b!.b!.P.

Когда стали широко использоваться алгоритмы шифрования при передаче данных в сети, одной из первых возникла задача организации безопасной оболочки. До этого существовала система rsh, которая позволяла определённым пользователям с определённых машин (между ними должны были быть доверительные отношения) работать на сервере с его оболочкой. Это практически то же самое, что и telnet-доступ. Но с развитием сетей стали видны вопиющие дыры rsh: данные, передаваемые через сеть, никак не шифруются, включая пароли; они, могут быть без проблем получены либо модифицированы третьей стороной; злоумышленник может спокойно подменить ip клиента и, используя полученный ранее хеш пароля, пройти аутентификацию на сервере со всеми вытекающими последствиями. Поэтому сейчас rsh применяется в чрезвычайно редких случаях, например, при переносе данных между двумя попарно соединёнными машинами (мне так пришлось работать с двумя машинами в разных комнатах). В основном стандартом де-факто стал ssh.

ВСЕВОЛОД СТАХОВ

Первая буква «s» означает безопасный (secure), то есть все данные, передаваемые через ssh, шифруются, а значит, защищены от просмотра. Существует несколько версий протокола ssh, различающиеся используемыми алгоритмами шифрования и общими схемами работы. В настоящее время повсеместно используется протокол ssh версии 2. Протокол младших версий является по современным меркам небезопасным (там есть несколько очень опасных дыр). Вообще-то сейчас ssh является коммерческим продуктом (что само по себе противоречит требованиям безопасности – всем должен быть известен исходный код системы защиты информации, чтобы убедиться в отсутствии всяких backdoors), но тем не менее доступна свободная реализация ssh – OpenSSH, которая может быть найдена на www.openssh.com. Наилучшим документом по ssh является, по-моему, банальный man ssh, поэтому в некоторых местах я не постесняюсь его просто переводить.

Итак, начнём, как обычно, с теории. SSH предоставляет 3 способа аутентификации клиента: по ip-адресу клиента (небезопасно), по публичному ключу клиента и стандартный парольный метод. Вот как работает ssh версии 2: при запросе клиента сервер сообщает ему, какие методы аутентификации он поддерживает (это определяется в опции Preferred Authentications sshd.conf), и клиент по очереди пытается проверить их. По умолчанию клиент вначале пытается аутентифицироваться своим адресом, затем публичным ключом и, если ничего не сработало, передаёт пароль, введённый с клавиатуры (при этом пароль шифруется асимметрическим шифрованием). После прохождения аутентификации одним из методов из имеющихся у клиента и сервера пар

ключей генерируется ключ симметрического шифрования, который, как я описывал ранее (см. статью «Теория и практика OpenSSL»), генерируется на основании своих секретного и удалённого публичного ключей. После чего все последующие данные, передаваемые через ssh, шифруются данным ключом (обычно используется алгоритм aes с длиной ключа 128 бит). Отмечу, что протокол ssh версии 1 имел некоторые баги в шифрации передаваемого трафика и являлся, по сути, методом безопасной аутентификации, поэтому по современным меркам данный протокол считается небезопасным. Протокол версии 2 поддерживает более современные методы шифрования трафика, также вместе с данными посылаются контрольные суммы формата sha или md5, что исключает подмену или иную модификацию передаваемого трафика (чего не было у ssh версии 1).

Теперь пару слов о способах аутентификации пользователей через ssh.

По адресу клиента

При данном способе аутентификации происходит следующее: каждый клиент и сервер имеют свои пары ключей RSA, которые называются ключами хоста. При этом существует несколько методов проверки адреса клиента. Сервер смотрит файлы $HOME/.rhosts, $HOME/.shosts, /etc/hosts.equiv или /etc/ssh/shosts.equiv, если же сервер настроен на проверку ключей клиентов (а это нужно в соображениях безопасности, т.к. иначе злоумышленник может подменить ip-клиента на свой), то он дополнительно проверяет /etc/ssh/ssh_known_hosts и $HOME/.ssh/known_hosts. Естественно, что файлы, расположенные в домашних каталогах сервера, действуют на пользователя, в чьём каталоге они размещены, а файлы, расположенные в /etc, имеют глобальный эффект. Для начала расскажу о синтаксисе вышеперечисленных файлов:

- .rhosts – определяет адрес машины и имя пользователя, с которой данному пользователю открыт доступ (файл расположен в домашнем каталоге пользователя);
- .shosts – аналогичен .rhosts, но предназначен исключительно для ssh, поэтому использовать лучше именно данный файл. Пример .shhosts:

```
user1.test.ru user1  userstend.test.ru
user1  null.test.ru user1
```

- /etc/hosts.equiv – также содержит пары имя машины/имя пользователя, но имеет эффект на всех пользователей;
- /etc/shosts.equiv – аналог hosts.equiv, но применяется только ssh, что также более предпочтительно. Пример файла:

```
/etc/shhosts.equiv  + user1.test.ru
user1  - server.test.ru xakep
```

Знак «+» означает разрешение пользователю работать с сервером с данного адреса, знак «-» запрещает подобное действие.

```
* /etc/ssh/ssh_known_hosts и $HOME/
.ssh/known_hosts
```

– данные файлы содержат список адресов и соответствующих им публичных ключей. При запросе клиента сервер генерирует рандомную строку и шифрует её публичным ключом удалённого хоста. Клиент, получив данную строку, расшифровывает её своим секретным ключом (который имеется только у него) и зашифровывает полученную строку ключом сервера. Сервер получает зашифрованное сообщение, расшифровывает своим секретным ключом и сравнивает с исходной. Если строки совпали, то клиент имеет валидный секретный ключ, что даёт ему право захода на данный сервер. Но для начала клиент должен иметь правильный адрес, которому соответствует публичный ключ на сервере в файле ssh_known_hosts. Файл состоит из 3-х полей: адрес (или адреса, разделённые запятой), публичный ключ для него одной (!) строкой и дополнительное поле комментариев (необязательно). Пример файла known_hosts:

```
user1.test.ru
 {SOME_VERY_LONG_PUBLIC_KEY}
```

Адрес клиента должен быть в полном формате (name.domain), иначе могут быть проблемы. Кроме этого, в адресе можно использовать шаблоны «*» и «?». Публичные ключи вставляются в данный файл самим администратором из генерированных клиентом ssh (identity.pub) публичных ключей. Вообще создание ssh_known_hosts – это прерогатива администратора (aka root).

И ещё добавлю: при аутентификации по хосту лучше использовать ssh_known_hosts, т.к. этот метод достаточно безопасен, если публичные ключи клиентов были получены из доверенного источника. Другие методы аутентификации не исключают подмену адреса, и потому считаются небезопасными.

Аутентификация пользователя по его публичному ключу

Аутентификация удалённого пользователя по ключу идентична проверке ключа хоста (с посылкой рандомной строки) за тем исключением, что проверяется не адрес клиентской машины, а ключ клиента и имя пользователя. Данному пользователю на сервере может соответствовать его публичный ключ, тогда клиент, имея секретный ключ, сможет заходить на сервер без пароля. Механизм работы я только что описал, поэтому сразу же расскажу, каким образом аутентифицировать пользователей по ключу (предполагается, что используется клиент и сервер openssh).

Для генерации пары ключей используйте программу ssh-keygen. Для указания типа ключа укажите ssh-keygen -t {RSA DSA}, например, ssh-keygen -t rsa создаст пару ключей RSA длиной 1024 бита. Для указания файла, в котором следует сохранить ключи, можно использовать опцию -f (традиционно используются файлы $HOME/.ssh/id_rsa и $HOME/.ssh/id_dsa для ключей rsa и dsa соответственно), для указания длины ключа в битах используйте опцию:

```
-b: ssh-keygen -t rsa -b 2048 -f
$HOME/.ssh/id_rsa
```

В результате работы программа запросит ввод пароля для шифрования секретного ключа, чтобы исключить использование его при попадании к посторонним лицам, не знающим пароля (пароль желательно выбирать не короче 10-и символов). После этого вам будет необходимо вводить данный пароль каждый раз при использовании секретного ключа (далее я расскажу, как избежать этого

при помощи программы ssh-agent). После работы ssh-keygen создаётся пара ключей: один секретный (зашифрованный введённым паролем), а другой – публичный с расширением .pub (id_rsa.pub). Публичный ключ вам необходимо будет скопировать в домашнюю директорию сервера $HOME/.ssh/authorized_keys. После этого сервер будет знать ключ данного пользователя и сможет аутентифицировать вас без пароля. Файл authorized_keys может содержать несколько публичных ключей, допустимых для данного пользователя: просто поместите их в данный файл по порядку. После этих операций вы сможете входить, имея секретный ключ, на сервер, где размещён ваш публичный ключ, причём под тем пользователем, в чьём домашнем каталоге данный ключ находится. Пароля удалённого пользователя не требуется, необходимо только знать пароль расшифровки секретного ключа. Для переноса своего публичного ключа на сервер надо использовать только безопасные источники, иначе ваш ключ могут подменить. Для переноса публичного ключа клиента служит программа ssh-copy-id. Для начала необходимо сделать следующее:

```
  # ssh-copy-id   -i public_key_file
user@machine
```

После соединения с сервером machine и передачей имени пользователя user (необходимо указывать, если удалённое имя отличается от локального) происходит парольная аутентификация заданного пользователя (или текущего) на удалённой машине, затем происходит копирование ключа public_key_file (или $HOME/.ssh/identity.pub, если имя файла не указано) на сервер в $HOME/.ssh/authorized_keys. После этого можно входить на сервер, не используя пароль пользователя. При выполнении данной операции учтите, что вы должны скопировать на удалённую машину ПУБЛИЧНЫЙ ключ, иначе всё будет очень печально (думаю, ясно почему).

Обычная парольная аутентификация

Тут можно отметить только одно: в любом случае вначале идёт обмен асимметрическими ключами, и хеш пароля передаётся в зашифрованном виде. Парольная аутентификация используется наиболее часто, но, честно говоря, ssh предлагает более удобные методы аутентификации, и пользоваться ими можно, если к ssh есть все заплатки. И, конечно же, протокол версии 1 необходимо вырубить вообще. Итак, начинаем настройку...

Я заметил, что большинство администраторов просто оставляют конфиги клиента и сервера по умолчанию, чтобы руки не марать. Но это неправильно: в разных системах эти конфиги различаются очень существенно, и это приводит к неразберихе и непониманию работы сервера, что создаёт дополнительную угрозу безопасности (свой сервак – потёмки). Для этого я решил описать файлы конфигурации ssh на примерах ssh_config и sshd.conf для клиента и сервера соответственно. Для конфигурации клиента используется файл $HOME/.ssh/config или /etc/ssh/ssh_config (для всей системы). Файл имеет следующий формат: определение адреса хоста и параметры для него. В адресе можно использовать обычные шаблоны «*» и «?», все имена параметров и их значения должны быть набраны в том же регистре, что и в примере (иначе параметр воспринят не будет). Вот пример ssh_config, который содержит наиболее полезные опции (на самом деле описывать некоторые параметры конфигурации ssh не имеет смысла, т.к. употребляются они очень редко):

```
# Определение хоста, в данном случае
включает все хосты домена test.ru, мож-
но использовать одиночный символ «*» что-
бы указать параметры доступа к любому
хосту
Host *.test.ru

# Эта опция определяет, будет ли ssh
использовать передачу данных от удалён-
ного X сервера через свой безопасный
канал. Далее будет описано, каким обра-
зом организуются безопасные туннели че-
рез ssh. Данная возможность позволяет
защищать по идее небезопасные
протоколы(X, pop, smtp, ftp) шифрован-
ем ssh. По умолчанию данная опция – no
ForwardX11 yes

# Список предпочтительных методов аутен-
тификации через ssh версии 2. Первым
стоит самый предпочтительный протокол,
думаю, значения данного параметра ясны
PreferredAuthentications hostbased,
publickey,keyboard-interactive

# Этот параметр определяет, будет ли
производится стандартная парольная про-
верка. По умолчанию – yes.
PasswordAuthentication yes

# Число попыток ввода пароля перед тем,
как клиент отсоединяется от сервера. По
умолчанию пароль можно вводить трижды
NumberOfPasswordPrompts 3

# Список допустимых пользователей для
данного сервера. Можно применять два фор-
мата: список пользователей, разделён-
ных пробелом, и список пользователей и
хостов, разделённых пробелом (USER@HOST
- разрешает данному пользователю доступ
только с данного адреса). Можно исполь-
зовать выражения «*» и «?». Подобное же
назначение имеют опции AllowGroups,
DenyUsers и DenyGroups(для групп нельзя
указывать адрес клиента)
AllowUsers *@*.test.ru DenyUsers xakep
lamer DenyGroups x*

# Использование ssh(2 версия) аутенти-
фикации через rhosts и RSA ключи. По
умолчанию – no
HostbasedAuthentication yes

# Будет ли клиент пытаться работать по
rsh, если ssh недоступен или по каким-
то причинам работает неправильно. По
умолчанию – no
FallBackToRsh no

# Используем ли rsh. По умолчанию – no
UseRsh no

# Режим скрипта, когда не спрашиваются
пароли с терминала. По умолчанию – no
BatchMode no

# Дополнительно проверяется ключ хоста
удалённой машины в known_hosts, что ис-
ключает подмену ip. По умолчанию – yes
CheckHostIP yes

# Данный параметр означает, будет ли
клиент доверять полученным от серверов
ключам. Параметр может принимать следу-
ющие значения: yes - ключи никогда ав-
томатически не помещаются в known_hosts,
ask - ключ может быть помещён в
known_hosts только после подтверждения
пользователя, no - все ключи автомати-
чески размещаются в known_hosts (небе-
зопасно). По умолчанию - ask
StrictHostKeyChecking ask

# Следующие параметры определяют сек-
ретные ключи ssh различных форматов:
rsa и dsa
IdentityFile  $HOME/.ssh/id_rsa
IdentityFile $HOME/.ssh/id_dsa

# Порт, на удалённой машине использу-
емый ssh. По умолчанию 22   Port 22
Версии протоколов, используемые клиен-
том в порядке убывания приоритета
Protocol 2

# Протокол шифрования для версии 1 про-
токола ssh
Cipher 3des

# Возможные протоколы шифрования в по-
рядке убывания приоритета для протокола
версии 2
Ciphers aes128-cbc,3des-cbc,blowfish-
cbc,cast128-cbc,arcfour,aes192-
cbc,aes256-cbc

# Значение escape-символа, сигнализи-
рующего, что идущие за ним символы
необходимо воспринимать специальным об-
разом (например ~. вызовет немедленное
отключение клиента от сервера) при пе-
редаче двоичных данных необходимо уста-
новить этот параметр в none, что выклю-
```

```
чает escape последовательности. По умолчанию ~
EscapeChar ~

# Управление работой компрессии зашифрованнного трафика. Полезный параметр для медленных сетей, т.к. зашифрованные данные обычно увеличиваются в размере за счёт фиксированной длины ключа. Компрессия позволяет уменьшить количество данных, передаваемых через сеть, но увеличит время работы самого протокола. Так что включать этот параметр желательно на медленных соединениях. По умолчанию – no
Compression yes

# Управляет посылкой сообщений о доступности клиента серверу, что позволяет нормально разорвать соединение, если произошла неполадка в сети или иная, приведшая к разрыву соединения. Если связь плохая, то лучше эту опцию отключить, чтобы дисконнект не происходил после каждой ошибки сети. По умолчанию – yes
KeepAlive yes
```

Я думаю, что в данном примере всё объяснено достаточно подробно и скажу только вот что: в большинстве случаев опции по умолчанию работают неплохо, необходимо только отключить поддержку ssh версии 1 и настроить необходимые методы аутентификации (кроме парольной) и указать пути доступа к ключам. На этом закончим с настройкой клиента и настроим сервер. Файл конфигурации сервера sshd находится в /etc/ssh/sshd_config, и многие его параметры совпадают с аналогичными в ssh_config, но здесь нет определений хостов, как это было в ssh_config. Я всё же приведу пример sshd_config, чтобы далее не возникало вопросов:

```
  # Номер порта и версия протокола
  Port 22 Protocol 2

  # Адреса, на которых слушает сервер, можно также указывать порт (server.test.ru:2022), но назначение ssh нестандартного порта нецелесообразно, т.к. заинтересует потенциальных взломщиков («А чего это там они прячут?»)
ListenAddress server.test.ru

  # Ключ сервера для протокола версии 1
  HostKey /etc/ssh/ssh_host_key

  # Ключи rsa и dsa для ssh версии 2
    HostKey /etc/ssh/ssh_host_rsa_key
HostKey /etc/ssh/ssh_host_dsa_key

  # Данные значения определяют длину ключа сервера и его время жизни для использования ssh версии 1 (данный ключ будет заново генерироваться через заданное время)
  KeyRegenerationInterval 3600
  ServerKeyBits 768

  # Далее определяем методы аутентификации для данного сервера и её параметры.

  # Сервер отсоединяется по происшествии данного времени в секундах, если клиент не проходит аутентификацию
  LoginGraceTime 600

  # Разрешаем заходить по ssh руту. Долгое время эта тема обсуждалась на форуме, но я думаю всё же, что со внутренней сети рут может заходить и по ssh (для этого надо настроить должным образом iptables). Также можно запретить руту входить по паролю: without-password, разрешая вход только по публичному ключу
  PermitRootLogin yes

  #Проверка sshd прав доступа и владельцев домашних каталогов. Полезно для тех пользователей, что дают права всему 0777. Хотя таких болванов лучше держать на расстоянии от сервера (лучше всего это делать бревном, подвешенным в серверной к потолку, чтобы придать нежеланному гостю должное ускорение, и не забудьте оббить конец бревна какой-нибудь железкой, иначе брёвна придётся менять лишком часто)
  StrictModes yes

  # Аутентификация через RSA (версия 1)
   RSAAuthentication yes

  # Аутентификация пользователя по ключу (версия 2)
  PubkeyAuthentication yes

  # Определяет публичный ключ пользователя для аутентификации по ключу. Можно применять шаблоны: %u – имя пользователя, %h – домашний каталог пользователя
  AuthorizedKeysFile      .ssh/
  authorized_keys

  # Не используем аутентификацию rhosts
RhostsAuthentication no

  # Можно также игнорировать rhosts и shosts при hostbased autentification, используя только known_hosts файл
IgnoreRhosts yes

  # Используем ли аутентификацию через known_hosts совместно с .rhosts или .shosts. Опция действительна только для протокола версии 1
  RhostsRSAAuthentication no

  # То же самое, что и предыдущее только для версии 2
  HostbasedAuthentication yes

  # Если нет доверия к known_hosts, то их можно не использовать при hostbased autentification. По умолчанию – no
IgnoreUserKnownHosts no

  # Чтобы запретить посылку хешей паролей через туннель ssh задайте значение данной опции no. По умолчанию аутентификация по паролю разрешена
PasswordAuthentication yes

  # Можно также разрешить пустые пароли, но это полный отстой, т.к. это огромная дыра на сервере, через которую можно наделать много гадостей! Поэтому должно быть – no (по умолчанию)
PermitEmptyPasswords no

  # Аутентификация через механизм PAM
PAMAuthenticationViaKbdInt no

  # Передача протокола иксов через туннель ssh
  X11Forwarding yes

  # Используем в качестве x-сервера данный, т.е. клиент, запуская у себя x-клиента будет фактически использовать наш сервер, но все данные от сервера к клиенту будут шифроваться, что есть хорошо!
  X11UseLocalhost yes

  # При логине пользователя выводим /etc/motd: в некоторых системах это отменено в целях безопасности
  PrintMotd yes

  # Сообщаем пользователю время и место последнего логина, ситуация, аналогичная предыдущей
  PrintLastLog yes

  # Посылать клиенту сообщения о доступности
  KeepAlive yes

  # Максимальное число возможных соединений, где не произошло аутентификации. Если клиентов, не прошедших аутентификацию больше, то новые соединения не будут обрабатываться
  MaxStartups 10

  # Путь к файлу, который будет отображаться при входе клиента ДО аутентификации
  Banner /etc/ssh_message

  # Проверка соответствия ip-адреса клиента и его символического имени в backzone, затем снова сравнение имени с ip адресом. Таким образом (извращённым) проверяется подлинность ip, но метод этот достаточно тормозной и по умолчанию он отключен
  VerifyReverseMapping no

  # Новые системы, работающие через ssh. В данном примере определяется «безопасный» ftp сервер – sftp, аналогичный доступ пользователя, но с возможностью передачи файлов (т.е. пользователь получает доступ ко всем своим файлам и нет возможности настройки разрешений и виртуальных пользователей, как, например в proftpd). По сути дела, подсистемы ssh могут обеспечивать прохождение других протоколов по сети, но под «крышком» ssh. Например, для sftp-сервера есть одноимённый sftp-клиент. Его интерфейс полностью идентичен оригинальному ftp, но с одним отличием: происходит та же самая аутентификация пользователя на удалённом сервере (методами ssh), но вместо оболочки с пользователем взаимодействует подсистема, в данном случае
  sftp. Subsystem sftp /usr/lib/ssh/sftp-server
```

Ну вот, вроде бы всё настроено! Теперь я бы хотел поговорить о некоторых фичах, работающих в ssh. Но с начала несколько слов о туннелях. SSH имеет встроенную возможность передавать данные с локального порта на удалённый, используя сетевой туннель, причём данные, передаваемые через туннель, будут шифроваться. То есть происходит аутентификация на удалённой системе, а затем начинается перенаправление трафика через туннель. Таким образом, можно перенаправлять любой трафик, а протокол иксов может работать в интерактивном режиме,

для этого необходимо включить соответствующие опции в файлах конфигурации сервера и клиента (это было описано ранее). Для других же портов необходимо вызывать ssh с параметром:

```
- L{LOCAL_PORT}:{LOCAL_ADDRESS}:
{REMOTE_PORT}:
# ssh -L10101:localhost:101
server.test.ru
```

Такой туннель довольно быстро умирает, т.к. сервер автоматически убивает «ленивых» клиентов. Поэтому можно применить метод, который позволяет устанавливать произвольное время удержания туннеля – выполнить sleep на удалённом сервере:

```
# ssh -f -L10101:
loclahost:101 server.test.ru sleep 100
```

Данная команда держит туннель 100 секунд, чего достаточно для любого соединения. И ещё одна вещь: когда по туннелю передаются данные, то он не уничтожается, что хорошо для реализации безопасного ftp smtp и pop3 протоколов (впрочем, sftp-сервер имеется уже и в поставке openssh, применение его не должно вызвать затруднений sftp [user@]hostname, т.к. фактически это особая реализация ssh протокола и механизм работы sftp абсолютно идентичен механизму ssh). Чтобы отключить перенаправление портов, необходимо установить опцию sshd AllowTcpForwarding в no. Использование длительной задержки ssh туннеля несколько уменьшает безопасность, т.к. во время ожидания злоумышленник имеет больше шансов на атаку (но механизм ssh версии 2 позволяет избежать подобной ситуации подписыванием передаваемых сообщений).

Вот что сделал бы я для безопасного ssh. Для начала создал бы rsa ключ длиной 4096 бит:

```
# ssh-keygen -t rsa -b 4096
```

Затем скопировал бы данный ключ с помощью дискеты или ssh-copy-id на удалённый сервер(а):

```
# ssh-copy-id -i $HOME/.ssh/id_rsa
remote_host
```

После этого запретил бы парольную и всякую hostbased аутентификацию в sshd_config:

```
IgnoreHosts                     yes
RhostsAuthentication            no
RhostsRSAAuthentication         no
RSAAuthentication               yes
HostbasedAutentification        no
PasswordAuthentication          no
PermitEmptyPasswords            no
UseLogin                        no

PermitRootLogin without-password
```

И отключим потокол версии 1 (по умолчанию – Protocol 2,1 и сервер падает к ssh 1, при неудаче – ssh 2):

```
Protocol 2
```

Ну вот теперь, чтобы зайти на сервак по ssh надо ввести нехилый (а он должен быть нехилый!) пароль секретного ключа. Немножко неудобно, не правда ли? Можно хранить пароль для секретного ключа в памяти на протяжении работы некоторой программы (например, bash) и при запросе его ssh клиентом доставать из памяти. Для этой цели служит программа ssh-agent (агент аутентификации ssh). Агент запускает необходимую программу и ждёт добавления новых секретных ключей (ssh-agent хранит расшифрованные секретные ключи). Для этой цели есть другая программа ssh-add, которая добавляет в агент ключи $HOME/.ssh/id_rsa, id_dsa, identity. Если необходимо добавить другие ключи, то надо запустить ssh-add с именем файла: ssh-add filename. Учтите, что при добавлении ключа в агент ssh-add вам всё равно необходимо ввести пароль для его расшифровки, но пока агент находится в памяти (пока вызванная им программа не завершилась), вводить пароль секретного ключа не надо: ключ берётся из ssh-agent. Причём контакт с ssh-agent может устанавливать только программа, запущенная им (и все её дочерние процессы), т.к. устанавливаются переменные окружения. Обычный метод запуска ssh-agent:

```
  # ssh-agent bash # ssh-add   Enter
passphrase for ".ssh/id_rsa":   Enter
passphrase for ".ssh/id_dsa":   .ssh/
identity : No such file or directory
```

После завершения работы оболочки ssh-agent завершается, а ключи удаляются из памяти.

Ну и наконец, можно разрешить доступ определённых пользователей с определённых машин. Это делается полем AllowUsers в sshd_config или правилами iptables. Думаю, этого достаточно для нормальной и безопасной работы на удалённом сервере через ssh. Особо мнительные могут запретить доступ рута по ssh (PermitRootLogin no) и делегировать часть его прав с помощью sudo. Ну и использовать протокол версии 2 (такие плюсы, как алгоритм вычисления симметрического ключа на основании пары асимметрических и подписывание сообщений, передаваемых по сети, а также 2 версия протокола ssh быстрее первой). Существует множество клиентов ssh, работающих на разных ОС:

Windows:

- putty: http://www.chiark.greenend.org.uk/~sgtatham/putty.html
- raju: ftp://ftp.franken.de/pub/win32/develop/gnuwin32/cygwin32/porters/mathur_raju
- cigaly: http://www.doc.ic.ac.uk/~ci2/ssh/
- f-secure: http://www.datafellows.com/f-secure/fclintp.htm
- secure crt: http://www.vandyke.com/products/securecrt/
- ttssh: http://www.zip.com.au/~roca/ttssh.html
- therapy: http://guardian.htu.tuwien.ac.at/therapy/ssh/
- chaffee: http://bmrc.berkeley.edu/people/chaffee/winntutil.html
- sergey okhapkin: http://www.lexa.ru/sos/
- fissh: http://www.massconfusion.com/ssh/

Mac:

- niftytelnet+ssh: http://www.lysator.liu.se/~jonasw/freeware.html
- f-secure: http://www.datafellows.com/f-secure/fclintp.htm

Хороший список ссылок по ssh находится на www.heimhardt.com, полезным мне показался также сайт www.openssh.com, ну и есть документация по ssh (правда версии 1) на сайте www.opennet.ru.