



OpenSSH

Section 1: The basics

Johannes Franken
<jfranken@jfranken.de>

People think, that `ssh` is just some kind of `telnet` with encryption on top. On this page I show what makes OpenSSH 3.x better than `telnet`.

Contents

1. [What's OpenSSH?](#)
 - a) [How to get it](#)
 - b) [Components](#)
 - c) [Alternatives to OpenSSH](#)
2. [ssh protocols](#)
3. [Configuration](#)
 - a) [Client-configuration](#)
 - b) [Server-configuration](#)
4. [Login Sessions](#)
 - a) [Escape-commands](#)
 - b) [Starting interactive programs automatically](#)
5. [Encryption](#)
6. [Compression](#)
7. [Public key authentication](#)
 - a) [ssh-keygen](#)
 - b) [Using hostkeys](#)
 - i) [known_hosts](#)
 - ii) [ssh-keyscan](#)
 - c) [Using identity keys](#)
 - i) [authorized_keys](#)
 - ii) [ssh-copy-id](#)
 - d) [Logging in without a password](#)
 - i) [Empty passphrase](#)
 - ii) [ssh-agent, ssh-add](#)
8. [Further readings](#)

What's OpenSSH?

OpenSSH is an implementation of the ssh protocol suite. It's open source and continuously being developed further by the OpenSSH project team, see <http://www.openssh.org>

How to get it

OpenSSH has been compiled for any operating systems, and is shipped with most linux or BSD distributions.

An excellent windows port using the CygWin32 libraries is available at <http://www.networksimplicity.com/openssh/>

Components

The OpenSSH-distribution contains the following programs:

Name	Purpose
<code>ssh</code>	is the ssh-client. It initiates the connection to a ssh-server. See detailed description in the next chapters.
<code>sshd</code>	is the ssh-server. It accepts connections from ssh-clients. For details, see configuration notes
<code>ssh-keygen</code>	creates and converts keys. For details, see below .
<code>ssh-agent</code> , <code>ssh-add</code>	keeps the decyphered privatekey in memory, where clients can access it. For details, see below .
<code>ssh-keyscan</code>	displays the hostkey of a ssh-server. For example uses, see below .

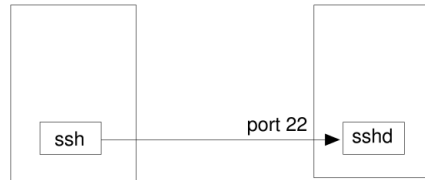
Alternatives to OpenSSH

Putty is free ssh client implementation, which - in contrast to the client that comes with OpenSSH - has got a graphical user interface. Get a version for Windows at <http://www.chiark.greenend.org.uk/~sgtatham/putty/> . Finally, putty is available for Linux, too.

Of course, there are some commercial suppliers as well, e.g. <http://www.ssh.com> und <http://www.f-secure.com> On top of the possibilities of OpenSSH they contain programs for central configuration of servers, keys and tunnels.

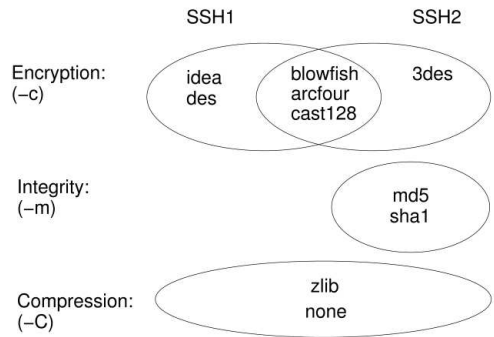
ssh protocols

For a typical usage of OpenSSH, there's usually one ssh-client (`/usr/bin/ssh`) calling one ssh-server (`/usr/sbin/sshd`), which is sitting on another host, listening to port 22. During their communication, very many different protocols will go to action.



```
jfranken@hamster:~ $ echo SSH-1.99-jfranken| nc -w 1 gate 22
SSH-1.99-OpenSSH_3.4p1Debian1:3.4p1-1\n[... ]
diffie-hellman-group-exchange-sha1,diffie-hell
man-group1-sha1\0\00017ssh-rsa,ssh-dss\0\0\0faes128-cbc
,3des-cbc,blowfish-cbc,cast128-cbc,arcfour,aes192-cbc,aes256-cbc
,rijndael-cbc@lysator.liu.se\0\0\0faes128-cbc,3des-cbc,blowfish-cbc
,cast128-cbc,arcfour,aes192-cbc,aes256-cbc,rijndael-cbc@lysator.
liu.se\0\0\0U hmac-md5,hmac-sha1,hmac-ripemd160,hmac-ripemd160@opens
sh.com,hmac-sha1-96,hmac-md5-96\0\0\0U hmac-md5,hmac-sha1,hmac-ripem
d160,hmac-ripemd160@openssh.com,hmac-sha1-96,hmac-md5-96\0\0\0\tnone
,zlib\0\0\0\tnone,zlib\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0\0
```

The list shows the algorithms covered by ssh protocol version 2. A subset was already contained in version 1, which should not be used any more for security reasons. OpenSSH 3 knows the ssh protocol versions 1 and 2.



Here are some details for selected algorithms:

- [draft-galb-secsh-gssapi-00.txt](#)
- [draft-galb-secsh-publickey-channel-00.txt](#)
- [draft-ietf-secsh-architecture-06.txt](#)
- [draft-ietf-secsh-auth-kbdinteract-01.txt](#)
- [draft-ietf-secsh-connect-08.txt](#)
- [draft-ietf-secsh-transport-08.txt](#)
- [draft-ietf-secsh-userauth-08.txt](#)
- [draft-nisse-secsh-srp-00.txt](#)
- [draft-salowey-secsh-kerbkeyex-00.txt](#)

Configuration

Client-configuration

The ssh client will read its configuration from

- Command line parameters,
- `~/.ssh/config` and
- `/etc/ssh/ssh_config`.

First hit matching. In the config files you can setup host regions, like this:

```
Host stunnel.our-isp.org
  ProxyCommand ~/.ssh/ssh-https-tunnel %h %p
  Port 443

Host hera 141.*
  User franken
  Protocol 1

Host 192.* gate mausi hamster tp
  Compression no
  Cipher blowfish
  Protocol 2

Host *
  ForwardAgent yes
  ForwardX11 yes
  Compression yes
  Protocol 2,1
  Cipher 3des
  EscapeChar ~
```

More about:

see [ssh\(1\)](#), [ssh_config\(5\)](#) manpages.

Server-configuration

The ssh-server reads its configuration from:

- command-line parameters and
- `/etc/ssh/sshd_config`

First hit matching.

More about:

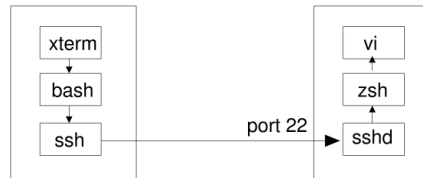
see [sshd\(8\)](#), [sshd_config\(5\)](#) manpages.

Login Sessions

By typing

```
ssh hostname
```

you can enter a shell on **hostname**.



The authentication is done for your local username.

If you want to login as a different user, there's a couple of ways to do that:

- `ssh -l username hostname`
- `ssh username@hostname`
- `ssh -o User=username hostname`
- `ssh hostname` having `User hostname` in your config file.

Escape-commands

During the ssh session you can call functions of the ssh-client by hitting **<Enter><Tilde>** and then one of the following keys:

.	abandon the session, breaking down any tunnels.
&	abandon the session, tunnels stay open.
C	Shows you a prompt (<code>ssh></code>), on which you can subsequently setup tunnels with the <code>-L</code> and <code>-R</code> commands (Details: see Teil 2).
<Ctrl-Z>	Suspend ssh. You are back in the shell from which you called ssh. Get back online with <code>fg</code> .
#	shows a list of connections that are tunneling on your session.
?	Help. Also shows the other options, which I left out here, because I didn't find them too useful.

If you're on a keyboard with deadkeys, you may need to press the tilde key twice for a tilde to pop up.

In case you are telescoping ssh sessions (one into another), you will get the n'th shell's escape mode (counting from the outside) by pressing **<Enter>** and then the tilde key n (deadkeys: 2n) times. People finding that too complicated can define different escape characters for each ssh, for example **<Ctrl-B>**:

```
jfranken@gate:~ $ ssh -e ^B hamster
jfranken@hamster:~ $ <Strg-B>.
Connection to hamster closed.
jfranken@gate:~ $
```

Starting interactive programs automatically

```
$ ssh tp -t vim /etc/hosts
```

Encryption

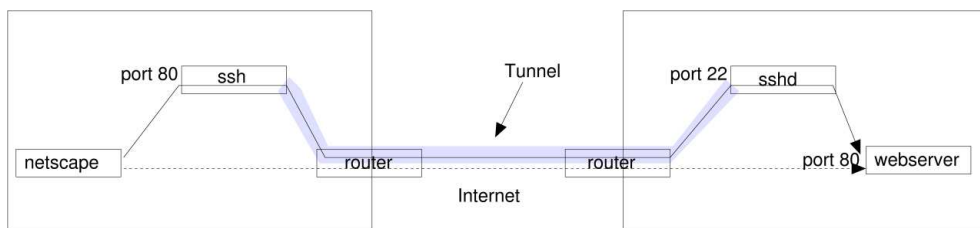
ssh will encrypt its full conversation with sshd, including authentication and any tunnels. You can select between different ciphers:

- **3des** is said to be particular secure, but needs a lot of CPU time.
- **blowfish** is quite fast

Here's some methods to use the **blowfish**-cipher in your local net, which relieves participating CPUs and thus speeds up communications:

- **ssh -c blowfish ...**
- **ssh -o Cipher=blowfish...**
- Put **Cipher blowfish** for some or any hosts in `~/.ssh/config` or `/etc/ssh/ssh_config`.

In contact to port-forwarding you can encrypt communications over unsecure network sections, being absolutely transparent for the client, which allows you to connect branches offices over the Internet:



Compression

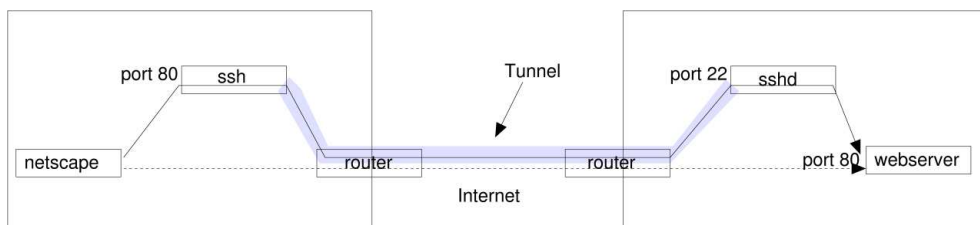
If your network turns out to be a bottleneck, you can have ssh compress any communications between ssh and sshd. At the cost of a little more CPU power on both sides, this saves up to 50% of all network packets.

To activate compression:

- **ssh -C ...**
- **ssh -o Compression=yes ...**
- Put **Compression yes** in `~/.ssh/config` or `/etc/ssh/ssh_config`.

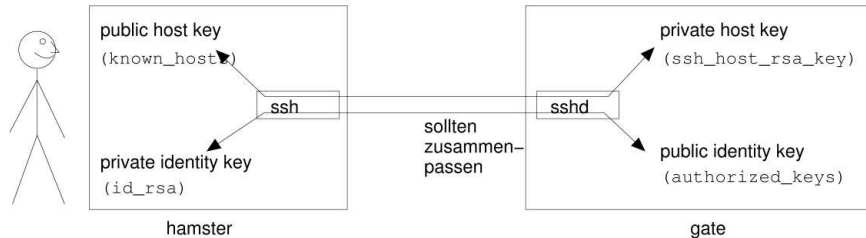
Additionally, you should have a line **Compression yes** in the server's `/etc/ssh/sshd_config`.

In contact to port-forwarding you can speed up communications over slow network sections, being absolutely transparent for the client:



Public key authentication

Public key authentication bases on the principle of asymmetric ciphers: The client and the server own one key that can encrypt messages in a way, that only the other partner can decrypt it. So if one keeps his key secret ("private key"), the other can assume authenticity, as soon as he can decompress a message using his own keys ("public key").



There are three different algorithms for this purpose:

algorithm	parameter to ssh-keygen	Filenames	publickey starting with
RSA for SSH version 2 and up	-t rsa	~/.ssh/id_rsa[.pub]	ssh-rsa
RSA for SSH version 1	-t rsa1	~/.ssh/identity[.pub]	1024 35 o.ä, (bits exponent)
DSA	-t dsa	~/.ssh/id_dsa[.pub]	ssh-dss

I'm not going into the specialities of each algorithm. Extract is, that DSA generates most CPU load while RSA1 is easiest to crack. Thus, in common cases RSA would be a reasonable tradeoff.

Every public key (e.g. ~/.ssh/id_rsa.pub) consists of the following parts:

- a type-definition (e.g. **ssh-rsa**, see above)
- space
- The modulus (that's the long string of letters and numbers)
- optional: one space, followed by a comment

It will be uuencoded to fit 6 bit ascii, which makes it look like a single, long line of text. This allows you to paste it into the text of an email.

```
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEAxtWFO8XbyT6+I1BAWYyOb
/VWraJ7iyMKVsb0TNmieBSzF6fgustkT0nX3udbSqTqiEC/wXFKqeyl27
bkd+rEcFba+s+wgV9MKRaiV0kOFVQrAvwrKnS1QI6YZWZIHSP7KS5QE0H
Rra+gy/47vGwHUn0RxksGOQ6YsKP5lNN8H3E= jfranken@hamster
```

ssh-keygen

Each authentication algorithm needs a separate key pair, which can be generated by the `ssh-keygen` command. Choose the algorithm, for which you are about to generate a pair of keys, and supply it to the `-t`-parameter.

```
$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/export/home/jfranken/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /export/home/jfranken/.ssh/id_rsa.
Your public key has been saved in /export/home/jfranken/.ssh/id_rsa.pub.
The key fingerprint is:
7c:02:29:1c:d4:8a:90:ad:f2:b0:65:9e:71:92:ef:0f jfranken@hamster
```

The fingerprint is a shared attribute of private- and publickey, and thus allows you to check which keys belong together.

```
$ ssh-keygen -l -f .ssh/id_rsa
1024 5a:b6:c4:50:ce:ec:18:78:e9:b2:e7:5b:04:c2:e4:c7 .ssh/id_rsa.pub
```

You can later change a private key's passphrase:

```
$ ssh-keygen -p -f ~/.ssh/id_rsa
Enter old passphrase:
Key has comment '/export/home/jfranken/.ssh/id_rsa'
Enter new passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved with the new passphrase.
```

If you happen to have lost your public key, you can generate a new one from your private key:

```
$ ssh-keygen -y -f ~/.ssh/id_rsa
ssh-rsa AAAAB3NzaC1yc2EAAAABIwAAAIEA5+UaqG/zI...
```

But remember, there's no way to regenerate a lost private key.

Using hostkeys

known_hosts

If you need to make sure, that your ssh connection goes to the right server, call ssh with `-o StrictHostkeyChecking=yes` or add `StrictHostkeyChecking yes` to your `~/.ssh/config` file for the hosts concerning.

The client will then stop connecting, if the server's secret hostkey does not match the public key, which you had stored on the client at `~/.ssh/known_hosts` or `/etc/ssh/ssh_known_hosts` before:

```
jfranken@hamster $ ssh -o StrictHostkeyChecking=yes gate
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@   WARNING: REMOTE HOST IDENTIFICATION HAS CHANGED!   @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
IT IS POSSIBLE THAT SOMEONE IS DOING SOMETHING NASTY!
Someone could be eavesdropping on you right now (man-in-the-middle attack)!
It is also possible that the RSA host key has just been changed.
The fingerprint for the RSA key sent by the remote host is
5c:6e:b2:99:3d:44:03:32:fb:e8:c1:ca:4f:cb:9e:8f.
Please contact your system administrator.
Add correct host key in /export/home/jfranken/.ssh/known_hosts to get rid of this message.
Offending key in /export/home/jfranken/.ssh/known_hosts:45
RSA host key for gate has changed and you have requested strict checking.
Host key verification failed.
jfranken@hamster $
```

or, if that host has never been entered to the `known_hosts` file:

```
jfranken@hamster $ ssh -o StrictHostkeyChecking=yes gate
No RSA host key is known for gate and you have requested strict checking.
Host key verification failed.
jfranken@hamster $
```

If you don't want to add all those hostkeys to your `known_hosts`-file by hand, but just be warned about changes, you can set `StrictHostkeyChecking` to `ask`:

```
jfranken@hamster $ ssh -o StrictHostkeyChecking=ask gate
The authenticity of host 'gate (192.168.42.1)' can't be established.
RSA key fingerprint is 5c:6e:b2:99:3d:44:03:32:fb:e8:c1:ca:4f:cb:9e:8f.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'gate,192.168.42.1' (RSA) to the list of known hosts.
jfranken@gate $
```

The lines of `known_hosts` are of the following format:

1. list any hostnames or ip addresses of a server, separated by comma, wildcard `*?` and negation `!` aware.
2. space
3. the server's public key (comment stripped)
4. optional: space, comment

sshd will look for the hostkeys at these places on the server:

- `/etc/ssh/ssh_host_dsa_key[.pub]` (DSA-Format)
- `/etc/ssh/ssh_host_rsa_key[.pub]` (RSA-Format f. SSH version 2+)
- `/etc/ssh/ssh_host_key[.pub]` (im RSA-Format f. SSH version 1)

You can generate these keys with the `ssh-keygen` tool. Just be sure to provide an empty passphrase to the privatekey. At most distributions this job is typically done by the install script of the sshd package.

When the hostkey of a server changes (for example after upgrading to a newer version of OpenSSH), you may need to remove the corresponding lines from your `known_hosts` files on the client.

More about:

see: [sshd\(8\) manpage](#)

ssh-keyscan

If you have got to enter quite a number of hostkeys to some `known_hosts` file, have `ssh-keyscan` do the job for you.

Because failures to read the hostkey point to network- or server-problems, you can run `ssh-keyscan` to detect such issues: if it does not display the hostkey, there is a problem.

More about:

see: [ssh-keyscan\(1\) manpage](#)

Using identity keys

authorized_keys

Alternative to entering a password you can use an keypair for authentication to a server. That private key can be protected by a passphrase again.

Advantages of public key- over password-authentication:

- It's much harder to crack keys on brute force as it is to try out passwords
- You got my public key? - Let me use it to access your server.
- One central passphrase to remember or change for every server I use.
- Leaving just my public key on the server is much safer than setting a personal password in `/etc/shadow`, which might get cracked.
- If a number of people share one account, each user can have a public key and passphrase of his/her own.

Using public keys for authentication is easy: XXX TODO

At the beginning of each line in `~/.ssh/authorized_keys` you can tell `sshd` from which hosts a public key can be used and what special options (environment settings, shell, etc) to use. The exact syntax is as follows:

1. An optional list of options, separated by comma, followed by a space:
 - a) `environment="SSHKEY=jfranken"`
(if you use `openssh ≥3.5p1`, you need to set `PermitUserEnvironment=yes` in `/etc/ssh/sshd_config` to allow this.)
 - b) `command="./menu.pl"`
 - c) `from="*.jfranken.de,egal.our-isp.org"`
 - d) `no-port-forwarding`
 - e) `no-X11-forwarding`
 - f) `permitopen="host:port"`
 - g) `no-agent-forwarding`
 - h) `no-pty`
2. the public key (see `*.pub` file), comment stripped.
3. optional: space comment

If the server option `PasswordAuthentication No` is set in `/etc/ssh/sshd_config`, the use of public keys becomes mandatory, which gives you pretty good security against brute-force attacks.

More about:
see: [sshd\(8\) manpage](#)

ssh-copy-id

If you regularly have to deposit your public key on servers, you will like `ssh-copy-id`, which is a little shell script doing all the work for you. Unfortunately it defaults to RSA1 (`~/.ssh/identity.pub`) public keys. If you're on RSA(2), append the parameter `-i ~/.ssh/id_rsa.pub` or use a [version](#) that was patched by me.

```
jfranken@hamster $ ./ssh-copy-id2 franken@hera.cs.uni-frankfurt.de
franken@hera.cs.uni-frankfurt.de's password:
Now try logging into the machine,
with "ssh 'franken@hera.cs.uni-frankfurt.de'", and check in:
  .ssh/authorized_keys
to make sure we haven't added extra keys that you weren't expecting.
jfranken@hamster $
```

More about:

see: [ssh-copy-id\(1\) manpage](#)

Logging in without a password

Empty passphrase

If your private key's passphrase is empty, and the corresponding public key is placed in a server's `~/.ssh/authorized_keys` file, you can greatly use ssh in automated scripts. But be very sure nobody gets hold of your private key file, especially if it's on a nfs/cifs fileservers.

ssh-agent, ssh-add

`ssh-agent` is a solution for those users, who are tired of always entering the same passphrase, but on the other hand don't want to abandon a passphrase for their private key for security reasons. In order to use that private key, a hacker would need access to the socket file on their local computer.

If you call `ssh-agent` without parameters, it will create an unix-domain-socket, tell you, how to reach it, and listen on it in the background.

```
$ ssh-agent > myagent.sh
$ cat myagent.sh
SSH_AUTH_SOCKET=/tmp/ssh-XXcMloq1/agent.21753; export SSH_AUTH_SOCKET;
SSH_AGENT_PID=21754; export SSH_AGENT_PID;
echo Agent pid 21754;
```

Use `ssh-add` every time you want to teach ssh-agent one of your private keys:

```
$ ./myagent.sh
Agent pid 21754;
$ ssh-add ~/.ssh/id_rsa
Enter passphrase for /export/home/jfranken/.ssh/id_rsa:
Identity added: /export/home/jfranken/.ssh/id_rsa (/export/home/jfranken/.ssh/id_rsa)
$
```

If you don't supply a private key, `ssh-add` will load the standard keys (`identity`, `id_rsa` and `id_dsa`).

Use `ssh-add -l` to get a list of any keys that that ssh-agent knows, or `ssh-add -L` to retrieve all corresponding public keys.

When connecting, the `ssh-client` will then try the private keys from `ssh-agent` first, instead of those from the identity files:

```
jfranken@hamster $ ./myagent.sh
Agent pid 21754;
jfranken@hamster $ ssh gate
[...]
debug1: userauth_pubkey_agent: testing agent key /export/home/jfranken/.ssh/id_rsa
debug1: input_userauth_pk_ok: pkalg ssh-rsa blen 149 lastkey 0x80926f8 hint -1
[...]
jfranken@gate $
```

Alternative to using the `myagent.sh` you can use `lsof` to find the values for the environment variables:

```
$ /usr/sbin/lsof -a -u jfranken -U -c ssh-agent
COMMAND  PID    USER  FD  TYPE   DEVICE  SIZE  NODE  NAME
ssh-agent 477  jfranken   3u  unix 0xc1dalaa0      1155 /tmp/ssh-XXWzoql0/agent.452
$ export SSH_AUTH_SOCKET=/tmp/ssh-XXWzoql0/agent.452 SSH_AGENT_PID=477
```

If you want the variables to be set automatically to the running ssh-agent at each login, you can add the program [keychain](#) to your `~/.bash_profile`.

More about:

see: [ssh-agent\(1\) manpage](#)
[ssh-add\(1\) manpage](#)

Further readings

- [Part 2: SSH-tunnelling](#)
- [Part 3: Breaking firewalls](#)
- [The Secure Shell TM Frequently Asked Questions](#)
- [OpenSSH Project Page](#)