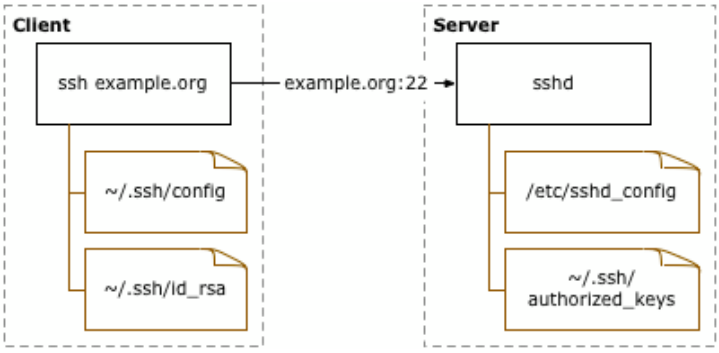
OpenSSH Public Key Authentication



Secure Shell (SSH) public key authentication can be used by a client to access servers, if properly configured. These notes describe how to configure OpenSSH for public key authentication, how to enable a `ssh-agent` to allow for passphrase-free logins, and tips on debugging problems with SSH connections. Password free logins benefit remote access and automation, for example if administering many servers or accessing version control software over SSH.

Public key authenticate can prevent brute force SSH attacks, but only if all password-based authentication methods are disabled. Other options to protect against brute force SSH attacks include `pam_tally`, or port knocking. Public key authentication does not work well with Kerberos or OpenAFS, which require a password or principal from the client.

Definition of terms used in this documentation:

- **Client**: the system one types directly on, such as a laptop or desktop system.
- **Server**: anything connected to from the client. This includes other servers accessed through the first server connected to.

Never allow root-to-root trust between systems. If required by poorly engineered legacy scripts, limit the `from` access of the public keys, and if possible only allow specific public keys to run specific commands. Instead, setup named accounts for users or roles, and grant as little `root` access as possible via `sudo`.

For more information, see also SSH, The Secure Shell: The Definitive Guide.

SSHKeyChain offers integration between the Apple Keychain and OpenSSH.

Public Key Setup

Key Generation | Key Distribution | Key Access Limits

First, confirm that OpenSSH is the SSH software installed on the client system. Key generation may vary under different implementations of SSH. The `ssh -V` command should print a line beginning with `OpenSSH`, followed by other details.

```
$ ssh -V
OpenSSH_3.6.1p1+CAN-2003-0693, SSH protocols 1.5/2.0, OpenSSL 0x0090702f
```

Key Generation

A `RSA` key pair must be generated on the client system. The public portion of this key pair will reside on the servers being connected to, while the private portion needs to remain on a secure local area of the client system, by default in `~/.ssh/id_rsa`. The key generation can be done with the `ssh-keygen(1)` utility.

```
client$ mkdir ~/.ssh
client$ chmod 700 ~/.ssh
client$ ssh-keygen -q -f ~/.ssh/id_rsa -t rsa
Enter passphrase (empty for no passphrase): …
Enter same passphrase again: …
```

Do not use your account password, nor an empty passphrase. The password should be at least 16 characters long, and not a simple sentence. One choice would be several lines to a song or poem, interspersed with punctuation and other non-letter characters. The `ssh-agent` setup notes below will reduce the number of times this passphrase will need to be used, so using a long passphrase is encouraged.

The file permissions should be locked down to prevent other users from being able to read the key pair data. OpenSSH may also refuse to support public key authentication if the file permissions are too open. These fixes should be done on all systems involved.

```
$ chmod go-w ~/
$ chmod 700 ~/.ssh
$ chmod go-rwx ~/.ssh/*
```

Key Distribution

The public portion of the RSA key pair must be copied to any servers that will be accessed by the client. The public key information to be copied should be located in the ~/.ssh/id_rsa.pub file on the client. Assuming that all of the servers use OpenSSH instead of a different SSH implementation, the public key data must be appended into the ~/.ssh/authorized_keys file on the servers.

```
# first, upload public key from client to server
client$ scp ~/.ssh/id_rsa.pub server.example.org:

# next, setup the public key on server
server$ mkdir ~/.ssh
server$ chmod 700 ~/.ssh
server$ cat ~/id_rsa.pub >> ~/.ssh/authorized_keys
server$ chmod 600 ~/.ssh/authorized_keys
server$ rm ~/id_rsa.pub
```

Be sure to append new public key data to the authorized_keys file, as multiple public keys may be in use. Each public key entry must be on a different line.

Many different things can prevent public key authentication from working, so be sure to confirm that public key connections to the server work properly. If the following test fails, consult the debugging notes.

```
client$ ssh -o PreferredAuthentications=publickey server.example.org
Enter passphrase for key '/…/.ssh/id_rsa': …
…
server$
```

Key distribution can be automated with module:authkey and CFEngine. This script maps public keys stored in a filesystem repository to specific accounts on various classes of systems, allowing a user key to be replicated to all systems the user has access to.

If exporting the public key to a different group or company, consider removing or changing the optional public key comment field to avoid exposing the default username and hostname.

Key Access Limits

As an optional step to limit usage of the public key for access to any servers, a `from` statement can be used before public key entries in the `~/.ssh/authorized_keys` file on the servers to limit where the client system is permitted to access the server from. Without a `from` limit, any client system with the appropriate private key data will be able to connect to the server from anywhere. If the keypair should only work when the client system is connecting from a host under `example.org`, set `from="*.example.org"` before the public key data.

```
server$ cat ~/.ssh/authorized_keys
from="*.example.org" ssh-rsa AAAAB3NzaC1…
```

If a text editor is used to add the `from` option, ensure the data is saved as a single line; some editors may wrap the public key and thus corrupt the data. Each public key in the `~/.ssh/authorized_keys` file must not span multiple lines.

Multiple hosts or addresses can be specified as comma separated values. For more information on the syntax of the `from` option, see the `sshd(8)` documentation.

```
from="*.example.org,10.*,external.example.com" …
```

Configure `ssh-agent` Process

To reduce the frequency with which the key passphrase must be typed in, setup a `ssh-agent(1)` daemon to hold the private portion of the `RSA` key pair for the duration of a session. There are several ways to run and manage `ssh-agent`, for example from a X11 login script or with a utility like Keychain. These notes rely on the setup of `ssh-agent` via an `@reboot` `crontab(5)` entry, along with appropriate shell configuration.

The `ssh-agent` must only be run on the client system. The private key of the `RSA` key pair must remain on the client system. Agent forwarding should be used to make the key available to subsequent logins to other servers from the first server connected to.

1. Startup cron job

The following crontab(5) entry should run the agent at system startup time. The crond daemon on BSD and Linux systems should support the special @reboot syntax required for this to work.

```
@reboot ssh-agent -s | grep -v echo > $HOME/.ssh-agent
```

To setup the agent for the first time without having to reboot the system, run the following.

```
$ nohup ssh-agent -s > ~/.ssh-agent
```

Once the ssh-agent is running, any shells already running will need to source in the environment settings from the ~/.ssh-agent file. The SSH_AUTH_SOCK and SSH_AGENT_PID environment variables set in this file are required for the OpenSSH commands such as ssh and ssh-add to communicate with the ssh-agent on the client system.

```
$ . ~/.ssh-agent
```

Notes on configuring all shells to be able to run arbitrary commands are available. This reduces the initial setup to the following commands, which can be done from the script reagent.

```
$ nohup ssh-agent -s | grep -v echo > ~/.ssh-agent
$ allsh - < ~/.ssh-agent
```

If csh or tcsh is being used instead of a Bourne-based shell, replace the -s argument with -c, and the source command used instead of . in any running shells.

2. Shell startup script changes

The shell's startup script on the client system will need to be modified to pull in the required environment settings from ~/.ssh-agent and setup useful aliases. The agent settings in ~/.ssh-agent should not be read in if the client system is being connected to as a server. Remote connections set the SSH_CLIENT

environment variable, so `~/.ssh-agent` must not be read in when this variable contains data.

```
[ -z "$SSH_CLIENT" ] && . $HOME/.ssh-agent

alias keyon="ssh-add -t 10800"
alias keyoff='ssh-add -D'
alias keylist='ssh-add -l'
```

The `-t` option to `ssh-add` will remove keys from memory after the specified number of seconds. This option prevents the keys from being left unlocked for long periods of time. Older versions of OpenSSH will not have the timeout `-t` option.

For the `csh` and `tcsh` shells, slightly different configuration of the agent and aliases is required. Consult the relevant `ssh-agent(1)` and shell documentation.

Once the `ssh-agent` is running and shell configured to read in the appropriate settings and set easy aliases, enable the key then test a login to a remote server. The `keyon` will only need to be run when initially adding the private key data to `ssh-agent`, and only rerun if `ssh-agent` is restarted or the key is removed with `keyoff`.

```
client$ keyon
…
client$ ssh server.example.org
server$ exit
client$ keyoff
```

Use the `keylist` command to see what keys are in the agent process.

```
$ keylist
1024 01:a1:aa:34:21:bc:7d:a4:ea:56:a4:a1:1a:c5:fa:9f /home/…/.ssh/id_rsa (RSA)
```

If password free logins do not work, see tips on debugging problems with SSH connections to work out where the problem may be.

To make other applications not run from a shell aware of the agent, the environment

definitions in the `~/.ssh-agent` file will need to be read into the software in question. Consult the documentation for the software to see whether this is possible.

Agent Forwarding

For simple client to server connections, SSH agent forwarding will not be a concern. However, if from the server connected to, one logs into other servers, SSH agent forwarding will need to be enabled. If SSH agent forwarding is disabled, a private key must be available on the proxy system that is recognized by the server being connected to.

To enable forwarding, either use the `-A` option to `ssh` when connecting, or set `ForwardAgent` in an OpenSSH `config` file, such as `~/.ssh/config`. Note that command line arguments override the user-specific configuration file, which in turn can override the global `ssh_config` configuration file, if any.

```
Host *
  ForwardAgent yes
  ForwardX11 no
```

Agent (and X11) forwarding may represent a security risk, providing more options to an attacker on a compromised server to work back to the client system. If paranoid, disable Agent and X11 forwarding by default, and only enable the features where needed. Also enable `StrictHostKeyChecking` and use configuration management software such as CFEngine to distribute a global `ssh_known_hosts` file to all client systems.

Questions or comments about this page? Current ruminations available on my blog.

$Id: index.xml,v 2.15 2009/04/25 05:52:36 jmates Exp $