен в этой части книги.

Глава 32

Уникальный рисунок отпечатка пальца человека

В начале этой главы давайте поговорим немного подробнее об устройстве типа FingerID, предназначенном для ввода отпечатка пальца человека в компьютер (рис. 32.1).

Рис. 32.1. Устройство типа FingerID, предназначенное для ввода отпечатка пальца человека в компьютер

Оно подключается к USB-порту компьютера. Для его работы необходимо специальное программное обеспечение, состоящее из сервера устройства (модуль FLSERVER.EXE) и динамической библиотеки FLRPC.DLL, содержащей основные процедуры для работы с устройством. В этой библиотеке имеются следующие процедуры, необходимые для получения отпечатка пальца в виде BMP-файла:

- `FingerLocInit` — инициализация устройства;
- `FingerLocOpenStream` — открыть поток для считывания отпечатка;

- `FingerLocGetCurrentImage` — получить текущее изображение отпечатка;
- `FingerLocConvertRawImageToOSBmp` — преобразовать текущее изображение в BMP-формат и некоторые другие.

Проблема заключается в том, что процедуру `FingerLocGetCurrentImage` получения текущего изображения отпечатка можно успешно осуществить не сразу, а после нескольких мгновений, в течение которых палец разогреет считыватель. Да и само изображение не сразу получается самым хорошим, возможно несколько плохих изображений, пока палец недостаточно разогрел считыватель. Все это приводит к тому, что необходимо создавать специальный интерфейс для ввода качественного и пригодного для криптографических целей отпечатка пальца человека в компьютер. Идеи такого интерфейса очень простые: пользователь прикладывает палец к считывателю, отпечаток отображается на экране и пользователь его визуально контролирует. Когда, на взгляд пользователя, отпечаток является наиболее четким, он снимает палец со считывателя. Момент снятия отпечатка пальца со считывателя опять же определяется по коду возврата процедуры `FingerLocGetCurrentImage`. Отображение в момент снятия пальца со считывателя поступает на криптографическую обработку.

С помощью устройства типа FingerID и этих процедур можно построить BMP-файл, содержащий черно-белый (с серыми оттенками) рисунок отпечатка пальца человека размером 128 × 128 точек (рис. 32.2).

На рис. 32.2 видны небольшие помехи, накладываемые на рисунок за счет устройства FingerID. Иногда эти помехи могут быть более значительными (рис. 32.3).

Рис. 32.2. Рисунок отпечатка пальца человека, полученный с помощью устройства FingerID

Рис. 32.3. Рисунок отпечатка пальца с помехами, вызванными устройством FingerID

Давайте сначала изучим контрастность изображения отпечатка пальца и постараемся ответить на вопрос: а можно ли от изображения с серыми оттенками перейти к строго черно-белому изображению? Я написал специальную программу для изучения получающихся отпечатков. В частности, она позволяет получать статистику встречаемости различных значений в матрице, соответствующей отпечатку. Далее на рис. 32.4 приведен пример одной из таких статистик.

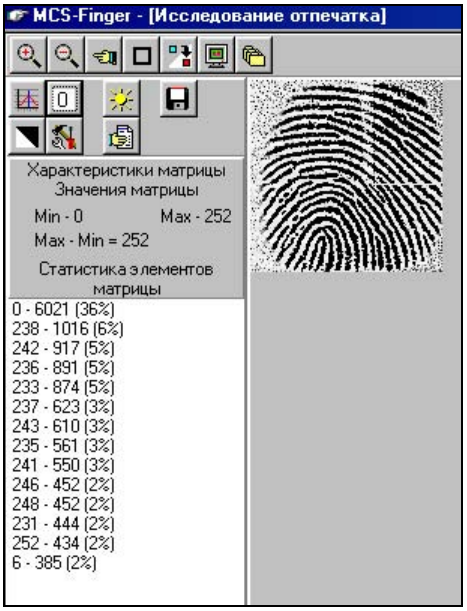

Рис. 32.4. Статистика элементов матрицы отпечатка пальца (указательный палец правой руки)

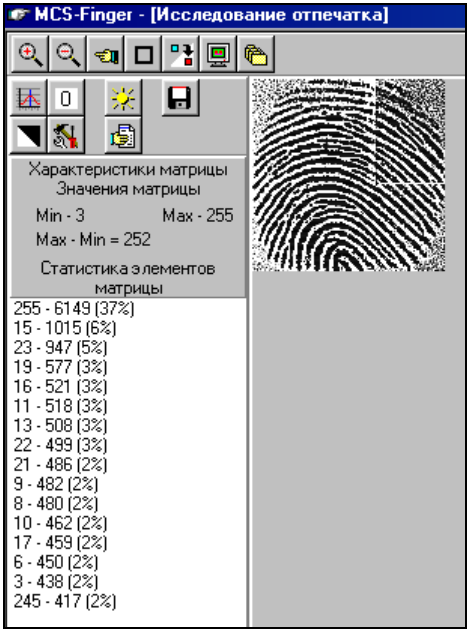

Рис. 32.5. Статистика элементов матрицы для отпечатка другого пальца одного и того же человека (средний палец правой руки)

В контрастном черно-белом BMP-изображении символ 0 соответствует самому черному цвету, а 255 — самому белому. Самое удивительное состоит в том, что для разных пальцев диапазон встречающихся значений оттенков в этой матрице оказывается совершенно различным! На рис. 32.5 приведен пример отпечатка другого пальца одного и того же человека. Обратите внимание, насколько различны диапазоны значений матрицы.

По-видимому, контрастность связана с такими показателями, как температура отдельных точек пальца или влажность. Но в любом случае это очень ненадежные показатели и от них лучше избавиться. Строить криптографические параметры целесообразно только по узору линий на отпечатке пальца. А тогда серую, с оттенками матрицу необходимо привести к строго черно-белой. Для этого используем некоторое пороговое значение, до которого все точки объявляем черными, а после — белыми.

Для работы с изображением отпечатка пальца в своих программах, написанных на Borland C++ Builder, я использовал имеющийся в этом языке механизм `TMemoryStream` (поток в памяти). Изображение отпечатка записывается в этот поток, затем с помощью различных функций работы с потоком его легко вывести на экран, записать в файл, модифицировать и т. п. Приведение матрицы изображения отпечатка к строго черно-белой я начал с ее "приведения к нулю", т. е. к вычитанию из всех элементов матрицы ее минимального значения.

Листинг 32.1. Приведение к нулю матрицы отпечатка пальца

```
TMemoryStream *MS0;
void __fastcall TFinger::MakeImage0()
{
  unsigned char MyIm[128]; // Массив для анализа изображения
  int i,j;
  unsigned char PMin=255; // Переменная для минимального значения матрицы
  // Переводим в начало указатель потока
  MS0->Seek(0,soFromBeginning);
  // Считываем BMP-заголовок
  MS0->Read(Header,iHeaderSize+14);
  // Image1 — изображение отпечатка на экране
  // Его можно увеличивать и уменьшать
  for(i=0;i<Image1->Picture->Bitmap->Height;i++)
  {
    MS0->Read(MyIm,Image1->Picture->Bitmap->Width);
    for(j=0;j<Image1->Picture->Bitmap->Width;j++)
      if(MyIm[j]<PMin)
```

```
        PMin=MyIm[j];
  }

  // Теперь вычитаем из всех элементов матрицы
  // вычисленное минимальное значение и
  // составляем новое изображение
  MS0->Seek(0,soFromBeginning);
  MS0->Read(Header,iHeaderSize+14);

  for(i=0;i<Image1->Picture->Bitmap->Height;i++)
  {
     MS0->Seek(0,soFromCurrent);
     MS0->Read(MyIm,Image1->Picture->Bitmap->Width);
     for(j=0;j<Image1->Picture->Bitmap->Width;j++)
       MyIm[j]-=PMin;
     MS0->Seek(-Image1->Picture->Bitmap->Width,soFromCurrent);
     MS0->Write(MyIm,Image1->Picture->Bitmap->Width);
  }
  // Указатель потока переводим в начало. Он еще пригодится.
  MS0->Seek(0,soFromBeginning);
  // Процедура получения статистики матрицы. Тоже пригодится.
  GetStat();
}
```

Вот листинг программной реализации процедуры получения статистики матрицы.

Листинг 32.2. Получение статистики матрицы отпечатка пальца

```
// Массив для частот встречаемости смволов
unsigned int Stat[256];
// Массив для переупорядоченных по частоте встречаемости символов
unsigned char StatI[256];

void __fastcall TFinger::GetStat()
{
  unsigned char MyIm[128];
  int i,j,k;

  for(i=0;i<256;i++)
```

```
  {
    Stat[i]=0;
    StatI[i]=i;
  }

  // Вычисляем частоты встречаемости символов
  MS0->Seek(0,soFromBeginning);
  MS0->Read(Header,iHeaderSize+14);
  for(i=0;i<Image1->Picture->Bitmap->Height;i++)
  {
    MS0->Read(MyIm,Image1->Picture->Bitmap->Width);
    for(j=0;j<Image1->Picture->Bitmap->Width;j++)
      Stat[MyIm[j]]++;
  }
  // Переупорядочиваем частоты встречаемости по убыванию
  MS0->Seek(0,soFromBeginning);
  for(i=0;i<256;i++)
    for(j=i;j<256;j++)
      if(Stat[j]>Stat[i])
      {
        k=Stat[i];
        Stat[i]=Stat[j];
        Stat[j]=k;
        k=StatI[i];
        StatI[i]=StatI[j];
        StatI[j]=k;
      }
}
```

В результате работы процедуры `GetStat()` мы имеем два массива: `Stat`, содержащий частоты встречаемости символов, расположенные по убыванию, и `StatI`, содержащий сами символы, расположенные по убыванию частот их встречаемости. Теперь можно строить строго черно-белое изображение отпечатка пальца.

Листинг 32.3. Процедура построения строго черно-белого изображения отпечатка

```
void __fastcall TFinger::MakeFineImage()
{
  unsigned char MyIm[128];
```

```
  int i=0,j;
  unsigned char G2; // Переменная для порогового значения

  // Вычисляем пороговое значение для черной точки
  // Оно не должно превосходить 128
  G2=0;
  while((Stat[i]*100)/(128*128)>0)
  {
    if(StatI[i]>G2&&StatI[i]<128)
      G2=StatI[i];
    i++;
  }
  G2++;

  // Приводим изображение к строго черно-белому
  MS0->Seek(0,soFromBeginning);
  MS0->Read(Header,iHeaderSize+14);

  for(i=0;i<Image1->Picture->Bitmap->Height;i++)
  {
    MS0->Seek(0,soFromCurrent);
    MS0->Read(MyIm,Image1->Picture->Bitmap->Width);
    for(j=0;j<Image1->Picture->Bitmap->Width;j++)
    {
      if(MyIm[j]<G2)
        MyIm[j]=0;
      else
        MyIm[j]=0xFF;
    }
    MS0->Seek(-Image1->Picture->Bitmap->Width,soFromCurrent);
    MS0->Write(MyIm,Image1->Picture->Bitmap->Width);
  }
  MS0->Seek(0,soFromBeginning);
}
```

Далее на рис. 32.6 показан пример приведения изображения отпечатка пальца к строго черно-белому виду.

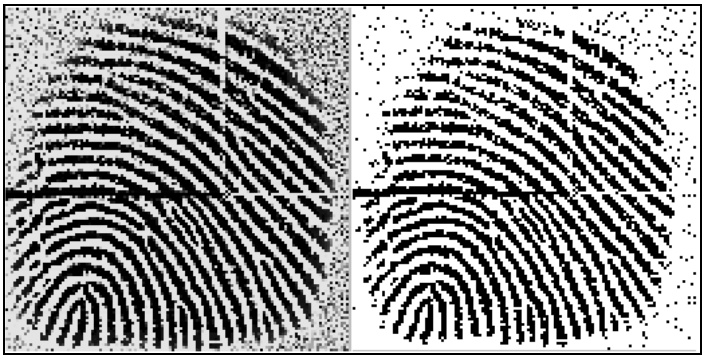

Рис. 32.6. Результат приведения изображения отпечатка пальца к строго черно-белому виду

На строго черно-белом рисунке мы уже имеем только линии, составляющие уникальную картинку отпечатка пальца человека. Помехи, создаваемые, в том числе, и считывателем, в некоторой степени удалось сгладить. Но не до конца. Основной источник неточностей — сам человек. Он не может абсолютно одинаково два раза приложить палец к считывателю. Общая картинка будет почти такой же, но частности, детали — разные. На рис. 32.7 дан пример двух отпечатков одного и того же пальца человека, уже приведенных к строго черно-белому виду.

Рис. 32.7. Отпечаток одного и того же пальца, снятый в разные моменты времени

Внимательно присмотревшись, можно увидеть небольшую разницу. Рисунок отпечатка на левой картинке чуть смещен вверх, отличаются углы наклона. Конечно же, криминалист (да и не только он) легко обнаружит в обоих этих отпечатках сходство, позволяющее сделать однозначный вывод об их идентичности. Но в криптографических приложениях задача другая. Эталонного отпечатка нет и быть не должно. Задачей является не выяснение личности владельца отпечатка, а вычисление по черно-белой матрице уникального секретного ключа. Обо всем этом — в следующей главе.

Глава 33

Формирование секретного ключа по отпечатку пальца

Итак, в точности повторить матрицу отпечатка пальца в разные моменты времени человек не может. А что может? Насколько точно можно повторить не саму матрицу, а какие-то ее фрагменты? Можно ли построить процедуру более-менее точного определения расположения фрагмента, вырезанного с эталонного отпечатка, во вновь введенном отпечатке? Вот примерный круг вопросов, на которые мы попытаемся прежде всего найти ответ.

Что такое фрагмент матрицы? Это некоторый ее кусочек, не обязательно квадратный. Но давайте, например, для начала будем рассматривать фрагменты размером 16×16 у нашей матрицы, размером 128×128. Для того чтобы "живьем" посмотреть на различные фрагменты и их особенности, я сделал специальную программу MCS-Finger. На рис. 33.1 приведен пример фрагмента размером 16×16 матрицы отпечатка пальца. Здесь и всюду далее матрица отпечатка пальца уже приведена к описанному в прошлой главе строго черно-белому изображению.

Каким показателем можно охарактеризовать выделенный фрагмент? Его уникальностью. Что это такое? Давайте попробуем "протащить" выделенный фрагмент по всей матрице и подсчитать в каждом случае количество несовпадений черно-белых точек. Если все точки совпадают, то этот показатель равен 0 (это может быть истинное расположение фрагмента), если найдено 68 несовпадений, то в этом месте показатель равен 68, и т. д. В левом нижнем углу рис. 33.1 видны результаты такого подсчета (X,Y — координаты левого верхнего угла фрагмента, V — количество несовпадающих точек). Они упорядочены по возрастанию значения V, и на экран выведены первые 8 результатов. Чем выше разница между первым (истинным) положением фрагмента и значением следующей строки, тем более уникальным является фрагмент. На рис. 33.1 явно видно, что в качестве фрагмента выбран наиболее "замысловатый" участок матрицы. Для других, более "типовых" фрагментов этот показатель оказывается ниже (рис. 33.2).

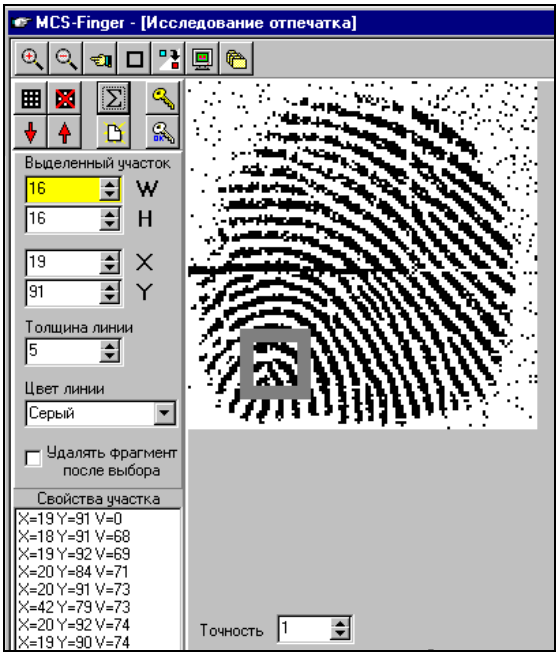

Рис. 33.1. Фрагмент матрицы отпечатка пальца с высокой уникальностью

Рис. 33.2. Фрагмент матрицы отпечатка пальца с низкой уникальностью

Уникальность фрагмента на рис. 33.2 почти в два раза ниже, чем на рис. 33.1.

Теперь проделаем следующую процедуру. Запомним выделенный фрагмент и возьмем другой отпечаток того же пальца. Постараемся точно таким же способом, которым мы подсчитывали уникальность фрагмента, найти наилучшее положение для сохраненного фрагмента на новой матрице. На рис. 33.3 приведен результат выполнения подобной операции для фрагмента рис. 33.1.

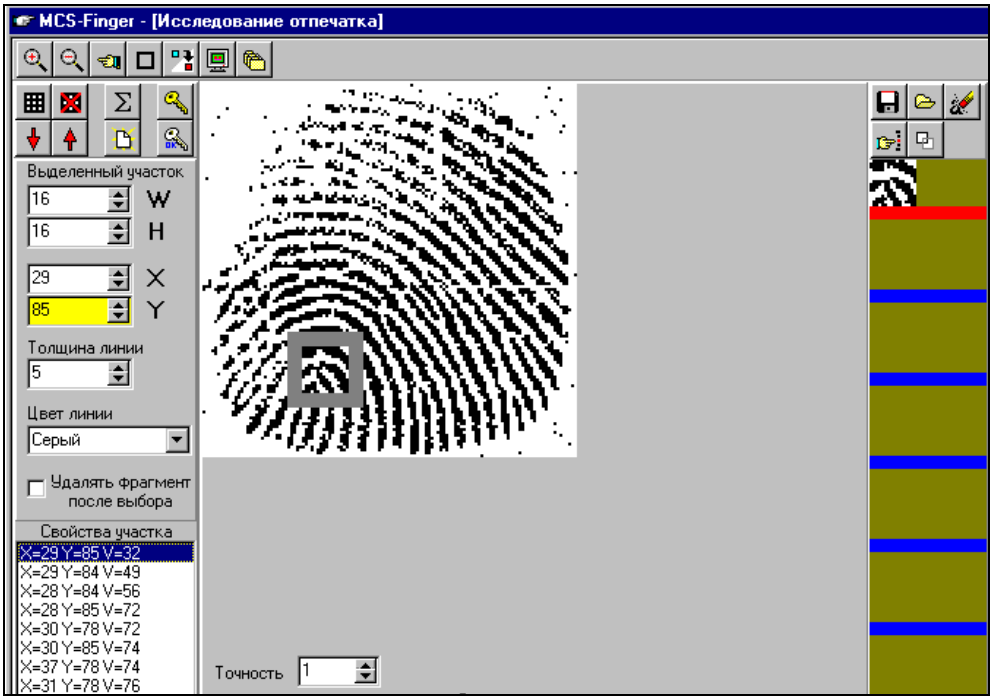

Рис. 33.3. Наилучшее расположение "хорошего" фрагмента на другом отпечатке

Почти угадали! Визуально хорошо видно, что программа почти точно подобрала расположение фрагмента. Новый отпечаток смещен вверх относительно предыдущего, но, несмотря на это, фрагмент подобран в нужном месте. Да и следующие два варианта отличаются сдвигами на одну только точку. Еще чуть-чуть — и доберемся до криптографических параметров!

А вот результат выполнения подобной операции для "плохого" фрагмента, приведенного на рис. 33.4.

Тоже близко к истине, но "со скрипом". Обратите внимание на значения свойств в левом нижнем углу рис. 33.4: привязка фрагмента более "размазана", нет явного скачка в значениях, как это было на рис. 33.3.

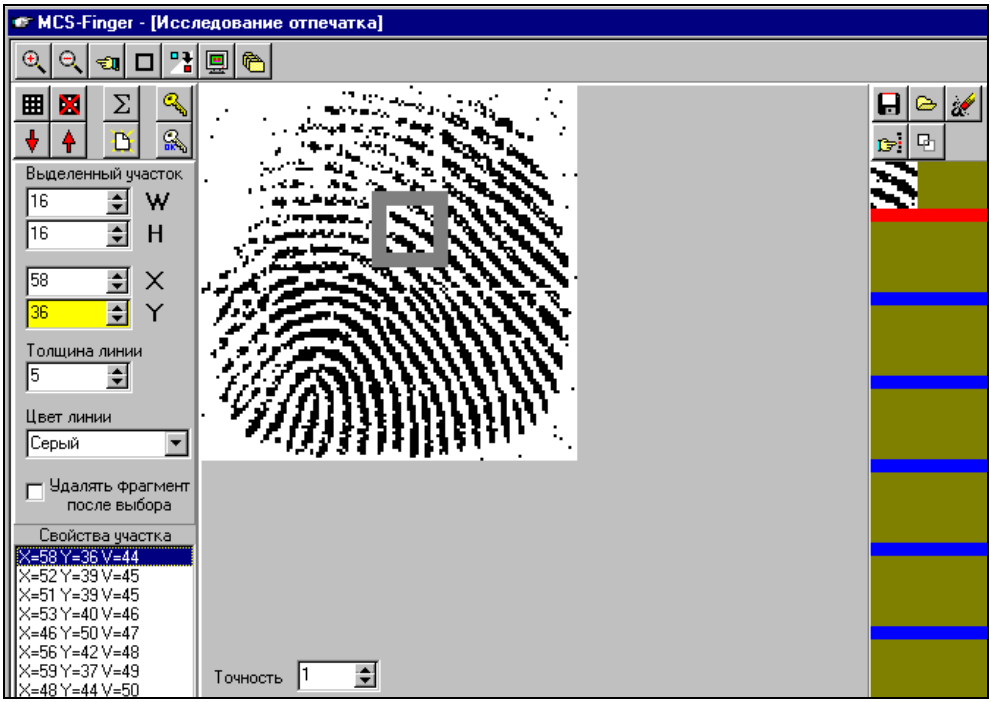

Рис. 33.4. Наилучшее расположение "плохого" фрагмента на другом отпечатке

А теперь — самое главное в предлагаемой в этой книге технологии отпечатков пальцев. Посчитаем (по модулю 256) разность координат левого верхнего угла фрагментов на первом и втором отпечатке.

Для первого отпечатка:

$$x_1 - x_2 = 19 - 43 = 232,$$

$$y_1 - y_2 = 91 - 46 = 45.$$

Для второго отпечатка:

$$x_1 - x_2 = 29 - 58 = 227,$$

$$y_1 - y_2 = 85 - 36 = 49.$$

До криптографических ключей осталось чуть-чуть. Либо получше выбрать второй фрагмент (мы выбрали один из самых плохих), либо в качестве ключей использовать не саму разность, а округленный по правилам округления результат ее деления на значение, которое далее будем называть *точностью* ключа. В данном примере в качестве такой точности можно выбрать число 10: в первом и во втором случае разности будут одинаковыми.

Чтобы двигаться дальше, давайте в этом месте попытаемся осмыслить предлагаемую технологию вычисления криптографических ключей по отпечатку пальца, ее сильные и слабые стороны, особенности реализации и ввода.

Итак, в качестве криптографических ключей предлагается использовать взаимное расположение нескольких фрагментов матрицы отпечатка пальца относительно друг друга. Эталонный отпечаток нигде не хранится, вместо него в специальном файле (пока просто назовем его *FCS-файлом*, но далее обязательно поговорим о нем подробнее) хранится набор фрагментов, вырезанных с эталонного отпечатка. С какой задачей столкнется хакер, решивший взламывать такую систему? Фрагментов не два, а несколько, например, 8, каждый — размером 16 × 16. Ниже на рис. 33.5 для такой ситуации приведено наглядное представление задачи: чистая матрица 128 × 128, на которую положили 8 известных фрагментов 16 × 16 каждый.

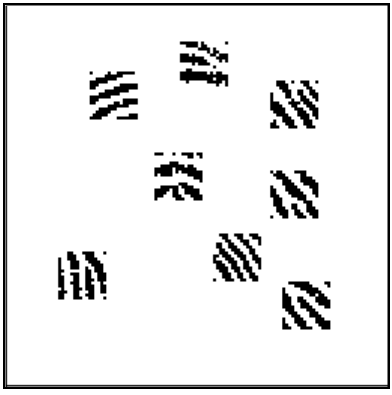

Рис. 33.5. Исходные данные при попытках вскрыть секретный ключ

Нужно найти правильное расположение этих фрагментов относительно друг друга. На мой взгляд, намного более трудная задача, чем, скажем, подобрать пароль. Сразу же отметим, что с критерием отсева ложных вариантов проблемы нет: его необходимо сознательно добавить в FCS-файл, чтобы истинный пользователь смог правильно ввести свой отпечаток пальца.

Конечно, даже визуально видна несовместимость расположения некоторых фрагментов относительно друг друга. Но все равно: перебор большого числа вариантов неизбежен! Весь вопрос в том, как его осуществлять, по каким критериям отсеивать ложные варианты? Здесь возможны две альтернативы: или простой алгоритм перебора (например, тотальный перебор), но одновременно и большое количество опробуемых вариантов, или сложный алгоритм перебора (с учетом особенностей линий на фрагментах, их толщины, углов наклона и т. п.), но за счет усложнения алгоритма можно будет достичь некоторого сокращения числа опробуемых вариантов.

Давайте оценим трудоемкость тотального перебора.

Положим x_1, y_1 — координаты левого верхнего угла первого фрагмента, x_2, y_2 — второго, ..., x_8, y_8 — последнего, восьмого. В качестве ключа, к примеру, возьмем следующие значения (операции вычитания — по модулю 256):

$$k_1 = x_2 - x_1,$$
$$k_2 = y_2 - y_1,$$
$$k_3 = x_3 - x_1,$$
$$k_4 = y_3 - y_1,$$
$$\ldots,$$
$$k_{13} = x_8 - x_1,$$
$$k_{14} = y_8 - y_1.$$

Получился ключ длиной 14 байт. Трудоемкость тотального перебора 14 байт составляет $(2^8)^{14} \cong 10^{33}$. Но, очевидно, эту оценку можно немного понизить. Ведь каждый байт ключа получается за счет разности верхних координат фрагментов, поэтому опробовать можно не байты ключа, а расположения фрагментов в матрице. Для расположения каждого фрагмента имеется $(128 - 16)^2 \cong 10^4$ вариантов. Всего фрагментов 8, общее число вариантов перебора составляет 10^{32}. Это, конечно, весьма грубые оценки, в них не учитывается то, что фрагменты, к примеру, не должны зацепляться друг за друга, поскольку этот факт легко установить по фрагментам.

Теперь от строгих рассуждений перейдем к несколько абстрактным. Попробуем уточнить эти грубые оценки. Какая часть матрицы размером 128×128 является "запретной" для фиксированного фрагмента? Это, скорее всего, углы и края матрицы. Какова площадь белых полей матрицы отпечатков пальцев? На глазок, по рисункам из этой главы — не более 1/3. Поэтому для количества "допустимых" вариантов расположения одного фрагмента на матрице примем не 10^4, а $10^4/3$. Тогда общее количество опробуемых ключей сократится до $10^{32}/3^8 \cong 10^{28}$. Надо заметить, что все равно немалая величина, сопоставимая с трудоемкостью решения задачи логарифмирования в конечных полях.

Ну, а чем же законный пользователь отличается от злоумышленника? Своим пальцем, с которого вырезались эти фрагменты. Он всегда имеет возможность ввести почти такую же эталонную матрицу в компьютер и дать программе сильную подсказку о том, как располагать фрагменты. Находятся наилучшие варианты расположения фрагментов на введенном отпечатке — см. рис. 33.6.

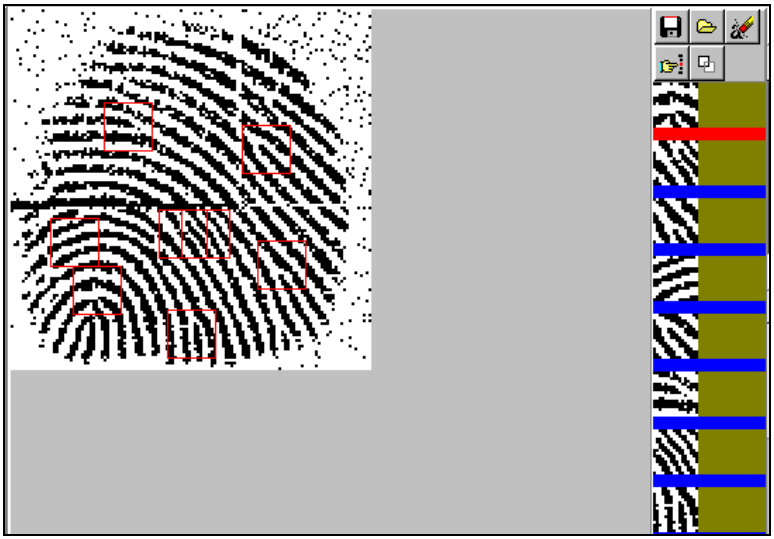

Рис. 33.6. Фрагменты, найденные в истинном отпечатке

Ну и, может быть, сравнительно небольшой перебор самых хороших из них. А критерий отсева закладывается в FCS-файл вместе с фрагментами отпечатков: это может быть проверочная комбинация, вычисляемая, например, с помощью процедуры `CryptBlock`. В который уже раз оказывается полезной эта процедура!

Кстати, на рис. 33.6 видно, что приведенный пример набора фрагментов — плохой. Два фрагмента цепляются друг за друга. А легко ли визуально по рис. 33.5 найти, какие именно? Сравнительно легко построить программу проверки опасных зацеплений. Моя программа выдала такое предупреждение (рис. 33.7), нумерация блоков начинается с 0.

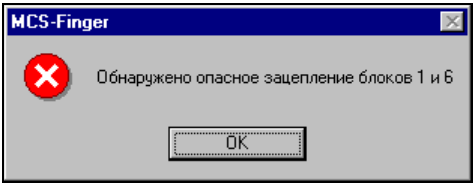

Рис. 33.7. Предупреждение об опасном зацеплении блоков

А вообще подобная идея в криптографии известна давно, в частности, в аппаратуре для защиты телефонных переговоров. Называется она — мозаичная перестановка сигнала. Аналоговый сигнал разбивается на части, и они переставляются в зависимости от секретного ключа. Только там, как правило, переставляются *все* части сигнала, а здесь — очень малая часть матрицы отпечатка пальца.

Но все-таки для криптографических ключей нужны очень точные значения разностей, а в данном случае полное их совпадение на эталонном и вновь введенном отпечатке — большая редкость. Об одном способе убрать эти различия, но при этом понижающем стойкость такой системы, мы уже говорили ранее. Это вместо точных значений разности использовать деление на некоторую константу, называемую точностью. Другой способ тоже достаточно очевиден: в FCS-файле использовать не один, а несколько наборов фрагментов, снятых с разных отпечатков одного и того же пальца. Не сумели вычислить ключ по одному — пробуем по другому набору фрагментов. Облегчение для хакера не особо сильное, а для истинного пользователя — ощутимое.

А теперь пора подробнее поговорить об FCS-файлах, содержащих наборы фрагментов, по которым при наличии истинного пальца вычисляется секретный ключ. При его построении пользователь сам имеет возможность определить, что для него важнее: стойкость или удобство ввода, и установить оптимальный компромисс между ними. Сделав в FCS-файле наборы из небольшого (3–4) числа фрагментов, пользователь получает сравнительно небольшую оценку стойкости (примерно 10^{10}), но достаточно высокую вероятность правильного распознавания отпечатка пальца. Немного привыкнув к работе с устройством, в этом случае без особого труда удается вводить правильный отпечаток с первого раза. Наличие в наборах порядка 6 фрагментов повышает стойкость до 10^{15}, однако правильного ввода удается достичь в среднем с 2–3 попыток. И, наконец, для особо конфиденциальных задач можно использовать в FCS-файле все 8 фрагментов, достигая при этом оценок стойкости 10^{25}–10^{28}. Для правильного ввода потребуется 5–7 попыток. Эти оценки, безусловно, весьма приблизительные, зависящие от аккуратности пользователя и от качества устройства для считывания отпечатка пальца, но общую тенденцию они отражают верно: платой за высокую стойкость является ограничение удобств для пользователя.

Итак, при составлении FCS-файла пользователь первым делом должен определить, сколько фрагментов отпечатка пальца будет в наборе, их размер, точность ключа. Эти величины позволят приблизительно оценить стойкость.

Но секретный ключ, вычисляемый по фрагментам отпечатка пальца, не всегда может подходить к той или иной криптографической системе. В начале этой части мы уже упоминали о криптосервере, когда секретные ключи криптографической системы некоторым образом зашифровываются и помещаются на сервер. Вот для таких целей очень удобен секретный ключ, вычисляемый по отпечатку пальца. Некоторый секретный ключ (или набор параметров, относящихся к секретным ключам) шифруется с помощью ключа, вычисленного по отпечатку пальца, и в зашифрованном виде записывается в FCS-файл. Сам FCS-файл при этом в открытом виде пишется на сервер — вот и готовый криптосервер!

Ну и, наконец, еще одна очень важная составляющая часть FCS-файла — проверочная комбинация для вычисления истинного варианта ключа по отпечатку пальца. Мы уже упоминали о ней. Проблема неточности ввода очень актуальна, даже при истинном отпечатке пальца неизбежен небольшой перебор, проверочная комбинация позволяет выделить истинный вариант. Заметим, что даже если наборов фрагментов несколько, сняты они все могут быть с разных вариантов отпечатка, но проверочная комбинация для всех будет одна и та же. Проверочная комбинация определяется исключительно по правильному секретному ключу, такой вариант — единственный.

Теперь давайте поговорим о программных средствах, предназначенных для изучения отпечатка пальца и построения по ним FCS-файлов. В начале главы я уже упоминал о программе MCS-Finger, которую написал одной из первых для всех этих целей. Приведенные ранее рисунки — часть интерфейса при работе с этой программой.

В MCS-Finger используются несколько баз данных. Одна база данных — для пользователей, другая — для отпечатков различных пальцев их рук, третья — для наборов фрагментов, из которых потом строятся FCS-файлы. При запуске MCS-Finger пользователь получает возможность увидеть записи в базе данных пользователей и некоторые отпечатки их пальцев (рис. 33.8).

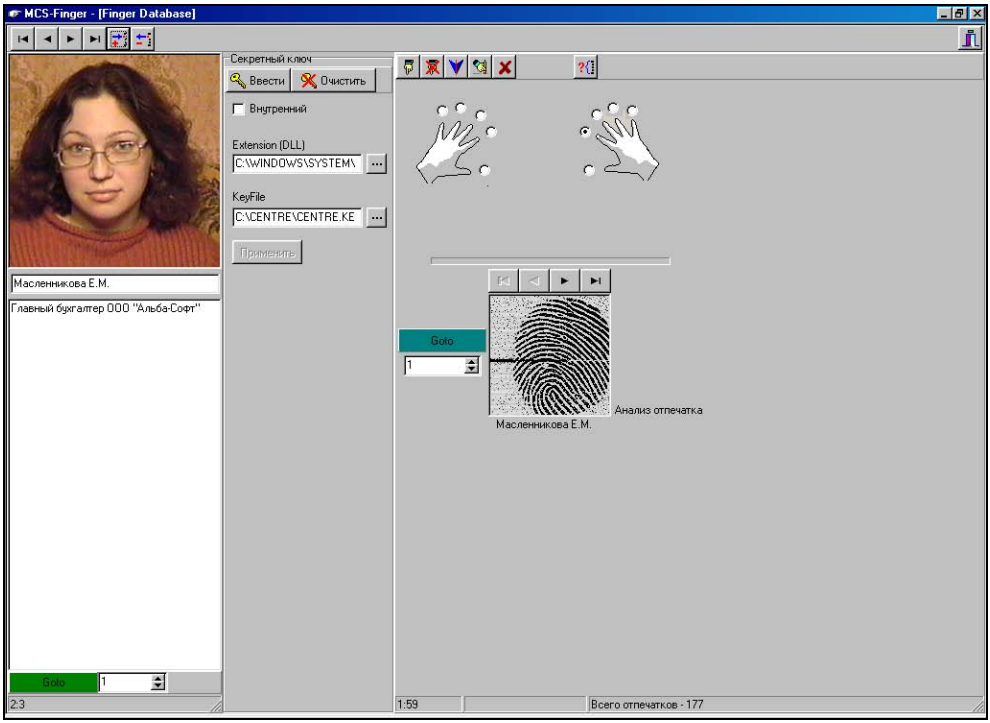

Рис. 33.8. База данных пользователей и отпечатки их пальцев в программе MCS-Finger

Отпечатки пальцев пользователей имеется возможность добавлять как с помощью устройства типа FingerID, так и из BMP-файла. Здесь же можно определить секретный ключ, который в FCS-файле будет защищать отпечаток пальца. Это может быть внутренний секретный ключ, т. е. отпечаток пальца будет защищать случайный секретный ключ, получающийся с помощью компьютерной игры. Необходимо установить длину такого ключа и нажать кнопку **Ввести.** Появляется уже знакомое по *главе 8* окошко с прыгающим ключиком, который надо ловить мышью (рис. 33.9).

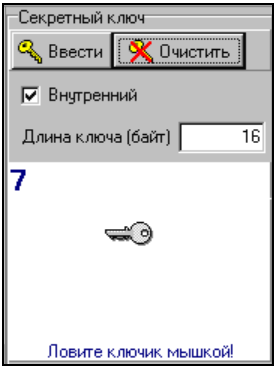

Рис. 33.9. Выработка внутреннего секретного ключа для FCS-файла

В каких случаях целесообразно использовать внутренний секретный ключ? Дело в том, что уникальность отпечатка пальца человека можно приспособить для простых житейских ситуаций. Пользователи Internet знают, как много различных паролей может потребоваться в сети. При регистрации на различных сайтах, как правило, требуется задать свой уникальный пароль. Запомнить их все обычно очень сложно, поэтому пользователи часто используют совсем тривиальные пароли, вроде "123456", причем одни и те же для всех сайтов. Нет нужды рассуждать о стойкости такого подхода.

С использованием отпечатка пальца такая проблема легко решается. С помощью внутреннего секретного ключа, вычисляемого по отпечатку пальца, создается система генерации случайных паролей. На жестком диске компьютера при этом ничего секретного не хранится: только FCS-файл и список сайтов, на которые требуются пароли. Более подробно об этой возможности использования отпечатка пальца мы поговорим в *главе 35.*

Несколько другая процедура ввода внешнего секретного ключа для FCS-файла. Для нее предварительно требуется создать специальную динамическую библиотеку, позволяющую установить "взаимопонимание" между внешним секретным ключом и системой использования отпечатков пальцев. Ведь внешний секретный ключ — это, в конечном итоге, последовательность байт, которая затем непосредственно используется в криптографическом приложении. На примере описанных в *главе 8* секретных ключей видно,

что такая последовательность может получаться путем перемешивания цифровой части ключа с паролем — см. листинг 8.3. И ключ, вычисляемый по отпечатку пальца, должен защищать именно уже готовый к применению секретный ключ, а не его "полуфабрикат", для которого еще требуется вводить пароль. Поэтому первой задачей, которую должна выполнить эта специальная динамическая библиотека, является получение из файла с секретным ключом и, возможно, пароля, который при этом запрашивается, некоторой последовательности байт, готовой к использованию в криптографическом приложении в качестве секретного ключа. Такая процедура называется `KeyForFinger`, и вот пример ее реализации для ключей, описанных в *главе 8*.

Листинг 33.1. Процедура `KeyForFinger`

```
// Формирование ключа для FCS-файла
// Параметры на входе:
// KeyFileName — имя файла с ключом
// Параметры на выходе:
// FKey — ключ для FCS-файла
// Код возврата:
// Если успех, то SKeyLen — длина ключа для FCS-файла
// Если неуспех, то 0

extern "C" __declspec(dllexport) __stdcall int KeyForFinger(
char *KeyFileName,unsigned char *FKey)
{
  int i,j;

  unsigned char key_d[256];
  unsigned char cont[8];
  unsigned char x[8],x1[8];
  unsigned char pi[256];
  unsigned char l[80],l1[40];
  unsigned char pi_cnt[256],pi_gam[256];
  unsigned char empt;

 FILE *fl;
  char keyfile[256];
  char password[128];
  char password1[128];
```

```
char attempt=0;
int result;

char buff[256];
char buff2[256];
char key_label[16];
char win_dos=0;
unsigned short sign_key[40];
unsigned short crypt_key[40];

// В начале процедуры из файла с базовым секретным ключом
// вводятся все необходимые параметры:
//    цифровая часть,
//    проверочные комбинации,
//    метка ключа
if((fl=fopen(KeyFileName,"rb"))==NULL)
{
   if(strrchr(KeyFileName,'\\')==NULL)
     sprintf(buff,"Ошибка при открытии файла с ключом %s",KeyFileName);
   else
     sprintf(buff,
   "Ошибка при открытии файла с ключом %s",strrchr\
   (KeyFileName,'\\')+1);
   MessageBox(0,buff,"ОШИБКА",MB_ICONERROR | MB_OK);
   return(0);
}
memset(keyfile,0,256);
fread(keyfile,142,1,fl);
fclose(fl);
memcpy(key_d,keyfile,110);
memcpy(cont,keyfile+110,8);

srand(7432);
memset(pi_cnt,0,256);
for(i=0;i<256;i++)
{
  empt=(unsigned char)rand();
  while(pi_cnt[empt]==1)
    empt++;
```

```
  pi[i]=empt;pi_cnt[empt]=1;
}

for(i=0;i<8;i++)
  x[i]=(unsigned char)rand();
for(i=0;i<32;i++)
{
  CryptBlock(x,pi,key_d,100);
  CryptBlock(x,pi,key_d,100);
  for(j=0;j<8;j++)
    pi_gam[8*i+j]=x[j];
}

memset(pi_cnt,0,256);
for(i=0;i<256;i++)
{
  empt=pi_gam[i];
  while(pi_cnt[empt]==1)
    empt++;
  pi[i]=empt;
  pi_cnt[empt]=1;
}

for(i=0;i<8;i++)
  x[i]=(unsigned char)rand();
for(i=0;i<5;i++)
{
  CryptBlock(x,pi,key_d,100);
  CryptBlock(x,pi,key_d,100);
  for(j=0;j<8;j++)
    l[8*i+j]=x[j];
}

CryptBlock(x,pi,l,40);
// Первая проверочная комбинация
if(memcmp(x,cont,8)!=0)
  return(0);
// Ввод пароля
memcpy(cont,keyfile+118,8);
```

```
  memset(password,0,128);
  memset(password1,0,128);
  memcpy(x1,x,8);
  memcpy(l1,l,40);

input_password:
  // Для совместимости DOS- и Windows-версий
  win_dos=0;
  // Окно для ввода пароля
  CentrePsw = new TCentrePsw(Application);
  CentrePsw->Edit1->Text = password1;
  CentrePsw->Edit1->PasswordChar = '*';
  result = CentrePsw->ShowModal();
  memset(password,0,128);
  memset(password1,0,128);
  strcpy(password1,CentrePsw->Edit1->Text.c_str());
  delete CentrePsw;
  // Если был отказ от ввода пароля
  if(result!=IDOK)
    return(0);

w_d:
  // Манипуляции с DOS- и Windows-форматами
  if(win_dos==0)
    strcpy(password,password1);
  else
    CharToOem(password1,password);

  i=0;
  while(password[i]!=0)
  {
    l[i%40]=(unsigned char)(l[i%40]+password[i]);
    i++;
  }
  CryptBlock(x,pi,l,40);

  // Правильность ввода пароля проверяется по второй проверочной
  // комбинации
  if(memcmp(x,cont,8)!=0)
```

```
{
  memcpy(x,x1,8);
  memcpy(l,l1,40);
  if(win_dos==0)
  {
    win_dos=1;
    goto w_d;
  }

 MessageBox(0,"Ошибка в пароле","ОШИБКА",MB_ICONERROR | MB_OK);
 attempt++;
 // Традиционные три попытки для ввода пароля
 if(attempt>=3)
   return(0);
 else
   goto input_password;
}
// Вычисление метки ключа
memcpy(key_label,keyfile+126,8);
for(i=0;i<8;i++)
  key_label[i]=(unsigned char)(key_label[i]+
  (unsigned char)rand());
srand(7643);

for(i=0;i<8;i++)
  x[i]=(unsigned char)rand();
for(i=0;i<32;i++)
{
  CryptBlock(x,pi,l,40);
  CryptBlock(x,pi,l,40);
  CryptBlock(x,pi,l,40);
  CryptBlock(x,pi,l,40);
  CryptBlock(x,pi,l,40);
  for(j=0;j<8;j++)
    pi_gam[8*i+j]=x[j];
}
memset(pi_cnt,0,256);
for(i=0;i<256;i++)
{
```

```
    empt=pi_gam[i];
    while(pi_cnt[empt]==1)
      empt++;
    pi[i]=empt;pi_cnt[empt]=1;
  }

  CryptBlock(x,pi,l,40);
  // Третья проверочная комбинация
  if(memcmp(x,keyfile+134,8)!=0)
    return(0);

  // В FCS-файле будут в зашифрованном виде храниться:
  //    метка ключа,
  //    подстановка,
  //    входное слово
  memcpy(FKey,key_label,8);
  memcpy(FKey+8,pi,256);
  memcpy(FKey+8+256,l,40);

  // Секретный ключ для подписи
  srand(7063);
  for(i=0;i<8;i++)
    x[i]=(unsigned char)rand();
  for(i=0;i<8;i++)
  {
    for(j=0;j<5;j++)
      CryptBlock(x,pi,l,40);
    for(j=0;j<4;j++)
      sign_key[4*i+j]=(unsigned short)(x[2*j]+256*x[2*j+1]);
  }

  // Секретный ключ для шифрования с применением открытых ключей
  srand(7561);
  for(i=0;i<8;i++)
    x[i]=(unsigned char)rand();
  for(i=0;i<8;i++)
  {
    for(j=0;j<5;j++)
      CryptBlock(x,pi,l,40);
```

```
  for(j=0;j<4;j++)
    crypt_key[4*i+j]=(unsigned short)(x[2*j]+256*x[2*j+1]);
}

// Проверочная комбинация ключа для подписи
srand(7411);
memset(pi_cnt,0,256);
for(i=0;i<256;i++)
{
  empt=(unsigned char)rand();
  while(pi_cnt[empt]==1)
    empt++;
  pi[i]=empt;
  pi_cnt[empt]=1;
}

memset(l,0,80);
memcpy(l,(char *)sign_key,64);
memcpy((char *)sign_key+64,key_label,8);
memcpy(l+64,(char *)sign_key+64,8);
memcpy(buff,l+64,8);

for(i=0;i<8;i++)
  x[i]=(unsigned char)rand();
CryptBlock(x,pi,l,72);
CryptBlock(x,pi,l,72);

memcpy((char *)sign_key+72,x,8);

// Проверочная комбинация ключа для шифрования
srand(7522);
memset(pi_cnt,0,256);
for(i=0;i<256;i++)
{
  empt=(unsigned char)rand();
  while(pi_cnt[empt]==1)
    empt++;
  pi[i]=empt;
  pi_cnt[empt]=1;
```

```
  }

  memcpy(l,(char *)crypt_key,64);
  memcpy((char *)crypt_key+64,key_label,8);
  memcpy(l+64,key_label,8);

  for(i=0;i<8;i++)
    x[i]=(unsigned char)rand();
  CryptBlock(x,pi,l,72);
  CryptBlock(x,pi,l,72);
  memcpy((char *)crypt_key+72,x,8);

  // Проверочные комбинации секретных ключей для подписи
  // и шифрования дописываются в FCS-файл
  memcpy(FKey+8+256+40,(unsigned char *)sign_key,80);
  memcpy(FKey+8+256+40+80,(unsigned char *)crypt_key,80);

  // Обнуляем лишнее
  memset(password,0,128);
  memset(keyfile,0,256);
  memset(key_d,0,256);
  memset(pi,0,256);
  memset(pi_cnt,0,256);
  memset(pi_gam,0,256);
  memset(l,0,80);
  memset(x,0,8);

  // Возврат — длина последовательности для FCS-файла
  return(8+256+40+80+80);
}
```

Из листинга 33.1 видно, что последовательность, включаемая в FCS-файл, может, помимо самих секретных ключей (их здесь несколько), содержать также метку ключа, различные проверочные комбинации и т. п. Необходимость тех или иных параметров, включаемых в FCS-файл, определяется из особенностей реализации секретных ключей в конкретной криптографической системе: в FCS-файле должны присутствовать все те параметры, которые получаются при обработке файла с секретным ключом и используются при дальнейших операциях.

Но помимо процедуры `KeyForFinger`, в динамической библиотеке также должна быть процедура `KeyFromFinger`, которая обеспечивает возможность расшифрования и ввода хранящейся в FCS-файле последовательности с помощью отпечатка пальца. Здесь необходимо сделать небольшое отступление и поговорить о том, как можно хранить FCS-файлы на компьютере. Поскольку каждый FCS-файл строго привязан только к одному пользователю, то, например, в реестре Windows можно задать список пользователей и соответствующие каждому пользователю FCS-файлы, расположенные где угодно: на локальном жестком диске, сети, дискете и т. п. Рассовать их по разным щелям, чтобы задать дополнительную головную боль для потенциального злоумышленника. При необходимости использования FCS-файла удобно вызывать его не по имени файла, а по имени его пользователя, прописанному в реестре Windows. Такую систему мы будем и дальше использовать, в частности в программе MCS-Password для генерации случайных паролей по отпечатку пальца. О ней подробно речь пойдет в *главах 34* и *35*.

В приводимом ниже листинге 33.2 процедуры `KeyFromFinger` именно такой способ обращения к FCS-файлу и используется.

Листинг 33.2. Процедура `KeyFromFinger`

```
// Ввод секретного ключа из FCS-файла
// Параметры на входе:
// User — имя пользователя
// Параметры на выходе:
// SKey — секретный ключ
// Код возврата:
// Если успех, то SKeyLen — длина ключа из FCS-файла
// Если неуспех, то 0

extern "C" __declspec(dllexport) __stdcall int KeyFromFinger(
char *User,unsigned char *FKey)
{
  char buff[128];
  SKeyLen=0;
  // Вычисление начальных параметров для CryptBlock
  CalculatePI();
  memset(ControlBlock,0,8);
  memset(key,0,14);
  NAtt=0;
  CurIsp=0;
  IsImage = false;
```

```
  strcpy(UserID,User);
  // Окно для ввода отпечатка пальца
  // и получения в глобальном массиве SKey
  // требуемой последовательности
  Finger = new TFinger(NULL);
  sprintf(buff,"Пользователь %s",User);
  Finger->Label2->Caption = buff;
  Finger->Timer1->Enabled = true;
  if(Finger->ShowModal()!=IDOK)
  {
    // Отказ от ввода
    delete Finger;
    return(0);
  }
  memcpy(FKey,SKey,SKeyLen);
  memset(SKey,0,SKeyLen);
  free(SKey);
  return(SKeyLen);
}
```

Таким образом, процедура ввода секретного ключа с помощью отпечатка пальца может быть легко встроена практически в любую криптографическую систему. Для этого разработчику необходимо подготовить динамическую библиотеку, содержащую процедуру `KeyForFinger`. Входные-выходные параметры должны быть такими же, что и в приведенном ранее листинге 33.1, а содержание самой процедуры зависит от типа криптографической системы и параметров секретного ключа, которые в ней используются. После этого вместо обычной процедуры ввода секретного ключа используется `KeyFromFinger`, находящаяся в специализированной динамической библиотеке MCS_FL.DLL.

Вернемся теперь к программе MCS-Finger и процедуре ввода внешнего секретного ключа. Для этого необходимо задать имя файла с секретным ключом и динамическую библиотеку, содержащую процедуру `KeyForFinger` (рис. 33.10).

С помощью кнопки **Ввести** осуществляется запуск процедуры `KeyForFinger` и осуществляется подготовка последовательности, защищаемой в FCS-файле отпечатком пальца.

Дальнейшая процедура построения FCS-файла во многом уже описана выше. С помощью функции "Анализ отпечатка" оператор получает изображение отпечатка пальца выбранного пользователя и имеет возможность

в интерактивном режиме вырезать на нем несколько (до 8) фрагментов, которые затем войдут в FCS-файл. Вырезанные фрагменты отображаются на экране, одни из них можно брать из одного отпечатка, другие — из другого, задавать им различные размеры (в том числе и прямоугольные), осуществлять тестирование получившегося набора фрагментов на различных отпечатках из базы данных и непосредственно получаемых с помощью устройства FingerID.

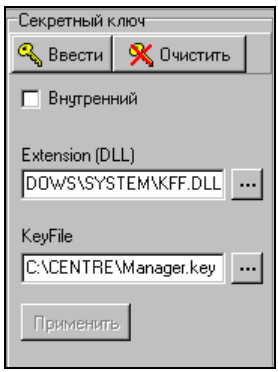

Рис. 33.10. Внешний секретный ключ для FCS-файла

Помимо программы MCS-Finger, я также написал программу BuildFCS, позволяющую строить FCS-файлы в автоматическом режиме. От пользователя там требуется немного: определить требуемый уровень стойкости, внешний или внутренний секретный ключ, немного половить маленький ключик мышью для создания "честного" генератора случайных чисел, по которому будут выбираться фрагменты отпечатка. Ну и, конечно, приложить палец к считывателю. Все остальное автоматизировано, и нет никаких баз данных.

Глава 34

Криптосервер

В предыдущих главах мы уже неоднократно употребляли это слово, так что общее представление о том, что такое криптосервер, читатель уже имеет. Настало время углубиться в детали.

Итак, криптосервер — это некоторая специально выделенная часть сервера, на которой хранятся секретные ключи к той или иной криптографической системе, и программные средства для работы с этой частью сервера. На сетевые средства защиты особо не полагаемся, но без особой нужды и не пренебрегаем ими. Уровень подготовки и опыт потенциального злоумышленника может быть таким, что этих средств окажется достаточно. Но основную защиту предполагаемого криптосервера должна обеспечивать трудоемкая задача перебора вариантов некоторых значений, с которой хакер практически любой квалификации справиться не в состоянии. Необходимо иметь четкое представление о том, что именно перебирается, каковы алгоритмы перебора, оценки его трудоемкости и надежности. Только после этого можно говорить о гарантированной стойкости.

Можно ли полагаться на существующие способы ограничения перебора, например, на программные способы блокировки системы после троекратного неправильного ввода пароля? Очень сомнительно, что таким способом можно обеспечить гарантированную защиту. Квалифицированный хакер-фанатик может работать на более низком программном уровне, чем предлагаемая программная система ограничения перебора. Он может иметь дело напрямую с портами ввода-вывода, программировать различные контроллеры, перехватывать сетевые сообщения, провоцировать систему ложными сообщениями и т. п. И вообще, писать свои программы на ассемблере, а программные коды для него — язык родной. Подтверждением тому служат многочисленные сообщения о взломах хакерами различных систем защиты, в которых пренебрегли вопросами криптографии. Гарантированная криптографическая защита всегда создает некоторые неудобства законным пользователям, а некоторые производители ставят удобство пользователя превыше всего. И, как правило, становятся при этом легкой добычей для хакеров.

Следовательно, секретные ключи, хранящиеся на криптосервере, лучше всего шифровать. Но тогда нужен некоторый секретный ключ для шифрования этих ключей. А где его хранить? Можно, конечно, предлагать системы, где есть некоторый сетевой администратор, обладающий своим секретным ключом, с помощью которого шифруется вся информация на криптосервере. Но, во-первых, такой человек будет иметь доступ ко всем секретным ключам, а, во-вторых, на какой-то момент времени, когда пользователям необходимо использовать свои секретные ключи, он должен будет их расшифровать. Пусть такой момент времени и мал, но где гарантия того, что потенциальный злоумышленник, подготовившись заранее, не сможет его использовать?

В общем, как ни крути, а система, основанная на отпечатках пальцев пользователей, наиболее удачно сочетается с идеей криптосервера. Сетевой администратор не имеет доступа к секретным ключам пользователей. Помимо доступа к криптосерверу, потенциальный злоумышленник столкнется с весьма трудоемкой задачей перебора расположения фрагментов, о которой шла речь в прошлой главе (см. рис. 33.5). Ограничений на время ввода секретного ключа нет, пользователь имеет возможность с помощью своего отпечатка ввести его в любой момент. Да и защищать с помощью отпечатка пальца можно не только секретный ключ, но и некоторую другую конфиденциальную информацию — в отличие от устройств типа Touch Memory или Smart Card на сервере места для этого достаточно.

Таким образом, криптосервером могут быть места обитания FCS-файлов. Плюс некоторый интерфейс для распознавания пользователей, о котором мы также немного упоминали в прошлой главе. Давайте теперь поговорим о нем подробнее.

Каждый FCS-файл строго привязан только к одному пользователю. Пользователь, в свою очередь, может иметь несколько FCS-файлов. Например, различные FCS-файлы для различных пальцев или различные FCS-файлы для различных секретных ключей, которые они защищают. Возникают пары "Идентификатор пользователя — Имя FCS-файла". Особо секретной информации в такой паре нет, разве что подсказка, где искать FCS-файл пользователя. Но будем считать, что и без нее с такой задачей хакер в состоянии справиться.

К паре "Идентификатор пользователя — Имя FCS-файла" можно добавить третий элемент — задачу, для которой используется FCS-файл. Такой задачей может быть, например, генерация пароля для определенного сайта в Internet. Но третий элемент — это необязательный параметр, если пользователь имеет на криптосервере только один FCS-файл, то третий параметр оказывается ненужным.

Дело вкуса разработчика криптосервера, куда помещать эти пары. Но для простоты можно предложить помещать их в реестр Windows. Для обращения

к криптосерверу используется идентификатор пользователя, в ответ пользователю предлагается ввести отпечаток своего пальца, по которому происходит попытка распознавания соответствующего FCS-файла. Ниже на рис. 34.1 приведен пример использования криптосервера для ввода секретного ключа.

Рис. 34.1. Пример использования криптосервера

Больше никаких дискет или паролей от пользователя не требуется! Например, в системе Криптоцентр для Microsoft Outlook секретный ключ, загружаемый с помощью криптосервера, можно сделать ключом по умолчанию, после чего от пользователя при запуске Microsoft Outlook будет требоваться только правильный ввод отпечатка пальца.

Реализация процедуры ввода отпечатка пальца и обращения к криптосерверу требует наличия в программе двух таймеров. Первый таймер отвечает за организацию попыток считывания отпечатка пальца со считывателя в оперативную память, а второй — за прекращение таких попыток, когда наилучший (визуально оцененный пользователем) вариант отпечатка пальца в памяти уже есть, и обращение к криптосерверу. Далее в листинге 34.1 приводится пример реализации процедур обработки этих таймеров. Первый таймер дает пользователю 200 попыток ввода отпечатка пальца, подсчетом которых занимается глобальная переменная `NAtt`. Наличие отпечатка пальца в памяти проверяется по глобальной переменной `IsImage`. После того как пользователь оторвал палец от считывателя, первый таймер должен сработать еще 8 раз (глобальная переменная `CurIsp`) с тем, чтобы убедиться, что пользователь сделал это сознательно, а не случайно. После этого первый таймер отключается, а включается второй, с помощью которого происходит обращение к криптосерверу.

Листинг 34.1. Таймеры для считывания отпечатка пальца и обращения к криптосерверу

```
// Второй таймер в этот момент отключен
void __fastcall TFinger::Timer1Timer(TObject *Sender)
{
  FL_CONTEXT_HANDLE contextHnd;
  FL_RESULT_CODE     iFingerLocResultCode;
  unsigned char *pImage;
  char MyIm[128][128];
  TMemoryStream *MS;
  int i;
  unsigned long i_l;
  char buff[256];
  LPBITMAPINFOHEADER m_pBitmap1;
  char Hdr[20000];

  // Сначала необходимо отключить сам этот таймер
  Timer1->Enabled = false;
  // Инициализация устройства FingerID
  contextHnd = FingerLocInit(NULL);
  // Чтобы программа не зависла при отсутствии устройства,
  // предусмотрена обработка отрицательного результата инициализации
  if(contextHnd==NULL)
  {
    detect:
    Timer1->Enabled = true;
    // Если образ отпечатка пальца уже сосчитан
    if(IsImage==true)
    {
      CurIsp++;
      // Надо убедиться, что прекращение подачи
      // отпечатков пальцев произошло сознательно, а не случайно
      if(CurIsp>8) // Если сознательно
      {
        Timer1->Enabled = false;
        Timer2->Enabled = true;
      }
      // Возврат или на Timer1, или на Timer2
```

```
    return;
  }
  NAtt++;
  // Если число попыток исчерпано
  if(NAtt>200)
    ModalResult = mrCancel;
  // Число попыток не исчерпано — возврат на Timer1
  return;
}
// Пользователь должен видеть, что устройство работает.
// Для этого предназначен индикатор
ProgressBar1->Position = 20;

// Открытие потока
iFingerLocResultCode = FingerLocOpenStream(contextHnd, FALSE);
// Если не удалось открыть поток
if (iFingerLocResultCode != FL_OK)
{
  // Закрытие устройства
  FingerLocClose(contextHnd);
  // Дальше — такая же обработка, что и в предыдущем случае
  goto detect;
}
ProgressBar1->Position = 40;

// Получение текущего изображения отпечатка пальца
pImage = FingerLocGetCurrentImage(contextHnd, IMAGE_250_DPI, FALSE);
// Если не удалось получить изображение отпечатка.
// Во всех предыдущих случаях отрицательный результат
// мог быть в том случае, когда устройство отсутствует или неисправно.
// В этом случае отрицательный результат получается тогда,
// когда пользователь не прислонил палец к исправному устройству
if(pImage==NULL)
{
  FingerLocCloseStream(contextHnd);
  FingerLocClose(contextHnd);
  goto detect;
}
```

```
ProgressBar1->Position = 60;

// Изображение считано в память.
// Осталось преобразовать его в BMP-вид и закрыть
// процедуры работы с устройством
m_pBitmap1 = FingerLocAllocOSBitmap(128 + 16);
FingerLocConvertRawImageToOSBmp(m_pBitmap1,pImage,128);
FingerLocReleaseImage(contextHnd, pImage);
FingerLocCloseStream(contextHnd);
FingerLocClose(contextHnd);

// Формирование Memory Stream с изображением отпечатка пальца
iHeaderSize = sizeof(BITMAPINFOHEADER) + 256 * sizeof(RGBQUAD);

for(i=0;i<128;i++)
  memcpy(MyIm[i],(unsigned char *)\
  m_pBitmap1+iHeaderSize+144*8+144*i+8,128);
(char *)m_pBitmap = Hdr;
memcpy(m_pBitmap,m_pBitmap1,iHeaderSize);
if(m_pBitmap1 != NULL)
  FingerLocDeleteOSBitmap(m_pBitmap1);
ProgressBar1->Position = 80;

MS = new TMemoryStream();
MS->Clear();
m_pBitmap->biSizeImage=128*128;
m_pBitmap->biWidth=128;
m_pBitmap->biHeight=128;

MS->Write("BM",2);
i_l=iHeaderSize+m_pBitmap->biSizeImage+14;
MS->Write(&i_l,6);
memset(buff,0,2);
MS->Write(buff,2);
i=iHeaderSize+14;
MS->Write(&i,4);
MS->Write((char *)m_pBitmap,iHeaderSize);
MS->Write(MyIm,m_pBitmap->biSizeImage);
MS->Seek(0,soFromBeginning);
```

```
  Image1->Picture->Bitmap->LoadFromStream(MS);
  delete MS;
  // Изображение введено в память.
  // Формируем соответствующие переменные
  NAtt=0;
  CurIsp=0;
  IsImage = true;
  ProgressBar1->Position = 100;
  // Для обработки должно быть использовано
  // только самое последнее изображение.
  // Надо проверить, что полученное изображение - самое последнее.
  Timer1->Enabled = true;
}

// Второй таймер — обращение к криптосерверу
void __fastcall TFinger::Timer2Timer(TObject *Sender)
{
  TRegistry *Reg;
  char buff[512];
  char buff1[512];
  char FCSName[512];
  unsigned short i,j,NFragm=0;
  unsigned short LRec=0;
  char *NBuff;

  // Сразу же отключаем второй таймер
  Timer2->Enabled = false;
  // Пользователя надо проинформировать о том, что введенное
  // изображение отпечатка пальца поступило на обработку.
  // Картинка с отпечатком пальца заменяется на стандартную
  // картинку, а надпись с просьбой прислонить
  // палец к считывателю пропадает
  Image2->Visible = true;
  Image1->Visible = false;
  Label1->Visible = false;
  Update();
  // Из реестра Windows считываем имя FCS-файла
  memset(FCSName,0,512);
  Reg = new TRegistry();
```

```
  Reg->RootKey = HKEY_CURRENT_USER;
  // UserID — идентификатор пользователя.
  // Он заранее задан
  sprintf(buff,"\\SOFTWARE\\AlbaSoft\\MCS-Finger\\Keys\\%s",UserID);
  if(Reg->OpenKey(buff,false)==true)
    strcpy(FCSName,Reg->ReadString("FCS-файл").c_str());
  else
  {
    delete Reg;
error:
    MessageBox(0,"Неизвестный пользователь","MCS_FL",
    MB_ICONERROR | MB_OK);
    ModalResult = mrCancel;
    return;
  }
  delete Reg;
  if(FCSName==NULL)
    goto error;

  // Поток MS0 — изображение пальца. Глобальная переменная
  MS0 = new TMemoryStream();
  MS0->Clear();
  MS0->Seek(0,soFromBeginning);
  Image1->Picture->Bitmap->SaveToStream(MS0);
  MS0->Seek(0,soFromBeginning);
  // Создаем наилучшее черно-белое изображение отпечатка пальца
  MakeImage0();
  MakeFineImage();
  MS0->Seek(0,soFromBeginning);
  Image1->Picture->Bitmap->LoadFromStream(MS0);
  MS0->Seek(0,soFromBeginning);
  // Поток MSKEY — из FCS-файла
  if(MSKEY!=NULL)
    delete MSKEY;
  MSKEY = new TMemoryStream();
  MSKEY->Clear();
  // MSKEY — поток, содержащий FCS-файл
  MSKEY->LoadFromFile(FCSName);
  MSKEY->Seek(0,soFromBeginning);
  MSKEY->Read(&NRec,2); // Число наборов фрагментов
```

```
MSKEY->Read(&SKeyLen,2); // Длина секретного ключа
srand(4320);
i=0;
// В FCS-файле может быть несколько наборов фрагментов,
// снятых с разных вариантов отпечатка пальца
while(i<NRec)
{
  // Очищаем фрагменты.
  // Эти картинки на экране невидимы
  Image3->Picture=NULL;
  Image4->Picture=NULL;
  Image5->Picture=NULL;
  Image6->Picture=NULL;
  Image7->Picture=NULL;
  Image8->Picture=NULL;
  Image9->Picture=NULL;
  Image10->Picture=NULL;
  MSKEY->Read(&LRec,2);
  if(LRec<=8+1+SKeyLen+2)
  {
    MessageBox(0,"Ошибка в FCS-файле!","MCS-Password",
    MB_ICONERROR | MB_OK);
    return;
  }
  LRec-=(8+1+SKeyLen);
  // Проверочная комбинация
  MSKEY->Read(ControlBlock,8);

 // Точность
 MSKEY->Read(&Tch,1);
 SKey=(unsigned char *)malloc(SKeyLen+8);
 // Зашифрованный секретный ключ
 MSKEY->Read(SKey,SKeyLen);
 NFragm=0;
 while((short)LRec>0)
 {
   // Заполняем очередную картинку из FCS-файла
   LRec=(unsigned short)FillFragm(LRec,NFragm);
   NFragm++;
 }
```

```
    // Если удалось расшифровать секретный ключ
    if(CheckSecretKey()==true)
      break;
    // Если не удалось расшифровать секретный ключ по отпечатку,
    // то переход к следующему набору фрагментов
    i++;
    free(SKey);
  }//while(i<NRec) — цикл по всем фрагментам
  Image1->Picture=NULL;
  Update();
  // Если ни на одном наборе фрагментов
  // расшифровать секретный ключ по отпечатку не удалось
  if(i==NRec)
  {
    SKeyLen=0;
    NAtt=0;
    CurIsp=0;
    IsImage = false;
    Image2->Visible = false;
    Update();
    i=MessageBox(0,
    "Ключ не определен. Хотите попробовать ввести еще раз?",
    "MCS-Finger",MB_ICONERROR | MB_YESNO);
    memset(key,0,14);
    if(i==IDNO)ModalResult = mrCancel;
      if(i==IDYES)
      {
        // Еще одна попытка ввода
        Image1->Visible = true;
        Label1->Visible = true;
        Timer1->Enabled = true;
      }
    return;
  }
  // Если секретный ключ по отпечатку вычислен успешно, то на нем
  // расшифровываем секретный ключ для криптографической системы
  DecryptKey(SKey);
  ModalResult = mrOk;
}
```

Пользователь умывает руки.

Глава 35

Генератор случайных паролей по отпечатку пальца

Как и всякий другой компьютерный фанат, я очень люблю Internet. Поисковые серверы — еще лет 15—20 назад никто не мог и подумать, что такое станет возможным. Задаешь интересующую тебя тему — в ответ огромная куча информации. Не нужно ездить в библиотеки, копаться в реферативных журналах, букинистических магазинах. Если хочешь — скачивай все, что нужно себе на компьютер, собирай собственную базу данных. Хочешь — читай газеты в Internet, узнавай о всех событиях в мире. Хочешь — изучай народный юмор, ничем не ограниченный.

А можно оставлять и свои следы во всемирной паутине. Зарегистрировать свой сайт, свои почтовые ящики, зарегистрироваться на чьем-то сайте в качестве партнера, покупателя, дистрибьютера, члена клуба по интересам, искателя компьютерной халявы, игрока, пирата, читателя, писателя, выполнять какую-то работу, в общем, при желании по Internet скоро можно будет забеременеть. Нужны только Login и Password. И в больших количествах, для каждой регистрации — свои.

Ну, с Login все понятно — ID пользователя. Обзывай себя по имени, по фамилии, по детскому прозвищу, по специальности, по увлечениям, в общем, по-всякому. Делай один и тот же Login хоть для всех своих регистраций — ни на что не влияет. А вот Password — это личный конфиденциальный параметр. Неприятно, если твои электронные письма будет читать кто-то другой. Или на твоем сайте появится неприличная надпись. Нужна хоть какая-то защита, и для такой защиты используется Password.

Интуитивно ясно, что защита в виде Password вряд ли может претендовать на стопроцентную гарантированную защиту. В большинстве случаев, если вы забыли пароль, то владельцы сайта любезно вышлют его на адрес вашей электронной почты в открытом и ничем не защищенном письме. Но и совсем пренебрегать паролем, выбирать совсем тривиальные варианты типа "12345" тоже как-то некомфортно. Все же он защищает личную информацию, не предназначенную для всеобщего обозрения.

И начинает пользователь мучиться, придумывая себе Password. Чтобы, с одной стороны, не забыть, не напрягаться при вводе, а с другой — чтобы было свое, оригинальное. Да еще хочется, чтобы были разные Password для различных регистраций. Сам многократно бывал в подобных ситуациях.

Давайте попробуем приспособить для генерации случайных паролей отпечаток пальца человека. Чтобы вообще не помнить никаких паролей, регистрироваться сколько угодно раз и где угодно и всякий раз с новым паролем, который нигде не будет храниться. А там, глядишь, такая система может пригодиться не только в Internet, но и вообще в любой другой ситуации, где пользователю требуется индивидуальный пароль.

Постараемся для этих целей приспособить FCS-файлы. Что в них запрятано? Некоторая секретная последовательность, зашифрованная с помощью ключа, вычисляемого по взаимному расположению фрагментов отпечатка пальца. Для выработки случайных паролей неважно, каким способом была получена эта последовательность — в качестве внутреннего или внешнего секретного ключа.

Каждый вырабатываемый случайный пароль предназначен для вполне определенных целей. Например, один — для регистрации на сервере Yandex, другой — на Rambler. Назначение пароля где-нибудь запишем (например, в реестре Windows) и для выработки пароля будем использовать расшифрованную секретную последовательность из FCS-файла, назначение пароля и, конечно же, функцию `CryptBlock`. При такой выработке для одного и того же назначения пароля всякий раз будет получаться один и тот же пароль. Через CLIPBOARD используем его по назначению. Вот основные идеи. Переходим к их реализации в программе, которую я назвал *MCS-Password*.

Эта программа загружается резидентно в оперативную память и ожидает вызова со стороны пользователя. В правом нижнем углу экрана отображается ее иконка, которая, кроме всего прочего, отображает текущее состояние программы MCS-Password. Таких состояний три (рис. 35.1—35.3).

Рис. 35.1. Отпечаток пальца не введен

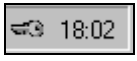

Рис. 35.2. Отпечаток пальца введен, но пароль в CLIPBOARD не загружен

Рис. 35.3. Отпечаток пальца введен и пароль загружен в CLIPBOARD

Вызов программы осуществляется щелчком правой клавиши мыши по иконке.

Предполагается, что программой MCS-Password, установленной на один компьютер, будут пользоваться несколько пользователей, каждый из которых имеет свой индивидуальный FCS-файл. Первым делом регистрируем пользователей и их FCS-файлы (рис. 35.4).

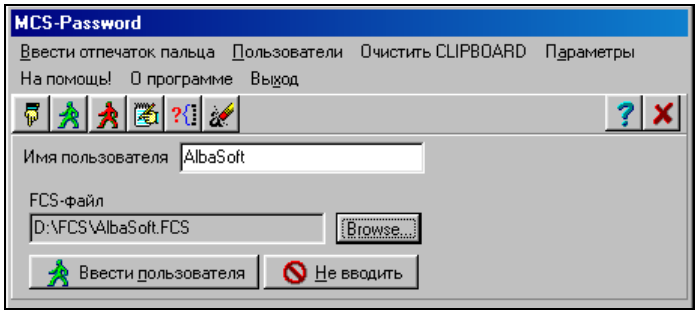

Рис. 35.4. Регистрация пользователя и его FCS-файла в системе MCS-Password

Пользователей у системы MCS-Password может быть несколько, однако при запуске программы автоматически запрашивается отпечаток пальца пользователя, установленного как **Пользователь по умолчанию** в разделе **Параметры** (рис. 35.5).

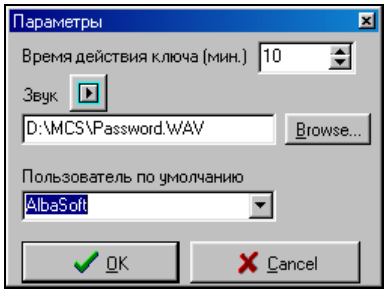

Рис. 35.5. Раздел **Параметры** системы MCS-Password

Кроме того, в разделе **Параметры** можно установить время, в течение которого будет действовать введенный отпечаток пальца. По истечении этого времени отпечаток пальца потребуется вводить заново. При значении 0 время не ограничено.

Успешную загрузку отпечатка пальца можно выделять определенным звуковым сигналом. Файл с таким сигналом также можно указать в разделе **Параметры**.

При вызове программы MCS-Password в случае, когда отпечаток пальца не введен (см. рис. 35.1), на экран автоматически выдается запрос на ввод отпечатка пальца (рис. 35.6).

Механизм осуществления запроса здесь точно такой же, что и в криптосервере — см. листинг 34.1 из прошлой главы. На рис. 35.6 хорошо видны помехи, вызываемые сбоями при работе считывателя. Источниками таких помех могут быть, например, небольшие заряды статического электричества, часто скапливающиеся на пальцах пользователя. Но благодаря тому, что в FCS-файлах используются только отдельные фрагменты отпечатка пальца, можно добиться того, что эти помехи не будут оказывать влияния на правильность распознавания истинного пользователя.

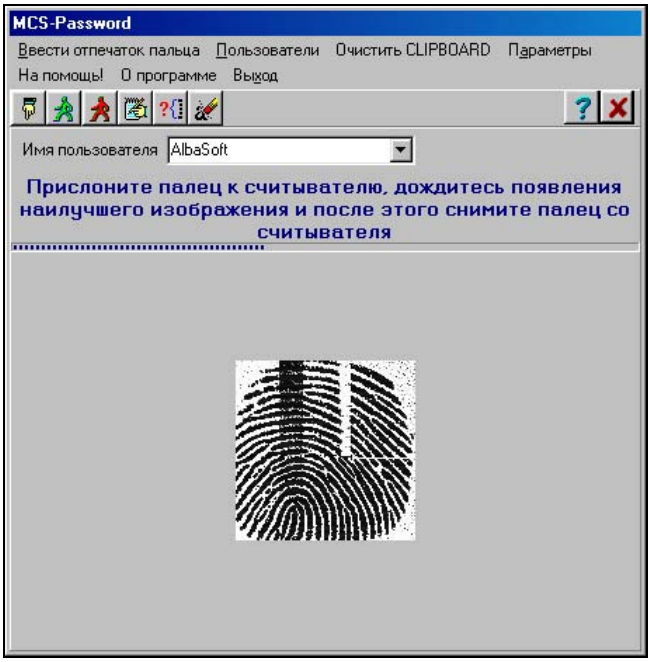

Рис. 35.6. Запрос на ввод отпечатка пальца

После правильного ввода отпечатка пальца, пользователь получает возможность обращения к своей таблице паролей (рис. 35.7).

Пользователь может добавлять в эту таблицу любое разумное число паролей. Еще раз подчеркнем, что сами выработанные случайные пароли нигде не хранятся, при необходимости их использования они каждый раз вычисляются заново с использованием секретного ключа из FCS-файла и значений, указанных в строке **Назначение пароля**. Изменение хотя бы одного символа в этой строке приводит к тому, что вычисленный пароль оказывается совершенно другим. Вот примеры паролей, вычисленных по таблице из рис. 35.7:

- A15B4CBA4DC723AF — Yandex;
- DF5CBA67F4354B6B — Rambler;
- 88B0BD075A04FD7A — АПОРТ;
- 5FF4AE5755FF1E65 — SoftLine.

Этот список появляется на экране только тогда, когда отпечаток пальца пользователя правильно введен в оперативную память компьютера. Этот момент отображается на экране изменением иконки программы — см. рис. 35.2. Для загрузки пароля в CLIPBOARD достаточно щелкнуть два раза мышью по строке с соответствующим паролем, при этом иконка краснеет — см. рис. 35.3. После использования пароля по назначению пользователь имеет возможность очистить CLIPBOAR и удалить находящийся там пароль, при этом красный цвет иконки пропадает. Если срок действия отпечатка

пальца истек, а пароль из CLIPBOARD не удален, то на экран выдается предупреждение (рис. 35.8).

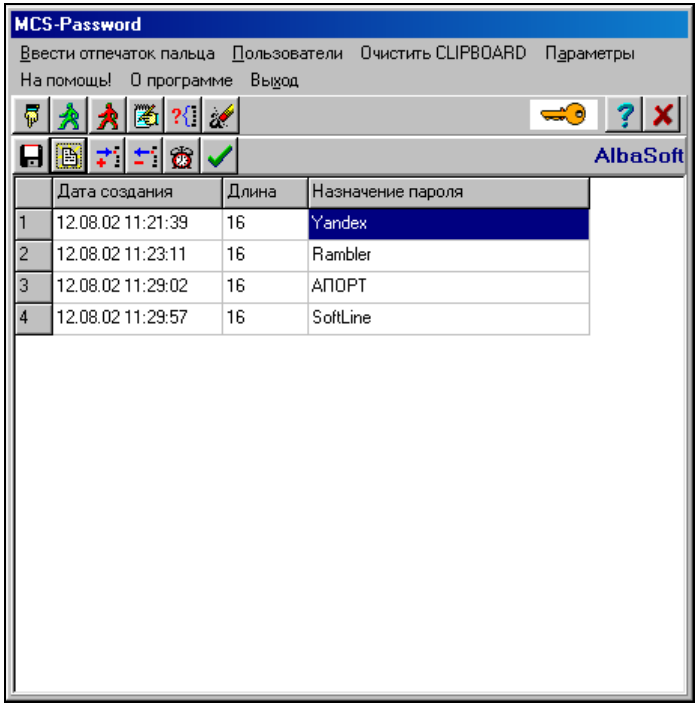

Рис. 35.7. Таблица паролей

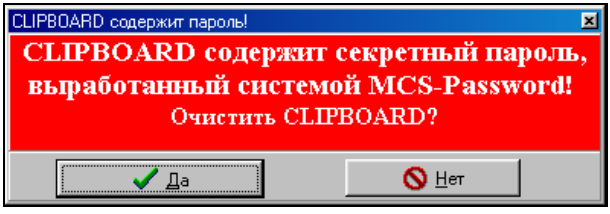

Рис. 35.8. Предупреждение о пароле, находящемся в CLIPBOARD

Вот такая получилась сравнительно простая и достаточно универсальная система генерации случайных паролей по отпечатку пальца. Конечно же, говорить о ее широком внедрении пока еще не приходится, поскольку само устройство типа FingerID достаточно экзотично и несколько дороговато для подобных целей. Но общеизвестно, насколько быстро могут развиваться и дешеветь различные компьютерные аксессуары, и то, что сегодня является дорогим и экзотичным, завтра оказывается обыденным и общедоступным. Это, на мой взгляд, в полной мере относится и к различным устройствам для ввода отпечатков пальцев в память компьютера.

Заключение

"В соответствии с решением Конгресса США (Закон 103−160 от 30 ноября 1993 г.), министру обороны предписывалось:

- поручить Национальному совету по исследованиям (NRC) провести детальное исследование по криптографическим технологиям и национальной политике в области криптографии;
- дать указание Национальному агентству безопасности (NSA), Управлению высших исследовательских проектов (ARPA) и другим органам министерства обороны, а также другим федеральным органам оказывать полное содействие NRC в этой работе.

В этом исследовании предлагалось дать оценку воздействия криптографических технологий на интересы правительства США по обеспечению национальной безопасности, интересы правоохранительной деятельности правительства США, коммерческие интересы промышленности США и интересы охраны неприкосновенности личной жизни граждан США, а также влияния контроля над экспортом криптографических технологий на коммерческие интересы промышленности США.

Отчет был подготовлен комитетом из 16 специалистов, представляющих науку (информатику, правоведение, медицину), государственные учреждения, частные фирмы и банки, и отрецензирован другой группой экспертов. Предварительная версия отчета вышла 30 мая 1996 г. и распространялась (бесплатно) издательством, а также доступна по Internet в виде фотокопий страниц по **http://www.nap.edu/readingroom/**.

Комитет пришел к выводу, что преимущества широкого распространения криптографии перевешивают его недостатки, и призвал к изменению национальной политики в криптографии. Эта политика должна преследовать следующие цели:

- широкая доступность криптографии для всех законных элементов американского общества;

- дальнейший экономический рост и лидерство ключевых отраслей промышленности США в глобальной экономике, в том числе (но не только) производства компьютеров, программного обеспечения и средств связи;
- общественная безопасность и защита от внешних и внутренних угроз.

Рекомендации комитета по национальной криптографической политике включают следующее.

1. Никакой закон не должен препятствовать производству, продаже или использованию любых видов шифрования в пределах США.
2. Национальная политика по криптографии должна разрабатываться исполнительной и законодательной ветвями власти на основе открытого публичного обсуждения и основываться на подчинении закону.
3. Национальная политика по криптографии в отношении разработки и использования коммерческой криптографии должна лучше согласовываться с силами рынка.
4. Экспортные ограничения на криптографию должны постепенно смягчаться, но не должны быть отменены полностью.
5. Правительство должно принять меры, чтобы помочь правоохранительным органам и органам национальной безопасности приспособиться к новым техническим реальностям информационного века.
6. Правительство должно разработать механизмы для обеспечения информационной безопасности в частном секторе".

Это выдержка из американского отчета "Роль криптографии в обеспечении безопасности информационного общества" (Cryptography's Role in Securing the Information Society. Kenneth W. Dam and Herbert S. Lin, Editors. Committee to Study National Cryptography Policy, Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications; National Research Council. National Academy Press, Washington D. C., 1996, 688 p., index. Library of Congress Catalog Number 96-68943, ISBN 0-309-05475-3).

С зарубежным опытом, на который так часто ссылаются в России, понятно. Рассмотрим российскую национальную криптографическую политику. Указ Президента РФ № 334 от 03.04.1995 г.

1. Придать Программе создания и развития информационно-телекоммуникационной системы специального назначения в интересах органов государственной власти статус президентской программы. Центру президентских программ Администрации Президента Российской Федерации совместно с Федеральным агентством правительственной связи и информации при Президенте Российской Федерации обеспечить ее доработку и реализацию.
2. Запретить использование государственными организациями и предприятиями в информационно-телекоммуникационных системах шифровальных средств, включая криптографические средства обеспечения подлинности

информации (электронная подпись), и защищенных технических средств хранения, обработки и передачи информации, не имеющих сертификата Федерального агентства правительственной связи и информации при Президенте Российской Федерации, а также размещение государственных заказов на предприятиях, в организациях, использующих указанные технические и шифровальные средства, не имеющие сертификата Федерального агентства правительственной связи и информации при Президенте Российской Федерации.

3. Предложить Центральному банку Российской Федерации и Федеральному агентству правительственной связи и информации при Президенте Российской Федерации принять необходимые меры в отношении коммерческих банков Российской Федерации, уклоняющихся от обязательного использования, имеющих сертификат Федерального агентства правительственной связи и информации при Президенте Российской Федерации защищенных технических средств хранения, обработки и передачи информации при их информационном взаимодействии с подразделениями Центрального банка Российской Федерации.

4. В интересах информационной безопасности Российской Федерации и усиления борьбы с организованной преступностью запретить деятельность юридических и физических лиц, связанную с разработкой, производством, реализацией и эксплуатацией шифровальных средств, а также защищенных технических средств хранения, обработки и передачи информации, предоставлением услуг в области шифрования информации, без лицензий, выданных Федеральным агентством правительственной связи и информации при Президенте Российской Федерации в соответствии с Законом Российской Федерации "О федеральных органах правительственной связи и информации".

5. Государственному таможенному комитету Российской Федерации принять меры к недопущению ввоза на территорию Российской Федерации шифровальных средств иностранного производства без лицензии Министерства внешних экономических связей Российской Федерации, выданной по согласованию с Федеральным агентством правительственной связи и информации при Президенте Российской Федерации.

6. Федеральной службе контрразведки Российской Федерации и Министерству внутренних дел Российской Федерации совместно с Федеральным агентством правительственной связи и информации при Президенте Российской Федерации, Государственной налоговой службой Российской Федерации и Департаментом налоговой полиции Российской Федерации осуществлять выявление юридических и физических лиц, нарушающих требования настоящего Указа.

7. Предложить Генеральной прокуратуре Российской Федерации усилить прокурорский надзор за соблюдением Закона Российской Федерации "О федеральных органах правительственной связи и информации" в части разработки производства, реализации и эксплуатации шифровальных средств, а также предоставления услуг в области шифрования информации в Российской Федерации, подлежащих лицензированию и сертификации Федеральным агентством правительственной связи и информации при Президенте Российской Федерации.

8. Создать Федеральный центр защиты экономической информации при Федеральном агентстве правительственной связи и информации при Президенте Российской Федерации (в пределах штатной численности этого

> Агентства), возложив на него разработку и реализацию комплексных программ обеспечения безопасности экономической информации российских кредитно-финансовых и других экономически значимых структур страны...

Запретов — огромное количество. Запреты для надежности подкрепляются Федеральной службой контрразведки Российской Федерации, Министерством внутренних дел Российской Федерации, Государственной налоговой службой Российской Федерации, Департаментом налоговой полиции Российской Федерации, Генеральной прокуратурой Российской Федерации. А что запрещается? "Разработка, производство, реализация и эксплуатация шифровальных средств". Как же при таких строгих запретах получилось, что по сведениям, содержащимся в упомянутом уже во введении к этой книге Международном отчете "Криптография и свобода", "средства шифрования, не лицензированные ФАПСИ, широко доступны в России"?

Обратимся за разъяснениями особенностей национальной криптографической политики к статье представителей самого ФАПСИ, О. А. Беззубцева и А. Н. Ковалева. Статья так и называется: "О лицензировании и сертификации в области защиты информации" и опубликована в Internet на сайте **www.security.ru**. Вот выдержки из нее (выделенные курсивом) и некоторые комментарии к ним.

"...можно констатировать, что уже имеется неплохая законодательная база, вполне позволяющая, с одной стороны, предприятиям осуществлять свою деятельность по защите информации в соответствии с требованиями действующих нормативных актов, а с другой — уполномоченным государственным органам на законной основе регулировать рынок соответствующих товаров и услуг, обеспечивая необходимый баланс интересов отдельных граждан, общества в целом и государства".

Для всех хороша существующая законодательная база — для отдельных граждан, общества в целом и государства. Только непонятно, почему же тогда

"...в последнее время в различных публикациях муссируется вопрос о том, что созданный механизм государственного регулирования в области защиты информации используется государственными органами, уполномоченными на ведение лицензионной деятельности, для зажима конкуренции и не соответствует ни мировому опыту, ни законодательству страны"?

Полностью соответствует мировому опыту — уверяют авторы статьи. Заметим: устаревшему опыту, от которого в США отказались. Буквально через год после выхода Указа № 334 в США были опубликованы приведенные выше выводы и рекомендации комитета по национальной криптографической политике, работа над которыми велась в Соединенных Штатах с 1993 года. И эти выводы и рекомендации явно не укладываются в логику ФАПСИ. Вот выдержка из Международного отчета "Криптография и свобода".

В течение долгого времени правительство США прилагало все усилия, чтобы ограничить разработку и распространение шифровальных технологий. За последние двадцать лет США неоднократно предпринимали такие попытки, манипулируя стандартами, рекомендуя принятие соответствующих законов и вводя экспортный контроль. Лишь недавно правительство признало, что электронный бизнес стал важным фактором американской экономики, и стало постепенно отказываться от прежней безрезультатной политики, вызывавшей негативные реакции в обществе.

Нет у "отдельных граждан и общества в целом" сильного желания связываться с ФАПСИ. Даже не касаясь прошлого этой организации, сегодняшний бюрократизм, нежелание считаться с интересами пользователей, навязывание своих правил, не подчиненных обычной логике, — вот типичный стиль работы ФАПСИ. Автор этих строк имел возможность долгое время наблюдать все это не только снаружи, но и изнутри. Чтобы избежать общения с этой организацией, многие ищут лазейки в сверхстрогом российском криптографическом законодательстве. И небезуспешно.

"...на данный момент времени в рассматриваемой нами области отсутствует единый, принятый во всей стране, понятийный аппарат (т. е. четкая система взаимоувязанных терминов и понятий). Многие понятия и термины, вводимые принимаемыми законами и иными нормативными актами, используемые различными органами, работающими в сфере защиты информации, или отдельными авторами, публикующими материалы по данной тематике, носят неоднозначный, противоречивый характер. В одни и те же термины часто вкладывается различное смысловое содержание. Манипулирование, а в отдельных случаях прямое спекулирование терминами и понятиями происходит зачастую из конъюнктурных соображений".

Ну вот, дожили до криптографических спекулянтов! Это, пожалуй, чисто российское явление, когда криптографические специалисты, математики по образованию, привыкшие к точным определениям и результатам, вынуждены "из конъюнктурных соображений" идти на "прямое спекулирование терминами и понятиями". Лучше бы переквалифицировались, как большинство советских инженеров, в челноков и продавцов, и не морочили бы голову чиновникам.

Чтобы обратить криптографических еретиков в истинную веру, авторы статьи взяли на себя роль криптографических миссионеров.

"Наша цель — довести до широкого круга специалистов суть и определение понятий в данной области так, как это отражено в нормативных актах ФАПСИ или официально трактуется Федеральным агентством (в случае наличия четких и ясных формулировок)".

Нет возможности пересказать здесь все это криптографическое Евангелие от О. А. Беззубцева и А. Н. Ковалева. Но некоторые выдержки — можно, особенно касающиеся тех понятий, за которые сжигают на костре Указа № 334.

*"**Шифрование** — процесс зашифровывания или расшифрования.*

***Зашифровывание данных** — процесс преобразования открытых данных в зашифрованные при помощи шифра.*

***Расшифрование данных** — процесс преобразования зашифрованных данных в открытые при помощи шифра".* Очевидно наличие ясных и четких формулировок.

*"**Шифр** — совокупность обратимых преобразований множества возможных открытых данных на множество возможных зашифрованных данных, осуществляемых по определенным правилам с применением ключей.*

***Ключ** — конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования данных, обеспечивающее выбор одного преобразования из совокупности всевозможных для данного алгоритма преобразований".*

Все предельно ясно и четко. Но, не останавливаясь на достигнутом, авторы проводят еще "анализ этих определений", по которому, как и положено, делают выводы.

"Анализ этих определений позволяет сформулировать три критерия, которые в первом приближении могут быть использованы в большинстве практических приложений для отнесения средств защиты информации к классу криптографических:

первый — наличие некоторого математического преобразования данных;

второй — наличие секретного параметра этого преобразования — ключа, с мощностью ключевого множества большего или равного двум;

третий — обратимость используемого математического преобразования данных".

Я думаю, что гг. Беззубцев О. А. и Ковалев А. Н. пользуются компьютером, операционной системой Windows, электронной почтой, Microsoft Office, различными архиваторами. Стоп! А архивирование с паролем — это разве не криптографическое средство защиты информации? Первый критерий — наличие некоторого математического преобразования — очевидно, выполнен. Второй — тоже, легко придумать два или более паролей. Обратимость — разархивирование — также присутствует. А имеют ли сами миссионеры лицензию ФАПСИ на использование такого криптографического средства? А сертифицировано ли оно?

Да и не только архиватор можно подвести под криптографическую статью. Например, Internet Explorer, SSL-протокол. Практически любая программа электронной почты в разделе **Сервис** предусматривает возможность шифрования отсылаемых сообщений. Да и в самой операционной системе Windows, если копнуть ее поглубже и залезть в реестр, можно найти встроенные функции шифрования, удовлетворяющие трем критериям Беззубце-

ва-Ковалева. Сознавая полную абсурдность такого подхода, авторы, видимо, изучавшие в свое время марксистско-ленинскую диалектику, делают следующий вывод, полностью противоположный предыдущим трем.

"При этом подчеркнем еще раз — математическое преобразование и реализованное на его базе средство защиты может быть отнесено или не отнесено к классу криптографических только на основании результатов экспертизы ФАПСИ".

Вот она, истина в последней инстанции! Про три критерия забудем, несем архиватор в ФАПСИ.

Ну а что делать с преобразованием, которое пока в ФАПСИ не донесли? Каким его считать: криптографическим или еще каким? Как быть в этом случае с грозными положениями Указа № 334? А кто должен нести преобразование в ФАПСИ? Разработчик? Продавец? Потребитель? А кто должен платить за подобную экспертизу ФАПСИ? Все гораздо проще.

"Специально подчеркнем, что факт одобрения ФАПСИ алгоритма до его технической реализации является одним из основных требований к представляемым на сертификацию изделиям. Это одобрение может быть осуществлено путем утверждения алгоритма:

в качестве государственного стандарта;

Правительством Российской Федерации;

Федеральным агентством.

Отсюда следует, что изделия, реализованные на базе собственных оригинальных алгоритмов, ранее не представлявшихся в ФАПСИ, а равно изделия, реализующие алгоритмы иностранной разработки, или импортные шифровальные средства на сертификацию не принимаются".

А вот и решение всех проблем! "Изделия, реализованные на базе собственных оригинальных алгоритмов" ФАПСИ не принимает, следовательно, они не могут быть отнесены к классу криптографических. ГОСТ — криптографическое преобразование, всем остальным — свобода!

"Все эти шарашки повелись с девятьсот тридцатого года, как стали инженеров косяками гнать. Опыт понравился. На воле невозможно собрать в одной конструкторской группе двух больших инженеров или двух больших ученых: начинают бороться за имя, за славу, за Сталинскую премию, обязательно один другого выживет. Поэтому все конструкторские бюро на воле — это бледный кружок вокруг одной яркой головы. А на шарашке? Ни слава, ни деньги никому не грозят. Николаю Николаевичу полстакана сметаны и Петру Петровичу полстакана сметаны. Дюжина медведей мирно живет в одной берлоге, потому что деться некуда. Поиграют в шахматишки, покурят — скучно. Может, изобретем что-нибудь? Давайте! Так создано многое в нашей науке! И в этом — основная идея шарашек" (А. И. Солженицын. "В круге первом").

Это — наше не столь давнее прошлое. Не будем о нем забывать при попытке понять современную национальную криптографическую политику.

Давайте отбросим все сиюминутные конъюнктурные соображения и постараемся взглянуть на проблему гарантированной стойкости глазами математика, признающего только бесспорные аксиомы и получающиеся на их основе строгие результаты. Что такое гарантированная стойкость, достижима ли она, и если да, то что для этого требуется?

В качестве абсолютной и бесспорной аксиомы смею утверждать, что абсолютно гарантированной на все случаи жизни стойкости нет и быть не может. Это как абсолютная истина в философии: состоит из бесконечного множества относительных истин, но никогда не достижима. Гарантированная стойкость складывается из отдельных кирпичиков, каждый из которых несет в себе определенную функциональную нагрузку, а все вместе они могут обеспечивать общую достаточно высокую стойкость, но не абсолютную. Практически гарантированную, но не абсолютную. Задачей как разработчика, так и потребителя криптографической системы является осознание того, какие основные кирпичики должны быть в системе, проверить, чтобы они фактически присутствовали в ней и выполняли положенные им функции.

Одной из наиболее важных составных частей гарантированной стойкости является криптографическое ядро, положенные в его основу базовые криптографические преобразования: алгоритмы выработки ключей, шифрования и подписи. Утверждения о том, что лучше, чем алгоритмы DES и ГОСТ, мировая криптография ничего не знает — лукавые. В середине 70-х годов одна спецслужба (USA) по недосмотру выпустила алгоритм DES, предназначенный для аппаратной реализации. Эффективность его использования в программных реализациях можно сравнить с эффективностью паровоза на современных железных дорогах. В ответ на это другая спецслужба (СССР), забыв про национальную гордость и авторитет отечественной криптографии, не нашла ничего лучшего, как в отместку приделать к этому паровозу свою трубу и поднять на нем красный флаг, объявив в 1989 году национальным советским стандартом. А с победой демократии в России наступил расцвет чиновничьей криптографии, когда все криптографические проблемы свелись к стремлению заставить всех без разбору ездить на паровозах.

Число криптографических алгоритмов, обладающих гарантированными свойствами, гораздо шире. Они проще для программной реализации, обладают более высокими скоростями, требуют меньше ресурсов. А самое, на мой взгляд, главное состоит в том, что оригинальный алгоритм является еще одним рубежом защиты от потенциального злоумышленника, еще одним кирпичиком в стене, именуемым гарантированная стойкость. Впрочем, здесь нужна оговорка. Речь идет об алгоритмах для традиционной криптографии, не связанной с открытыми ключами. Для систем с открытым распределением ключей приемлемых алгоритмов не так уж и много: алгоритмы Диффи-Хеллмана да RSA. Возможны два варианта: или криптографы еще

не научились строить их в большом количестве, как алгоритмы для традиционной криптографии, или не за горами тот день, когда найдутся методы взлома и этих. На всякий случай подстраховка из традиционной криптографии в системах, претендующих на гарантированную стойкость, не помешает.

Но качественное криптографическое ядро — это хотя и очень важная, но далеко не единственная составляющая гарантированной стойкости. Хакер, убедившись в тщетности попыток взлома системы защиты криптографическими методами, будет искать всяческие способы компрометации ключей. Получив секретный ключ, он часто оказывается в тех же условиях, что и законный пользователь. Здесь не спасет никакое даже самое хорошее криптографическое ядро. Нужно не допускать компрометации ключей всеми возможными средствами. А к таким средствам относятся, во-первых, чисто организационные. Строгий контроль за компьютерами, на которых используется криптографическая система, включая контроль за создаваемыми ими побочными излучениями. Это целое направление в области обеспечения информационной безопасности, выходящее за рамки этой книги. Без него нельзя говорить о гарантированной стойкости. Разработчик криптографической системы может ограничиться общими рекомендациями, направленными на снижение вероятности утечек по побочным каналам: иметь для компьютера развязку по сети питания, не ставить компьютер вблизи телефонных аппаратов, проводов общей сети электропитания, обеспечить вокруг компьютера контролируемую зону, использовать для подавления побочных сигналов различные шумовые генераторы и т. п. А лучше всего — использовать Notebook с жидкокристаллическим экраном на автономном питании. Для особо конфиденциальных целей — еще где-нибудь за городом, на открытом месте.

Вторая составная часть организационных мер касается персонала, работающего с криптографической системой. Гарантированная стойкость немыслима без создания некоторых неудобств для пользователей и в первую очередь это связано с секретными ключами. Не может криптографическая система, претендующая на гарантированную стойкость, помочь пользователю, если он потерял дискету с секретным ключом или забыл пароль. Создавайте на такой случай дубликаты дискет, храните их в строгом секрете, но программа не должна оказывать в этом помощи пользователю, ибо тогда она станет слишком легкой добычей для хакеров. Никакой дополнительной информации о секретных ключах в программной реализации криптографической системы гарантированной стойкости быть не должно! А пользователям нужно подготовить соответствующие четкие должностные инструкции, правила, провести обучение. Нельзя оставлять дискету с секретным ключом без присмотра, нельзя никуда записывать пароль, нельзя использовать компьютер с криптографической системой для неслужебных целей. Без работы с персоналом достичь гарантированной стойкости нельзя.

Задачей разработчика, также непосредственно связанной с гарантированной стойкостью, является создание для пользователей удобного интерфейса,

максимальная автоматизация однотипных процессов, контроль за выполнением криптографических и иных операций, блокировка опасных ошибок оператора.

Убедившись в несостоятельности попыток заполучить секретный ключ физически, хакер попытается украсть его программно. Криптовирусы, программные закладки, дописывание секретного ключа к шифртексту, сканирование оперативной памяти в момент нахождения в ней секретного ключа — большой простор для полета фантазии. Гарантию от подобных вещей можно дать только в том случае, если компьютер не подключен ни к какой сети и на нем никто не работает, кроме проверенных пользователей криптографической системы. А это в реальной жизни практически неосуществимо. И здесь мы опять же возвращаемся к криптографическим стандартам. Стандарт — хорошее поле деятельности для хакера. Чем шире он распространен, чем больше о нем информации, тем больше вероятность того, что под него уже написан криптовирус или еще какая-то программная закладка для компрометации секретных ключей. Оригинальное криптографическое решение, оригинальная программная реализация — дополнительный аргумент в пользу гарантированной стойкости.

И все же исключать в реальной жизни компрометации ключей нельзя. Значит, нужно сделать все возможное, чтобы локализовать последствия такой компрометации, "не класть все яйца в одну корзину". Периодически проводить полную смену всех секретных и открытых ключей. Соответственно заранее и обстоятельно готовиться к такой смене, предусматривать ее возможность при проектировании криптографической системы. Это тоже составная часть обеспечения гарантированной стойкости.

И, наконец, может быть один из самых главных моментов. Это доверие к разработчику криптографической системы, к его профессионализму, порядочности, опыту. Уверенность в том, что он не будет целенаправленно вводить в систему Троянских коней, жучков и закладки, потому что дорожит своим именем.

И возрождение нашей страны, о котором так много говорят в последнее время, обязательно приведет к возрождению ее сильной криптографии. Гражданской криптографии.

Приложение

Содержание компакт-диска

На диске находятся папки, в которых расположены:

- FingerID\DOC — документация к пакету прикладных программ, связанных с использованием отпечатка пальца. Для просмотра документации необходимо открыть файл FT.HTM.
- FingerID\SETUP — демонстрационный пакет программ, связанных с использованием отпечатка пальца. Для установки пакета необходимо запустить файл SETUP.EXE.
- INTERNET — перечень наиболее интересных WEB-страниц, посвященных криптографии. Для его просмотра необходимо открыть файл LIBRARY.HTM.
- LotusNotes\DOC — документация к пакету прикладных программ "Криптоцентр для Lotus Notes". Для ее просмотра необходимо открыть файл CryptoCentreForLotus.HTM.
- MCS\DOC — документация к библиотеке криптографических процедур, используемых в системе "Криптоцентр". Для ее просмотра необходимо открыть файл INDEX.HTM.
- MSOutlook\DOC — документация к пакету прикладных программ "Криптоцентр для MSOutlook". Для ее просмотра необходимо открыть файл CryptoCentreForExchange.HTM.
- MSOutlook\SETUP — демонстрационный пакет программы "Криптоцентр для Microsoft Outlook". Для его установки необходимо запустить файл SETUP.EXE.
- OMEGA\DOC — документация по автоматизированной системе электронного документооборота OMEGA. Для ее просмотра необходимо запустить файл INDEX.HTM.

Список литературы

1. Задача нахождения ключа шифрования RSA решена. // NewScientist. — 1994. — т. 142, № 1924. — P. 5.
2. Ирвинг де ла Круз, Лес Талер. Основы MAPI. — Пер. с англ. — М.: Издательский отдел "Русская редакция" ТОО "Channel Trading Ltd.", 1997.
3. Кан Д. Взломщики кодов. — М.: Центрополиграф, 2000.
4. Крайсл Б. Изучи сам Lotus Notes. — Пер. с англ. — Минск: ООО "Попурри", 1998.
5. Рейсдорф К., Хендерсон К. Освой самостоятельно Borland C++ Builder. — М.: БИНОМ, 1998.
6. Солженицын А. В круге первом. Роман. — М.: "Голос", 1994.
7. Шамис В. А. Borland C++ Builder 3. Техника визуального программирования. — М.: Нолидж, 1998.
8. Шнайер Б. Прикладная криптография, 2-е издание: протоколы, алгоритмы и исходные тексты на языке С. — (перевод с оригинала Applied Cryptography, Second Edition: Protocols, Algorthms, and Source Code in C (cloth) Publisher: John Wiley & Sons, Inc. Author(s): Bruce Schneier ISBN: 0471128457 Publication Date: 01/01/96).
9. Lotus Notes C Application Program Interface (API) // Reference Release 4.6.2. 1996−1998 Lotus Development Corporation.
10. Messaging Application Programming Interface (MAPI) // Programmer's Reference. ©1996 Microsoft Corporation.
11. Michael C. Amundsen. MAPI, SAPI, TAPI Developer's // Guide (Sams; ISBN: 0672309289; 1st edition, October 1996).

Internet-ресурсы

1. Андреев Н. Н. Россия остается в числе лидеров мировой криптографии. Журнал "Конфидент", № 24. **http://www.ssl.stu.neva.ru/psw/crypto/Andreev23.html**.
2. Беззубцев О. А., Ковалев А. Н. О лицензировании и сертификации в области защиты информации. **http://www.security.ru/sert0.html**.

3. Винокуров А. Цикл статей по криптографии. Введение. Криптография, ее истоки и место в современном обществе. Октябрь 1998 г. **http://www.enlight.ru/ib/tech/crypto/index.htm**.
4. Криптография и свобода 2000. Международный отчет о ситуации с криптографией. Electronic Privacy Information Center, Вашингтон. Отчет по странам. **http://www.hro.org/docs/reps/crypto2k/countries.htm**.
5. Кузьмин Л. А. ГУСС — этап в развитии советской криптографии. Журнал "Конфидент", N 22. **http://www.ssl.stu.neva.ru/psw/crypto/Kuzmin22.html**.
6. ЛАН-Крипто. Защита информации. Алгоритмы. **http://www.lancrypto.com/index_ie_rus.html**.
7. Лунин А. В., Сальников А. А. Перспективы развития и использования асимметричных алгоритмов в криптографии. Журнал "Конфидент", N 24. **http://www.ssl.stu.neva.ru/psw/crypto/Lunin24.html**.
8. Молдавский государственный университет. Курс "История криптографии". Лекция 4. Криптография в древние времена. **http://virtualka.boom.ru/kr4.html**.
9. ОАО "Инфотекс" (Информационные технологии и коммуникационные системы). Продукты: ViPNet. "Домен-К". **http://www.infotecs.ru/Soft/domen_k.htm**.
10. Официальный сайт Института криптографии, связи и информатики. **http://www.fssr.ru**.
11. Средства криптографической защиты информации. Московское отделение Пензенского научно-исследовательского электротехнического института. Технология "Витязь". **http://www.security.ru**.
12. Султанов Б., Семенов Д., Клюев А. Электронный учебник по теории информации. **http://www.msclub.ce.cctpu.edu.ru/bibl/TI/Index.html**.
13. Указ Президента Российской Федерации № 334 от 03.04.95 "О мерах по соблюдению законности в области разработки производства, реализации и эксплуатации шифровальных средств, а также предоставления услуг в области шифрования информации". **http://www.security.ru/uk4.html**.
14. Ященко В. В. Криптография, которая раньше была засекречена, Компьютерра, № 20 (248), 1998. **http://www.cryptography.ru/db/msg.html?mid=1161991**.
15. Cryptography's Role in Securing the Information Society. Kenneth W. Dam and Herbert S. Lin, Editors. Committee to Study National Cryptography Policy, Computer Science and Telecommunications Board, Commission on Physical Sciences, Mathematics, and Applications; National Research Council. National Academy Press, Washington D. C., 1996, 688 p., index. Library of Congress Catalog Number 96-68943, ISBN 0-309-05475-3. **http://www.infomag.ru:8082/dbase/N002R/970210-037/text0020.html**.

Предметный указатель

B

BuildFCS 426

C

CryptBlock 412

D

DDB (Document Data Base) 296

F

FCS-файлах 413
FCS-файлов 428
FCS-файлом 410
FingerID 398, 415, 441

I

Internet-пакетов 345

K

KeyForFinger 416
KeyFromFinger 424

L

LotusScript 229, 261, 264

M

MaxAttempt 328
MaxWait 328
MCS_FL.DLL 425
MCS-Finger 406, 414
MCS-key 215
MCS-Password 438

R

RDB (Reestr Data Base) 296

S

SDB (Sign Data Base) 296

А

Администратор 286
Администрация 286
Адресная книга 217
Альтернативнаяпочта 338
Аннулирование
  в протоколе подписи 332
Архив 296

Б

Базы данных:
  Notes 199
  главного менеджера 365
  менеджера СБ 377
  отправок 374
  рассылок 381
Базового секретного ключа 213

В

Внешний 380
Внешний секретный ключ 415
Внутренний секретный ключ 415
Выписки для системы выработки ключей 373
Выписки из архива 362

Г

Главный менеджер системы OMEGA 285

Д

Документ:
Note 199
типа пакет 294

И

Индивидуальные конфигурационные файлы системы OMEGA 285
Инсталляционные пакеты 373

К

Ключевая книга 209, 227, 207
Конструктор 229
Контроль за доставкой электронных документов 357
Конфигурационные файлы 302
Криптосервер 413, 427
Криптоцентр 391, 429

М

Менеджер
системы безопасности 286, 377
Мозаичная перестановка сигнала 412

Н

Набор фрагментов 410

О

Объединенного файла 329

П

Пароль 213, 395
для доступа к хосту и для приема почты 348
типа RichText 232, 251, 261
Пользователями системы OMEGA 284
Протокол 291

Р

Региональный менеджер 374, 381, 386, 289

С

Сведения о рассылках ключей 383
Сертификация открытых ключей 226
Система рассылки секретных ключей 384
Система с открытым распределением ключей 216, 397

Т

Текстовые поля 230, 251
Точность ключа 409

У

Удаленный 309

Ф

Файл-директива 329
Файл-реестр 291
Фильтры 359

Х

Хэш-вектор 263
Хэширование 251, 269, 273
Хэш-функция 251, 263, 269

Ц

Центральные менеджеры 386
Циркулярные 328

Э

Электронная подпись 263
Электронные документы 291