

SSH Key Exchange: Windows client to Unix/Linux server

Table of contents

1 Audience.....	2
2 Purpose.....	2
3 Prerequisites.....	2
4 SSH Key exchange: Windows client & Unix/Linux Server.....	3
4.1 Outline of steps (please start here!!).....	3
4.2 Overview: About Public Key Authentication.....	3
4.3 Testing whether SSH Keys have already been exchanged.....	4
4.4 Determine whether you've already generated SSH keys.....	5
4.5 Generate SSH Keys (if necessary).....	6
4.6 Create an identification file (if new keys were generated).....	8
4.7 Make public key available on the remote host.....	8
4.8 Prepare home directory on the server.....	8
4.9 Copy user's public key to server.....	9
4.10 Transform the public key and add it to the authorized_keys file.....	10
4.11 Testing and Troubleshooting.....	10

1. Audience

Windows users who wish to exchange [SSH Secure Shell](#) keys with one or more Unix or Linux servers. While this HowTo is oriented to a Windows-based SSH client site-licensed by the UC Berkeley campus, the general steps and concepts are the same across clients and operating systems; however, *caveat emptor*: with respect to key exchange, the devil is in the details.

2. Purpose

To implement public-key login onto one or more Unix or Linux servers from a Windows box (e.g., to allow password-free CVS or Subversion access).

For public-key login to work, the server's SSH public key must reside in the appropriate directory on the workstation from which login is attempted; while the user's SSH public key resides in the appropriate place on the server. These instructions explain how to tell if this is already done for the workstation-server pair of concern; and, if not, how to effect appropriate exchange of public keys.

3. Prerequisites

These are the basics; if your machine isn't running these, please install or upgrade as appropriate:

- **Operating System:** Upgrade to latest **stable**, high-encryption releases of [Windows](#)
- **SSH Secure Shell:** Install on your box; if you already have it installed, be sure the version is recent. For UCB community members, the [currently licensed version of SSH for download](#) is maintained at the [Software Central](#) site. **Note:** These instructions presume that the installation directory for SSH is `C:\Program Files\SSH Communications Security\SSH Secure Shell\`. If your installation directory is different, translate these instructions accordingly.

Please note: It is important throughout this procedure to use consistent references to the server:

- It may seem obvious, but `myserver` is not the same as `myserver.berkeley.edu`; ditto for the client machine (i.e., the developer desktop), `mybox` and `mybox.berkeley.edu`
- If a key is generated using the fully-qualified name, and an unqualified name is used in a login attempt, SSH will not authenticate properly and the login will fail.
- Note that in cases where this sort of mistake is made, the client's error message and/or log file may not be very helpful in tracking down what's really wrong...

4. SSH Key exchange: Windows client & Unix/Linux Server

4.1. Outline of steps (please start here!!)

Success in exchanging keys - particularly for those who are creating directories and converting keys on the remote host - is dependent on careful attention to detail; failure is often the result of inattention to permissions in the user's home directory tree on the server to which public-key login is desired.

Follow the steps below, noting that some of the steps only pertain to servers that allow shell access, such as `canvas.berkeley.edu`. Note that `svn.berkeley.edu` does NOT allow shell access to users of UC Berkeley's Subversion Repository Service.

- [Overview](#) (optional)
- [Testing whether SSH Keys have already been exchanged](#)
- On the local machine (e.g., developer's desktop):
 - [Determine whether you've already generated SSH keys](#)
 - [Generate SSH Keys](#) (if necessary)
 - [Create identification file](#) (if new keys created)
- [Make public key available on the remote host](#)
- On the remote host (ONLY if shell access is allowed on the server):
 - [Prepare home directory on the server](#)
 - [Copy user's public key to server](#)
 - [Transform the public key and add it to the authorized keys file](#)
- [Testing and Troubleshooting](#)

Note:

On the Berkeley campus, some hosts (servers) do not allow shell login. For these hosts, server-side tasks are performed by a system administrator, not by the user. Where the procedure differs between hosts that allow shell access and those that do not, this HowTo is clearly marked. Following the list above will help users navigate the correct procedure for a given server.

4.2. Overview: About Public Key Authentication

This section is included to provide context for those new to public-key authentication concepts. It is not necessary to read this section, but doing so may make it easier to understand the step-by-step instructions that follow.

Authentication using a public key relies on cryptographic technology that allows generation of a public-private key *pair* -- a public key and private key -- that are uniquely matched. For any given public key, only one possible private key can be its other-half.

The central idea of public key authentication is this: If I have a public key that I believe to belong to Alice, then I will believe that any entity who can prove she has access to the corresponding private key is Alice. That's it in a nutshell. All the rest are implementation details: how software negotiates a determination of whether a requesting entity has a private key that corresponds to a public key whose origin/ownership is presumed to be known and trusted.

A public key that has been accepted by someone (that someone could be you yourself, logged into an account on a remote machine) implies trust that the entity who provided the public key is who she claims. A private key held in a location inaccessible to anyone but its owner constitutes proof that an entity who has access to it is the same entity who distributed the corresponding public key.

The public key is not intended to be kept secret. But it is essential to the integrity of public-key authentication that its corresponding private key is kept *absolutely* secret: if someone has your private key, s/he can impersonate you (see previous paragraph). The integrity of a private key is usually assured by storing it in a directory to which access is strictly limited. On a Windows machine, this secure directory is usually located in the C:\Documents and Settings\myuserid directory (substitute your Windows login for myuserid). A private key may be further secured by the requirement that it cannot be used except when the user supplies a passphrase; this additional security is in play so long as the passphrase is known only to the legitimate owner of the private key, and cannot be guessed by a person or program.

This HowTo explains how to use software licensed by UC Berkeley to generate a public-private key pair, then to properly covert and store the public key on a server to which one wishes to log-in using public key authentication.

Warning:

Do not EVER leave a private key in a location that can be read by another person / machine / computer user! Always keep your private key private! Allowing others to read the file containing your private key is like publishing your password in a newspaper. Once done, the key pair is compromised and a new key pair *must* be generated to replace the compromised pair. Even if a private key is passphrase-protected, a key-pair whose private key is published *must* be treated as compromised, and must be replaced post haste...

4.3. Testing whether SSH Keys have already been exchanged

4.3.1. Testing with host that allows shell access

If Host Keys have already been exchanged *with a server on which shell access is permitted*, you'll be able to open an SSH Secure Shell Client window logged in to the server of interest using Public Key authentication (as opposed to Password authentication). Note that the UC

Berkeley Subversion server, `svn.berkeley.edu`, does NOT allow shell access (see next sub-section).

- Open SSH Secure Shell client terminal window
- Request a new connection
- Fill out the server name (remember the name-consistency warnings, above!), your User Name, the Port Number (generally 22 for an SSH connection), and choose "Public Key" from the list of Authentication Methods.
- Click the `Connect` button.
- If this is the first time you've attempted to use SSH to log on to this server, you'll be asked whether you want to accept and store the server's Host Key. You want to do this.
- If you see a prompt on your server, Host Keys have been correctly exchanged. You're done!

4.3.2. Testing with host that does NOT allow shell access

If you are not allowed shell access on a host (server), it's most likely that a system administrator stored a public key you supplied in the proper location on that machine. In this case, you must attempt to perform the activity that your public-key login is supposed to permit. If you can perform the activity, and are sure you would not be able to perform the activity unless public-key access were properly set up, your public-key authentication is working. You're done!

In the context for which this HowTo was published, the most likely scenario is that you are attempting to establish public-key login with UC Berkeley's Subversion server, `svn.berkeley.edu`. A quick way to test that public-key authentication is working is to attempt to log in using your SSH client (SSH Secure Shell) in public-key authentication mode. If you do so, and:

- ...see a message like the one displayed in the screenshot below, your public-key authentication is working - you're done!
- ...you get a message similar to: "No further authentication methods available" then your public-key access is not set up properly. *If you believe the key was already placed in the appropriate place by a system administrator, see [the last part of the troubleshooting section of this document](#), below.*

success message on disallowed-shell-login attempt

4.4. Determine whether you've already generated SSH keys

If you've already generated a pair of SSH keys (one public, the other private), you need not generate another pair. The key pair identifies you on your workstation. It can identify you to multiple servers to which you want to connect, just like your single driver's license can

identify you to multiple merchants, or your single passport can identify you to customs agents all over the world.

To determine whether you've already generated a pair of SSH keys:

- Open Windows Explorer.
- If you can't already see "hidden" files and folders, navigate to `Tools:Folder Options:View` and:
 - Assure that `Show hidden files and folders` is selected
 - Assure that `Hide file extensions for known file types` is **not** selected
 - Assure that `Hide protected operating system files (Recommended)` is **not** selected
 - Click the `Like Current Folder` button at the top of the pane.
- Look in the directory `C:\Documents and Settings\mylogin\Application Data\SSH\UserKeys\` (substituting your own login name for "mylogin").
- The public and private keys will be files that may be named something like this: `id_dsa_1024_a` and `id_dsa_1024_a.pub`; or you may have given them meaningful names, such as `aliceatwork` and `aliceatwork.pub`.
- If the files already exist, you need not generate them again. Skip ahead to the section titled [Prepare home directory on the server](#). Otherwise, generate SSH keys as explained in the following section.

Note:

The "public" key of the pair is the one with the suffix ".pub". Only share the public key. Never allow another person to read / see / copy your private key!

4.5. Generate SSH Keys (if necessary)

You're here because you haven't generated SSH keys yet (you determined this by following steps in the preceding section).

SSH key generation dialog

- Generate a pair of keys (one public, one private) in one of these ways:
 - by typing the following from a Command prompt:
`C:\Program Files\SSH Communications Security\SSH Secure Shell\ssh-keygen2` and answering the prompts presented; or,
 - by:
 - opening an SSH Secure Shell client (`SshClient.exe`),
 - navigating to the Settings dialog (`Edit : Settings`)

- navigating the settings "tree" to open Global Settings : User Authentication : Keys
- clicking the [Generate New] button under Key Pair Management, following instructions given in the Key Generation Wizard
 - **Should you give your key a password???** Please consider carefully the note below to determine whether it makes sense to give your key a password / passphrase. *If your key has a passphrase, you will be required to supply it each time the key is used for authentication. Depending on how the software using the key works, this can be a quick, once-per-day event, or it could be a constant annoyance, rendering a significant part of the client-side advantage of using key-based authentication moot. Whether or not your key is protected by a passphrase, permissions on the directory where your keys are generated - and where the private key will be stored - MUST be highly restricted ... e.g., **only** you should have permission to read files stored in that directory.*
 - **You can name your key whatever you wish.** The default is not especially informative, so you might want to use a name that conveys WHO generated the keypair and ON WHAT MACHINE, e.g., aliceatwork.

=====

Note on passwords / passphrases on private keys

=====

If your key will *only* be used to access a Subversion or CVS repository from within the Eclipse IDE, it may be best NOT to give your key a passphrase, because it doesn't offer much extra protection (see Background, below).

If you are using your key for other purposes (instead of or in addition to usage through Eclipse and its plug-ins), it probably makes sense to use a passphrase. Keys that authenticate to accounts with privileged access on a host, or with access to sensitive information or [restricted data](#), should always be protected with a strong passphrase.

Background: If you are using the Eclipse IDE, you will likely decide to allow Eclipse to cache (remember) your passphrase. This is not a good idea, because Eclipse encrypts cached passwords and passphrases weakly, so that a malicious user (hacker) could discover them; what's worse, a malicious user could use a file stolen from your Eclipse directory in conjunction with your stolen private key to impersonate you from within another Eclipse installation, without even bothering to hack the weakly-encrypted file. Because it's convenient to let Eclipse cache your passphrase, most people will end up doing so even if they start out determined not to. For this reason, it's probably better to operate without the false sense of additional protection afforded by a passphrase on your private key.

In any case, if your private key is ever compromised, you *must immediately invalidate it* (remove the corresponding public key from the `authorized_keys` files to which it has been concatenated on any and all hosts (servers)).

=====

- The files described above should now exist in your `C:\Documents and Settings\mylogin\Application Data\SSH\UserKeys` folder.

4.6. Create an identification file (if new keys were generated)

The `identification` file tells the SSH client software which private key in your `UserKeys` directory is the one that proves your identity.

- In the same directory where your key files reside, create a text file named `identification` (note that there's no extension), if it doesn't exist already.
- The file should contain a single line, indicating the name of your private key (substitute your own private key name if it differs from the example):

```
idKey id_dsa_1024_a
```

4.7. Make public key available on the remote host

Depending on whether the host you want to log in on allows shell logins or not, follow one of these two paths:

Host does **NOT** allow shell access (e.g., `svn.berkeley.edu`)

- Make your public key available to the remote host's system administrator, or to the person responsible for coordinating your group's request with the service-provider / host administrator.
- If the remote host is `svn.berkeley.edu`, please communicate with IST (the service provider) through the individual in your group who authorizes changes to your group's repository (or repositories).
- After you have been notified that your public key has been associated with your account on the remote host, proceed to [Testing and Troubleshooting](#), below.

Host allows shell access

Continue with the next section, [Prepare home directory on the server](#)

4.8. Prepare home directory on the server

Note:

It is only possible for a user to perform this step if the remote host allows shell access. Otherwise, see [Make public key available on the remote host](#), above.

In these steps, the user's home directory is prepared by setting the correct permissions and creating a directory, if it doesn't exist already, for the SSH keys. It is essential to attend carefully to [permissions](#) in server directories! Incorrect permissions may cause authentication attempts to fail.

- Log into the server of interest, using the SSH Secure Shell client to open a terminal session. You'll use password authentication, since public-key authentication isn't implemented yet...
- Normally, when you first log in, your current directory will be your home directory. The Unix command `pwd` shows which directory you're in

```
$ pwd
/home/staff/mylogin
```

- If the [permissions](#) on your home directory are different from 750 (`drwxr-x---`), change them using the `chmod` command, e.g.:

```
chmod 750 /home/staff/mylogin
```

- From your home directory on the server, create a directory `.ssh` if it doesn't already exist, using `mkdir`:

```
mkdir .ssh
```

- From your home directory on the server, set [permissions](#) on the new directory so that only you have [permissions](#) to read, write, or execute in the `.ssh` directory:

```
chmod 700 .ssh
```

4.9. Copy user's public key to server

Note:

It is only possible for a user to perform this step if the remote host allows shell access. Otherwise, see [Make public key available on the remote host](#), above.

In these steps, the user's public key (the one with the `.pub` extension) is copied, using a secure file transfer protocol (SFTP), to the server. (Note: depending on your version of the SSH client and your operating system, you may need to set the folder attribute of `C:\Documents and Settings\[YourWorkstationUserId]\Application Data` so that it is not Hidden in order to view it using the SSH Secure File Transfer Client.)

- From your logged-in SSH Secure Shell terminal window, open a new File Transfer window (from the menu, select `Window : New File Transfer`)
- On the right side of the File Transfer window you should see your home directory on the server. You can upload to the home directory: that's easiest as it requires no navigation on your part, and that's how these instructions are written.
- In the File Transfer window's menu, choose `Operation : Upload Dialog`
- Navigate to the folder on your local computer where your SSH keys reside (usually `C:\Documents and Settings\mylogin\Application Data\SSH\UserKeys`)
- Select the **public** key (the one with the `.pub` extension) and click the `Upload` button

4.10. Transform the public key and add it to the `authorized_keys` file

Note:

It is only possible for a user to perform this step if the remote host allows shell access. Otherwise, see [Make public key available on the remote host](#), above.

Now that the public key from your local computer is copied to your home directory on the remote server, the next step involves transforming it from the key format supported by the client machine (`ssh.com` format if you are using the software described in this HowTo) to that used by the server's implementation of SSH; in the same step, you'll add (concatenate) the key to the set that identifies users authorized to log into your account using public-key authentication.

- From a terminal shell in your home directory on the server, run the following command (substituting the name of your public key if it is different from `id_dsa_1024_a.pub`):

```
ssh-keygen -i -f id_dsa_1024_a.pub >> .ssh/authorized_keys
```

This command converts the `ssh.com`-protocol public key to the protocol used by the remote server, then concatenates it to the `authorized_keys` file (this command will create `authorized_keys` if it doesn't exist...)

- The `.ssh/authorized_keys` file now contains a converted copy of your public key. Erase the copy of the Windows-format public key you placed into your home directory. You can use the Unix command `rm`, or use an SSH Secure Shell File Transfer window if you prefer a GUI client to perform this task.
- From your home directory, set [permissions](#) on the `authorized_keys` file

```
chmod 644 .ssh/authorized_keys
```

4.11. Testing and Troubleshooting

Try again to access the remote host using public-key authentication, as described in [Testing whether SSH Keys have already been exchanged](#), above.

If you still can't access the remote host, follow the troubleshooting path that pertains to your circumstance, below:

Host does **NOT** allow shell access

Go directly to [Troubleshooting For All Hosts](#)

Host allows shell access

Assuming you are able to access the remote host via password (as you probably did to set up public-key access in the above steps), but are unable to log in via the public-key, the most likely culprit is permissions settings on the server. They should be as follows (use `ls -la` to view file permissions on the contents of the directory in which the command is executed):

- Home Directory: `drwxr-x---` (750)
- `.ssh` Directory: `drwx-----` (700)
- `authorized_keys` file: `-rw-r--r--` (644)

Troubleshooting For All Hosts

If you're still having trouble, whether your remote host allows shell access or not, it is possible that a mistake was made in following one or more of the steps described above. Some things to look for:

- You should have added your **public** (not private) key to the `authorized_keys` file (or forwarded it so that a system administrator could perform this task on your behalf); cf. [Make public key available on the remote host](#) or [Move user's public key to server](#) (depending on whether shell access is prohibited or allowed on the host to which you are trying to connect) for more information. *Be sure you did not add (or forward) the private key - if you did, be sure to erase (or request erasure of) the private key from the remote server immediately: both the file you uploaded, and the `authorized_keys` file (or the portion of `authorized_keys` that contains the private key). In general, if you have exposed your private key to others, you should ALWAYS inactivate the key-pair in question and generate a new set of keys.*
- The public key you added to `authorized_keys` (or forwarded so that a system administrator could perform this task on your behalf) must be the "mate" of the private key that is:
 - stored in the location expected by your SSH client (e.g., `C:\Documents and Settings\mylogin\Application Data\SSH\UserKeys\` - substituting your own login name for "mylogin")

- named in the `identification` file typically located in the same directory as the private key itself (cf. [Create an identification file](#), above).

Note:

If you have carefully followed the steps in this HowTo, and after working through the troubleshooting sections are still unable to resolve your problem, please contact the IST Service Desk.