How to secure SSH on Ubuntu
Revised: 15-August-2016 by David Walling

This "How To" document describes configuration changes useful for securing an Ubuntu 16.04 instance. This document builds on earlier "How To" documents that describe installing Ubuntu 16.04 from an OSBoxes VM (VDI) into an Oracle VirtualBox emulation environment.

By default, the Ubuntu 16.04 guest OS image provided by OSBoxes includes a single logon, osboxes, with both SSH and sudo permissions. We're going to start by creating a new user account that has minmal privileges on the guest OS host but which is authorized to use SSH.

We're also going to edit the sshd_config file to use a non-standard port, allow only SSH-2, limit SSH to a single network interface, restrict root access, require public-key authentication, set restrictive login-completion times and other changes designed to make the use of SSH more secure.

Logon to Ubuntu and open a Terminal session. Create a new user with the useradd command. In our case, I am creating the user "dwalling", following a first initial and last name convention. Choose an appropriate user name for your environment. Then, use the passwd command to set an initial password for the new user. Use password conventions suitable for your environmnet. Avoid common words, names and birth dates. Include capital and lower case letters, one or more numbers and special characters. The password should be at least eight characters in length and should be something you can remember without writing it down.

```
osboxes@osboxes:~$ sudo useradd -m -d /home/dwalling -g users dwalling
[sudo] password for osboxes:
osboxes@osboxes:~$ sudo passwd dwalling
Enter new UNIX password:
Retype new UNIX password:
passwd: password updated successfully
osboxes@osboxes:~$
```

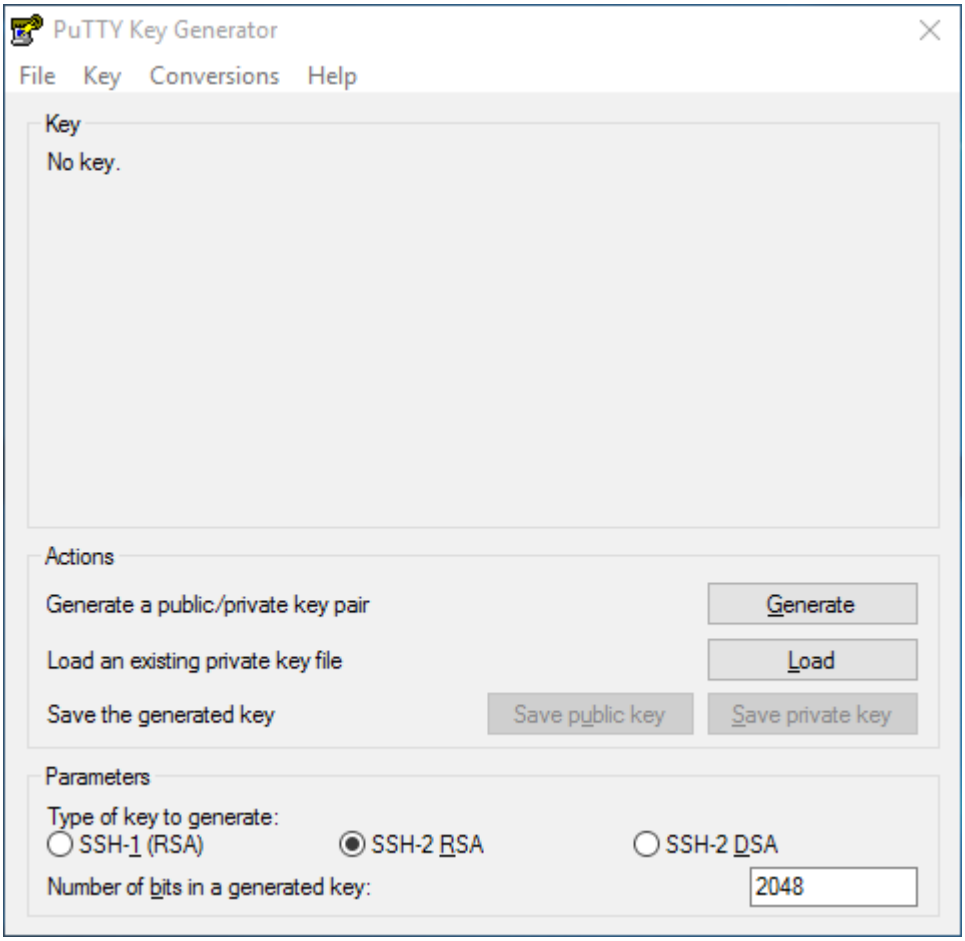
The ls and id commands can show the contents of the new user's home directory and the groups to which the new user belongs.

```
osboxes@osboxes:~$ ls -al /home/dwalling
total 32
drwxr-xr-x 2 dwalling users 4096 Jul 29 15:31 .
drwxr-xr-x 4 root     root  4096 Jul 29 15:31 ..
-rw-r--r-- 1 dwalling users  220 Aug 31  2015 .bash_logout
-rw-r--r-- 1 dwalling users 3771 Aug 31  2015 .bashrc
-rw-r--r-- 1 dwalling users 8980 Oct  4  2013 examples.desktop
-rw-r--r-- 1 dwalling users  655 Jun 24 10:44 .profile
osboxes@osboxes:~$ id dwalling
uid=1001(dwalling) gid=100(users) groups=100(users)
osboxes@osboxes:~$ id -Gn dwalling
users
osboxes@osboxes:~$
```

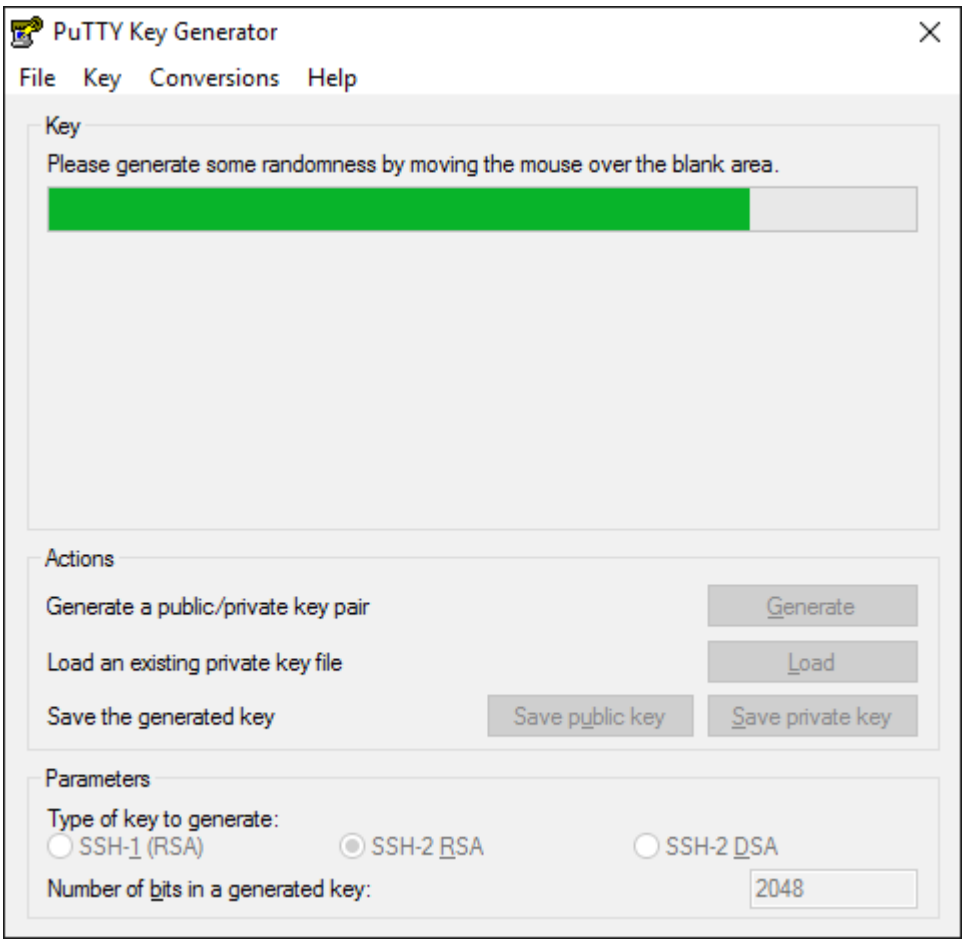
We now want to create an RSA key pair that will be used to secure the SSH connection for our new user. Here is where some basic conceptual understanding is important.

First, we create the key pair on the client that will be used to initiate the connection to our Ubuntu server. In our case, this is on a Windows 10 host. When the key pair is created, both a private and a public key will be generated. The private key should never leave the computer on which it is created. Its purpose is to validate the client to the Ubuntu server. Copying a private key will undermine this purpose and increase the risk that it will be compromised. After the key pair is created, the public key will be uploaded to the Ubuntu server and stored in a subfolder for the user. During SSH authentication, the Ubuntu server will challenge the client to prove that it possesses the private key.

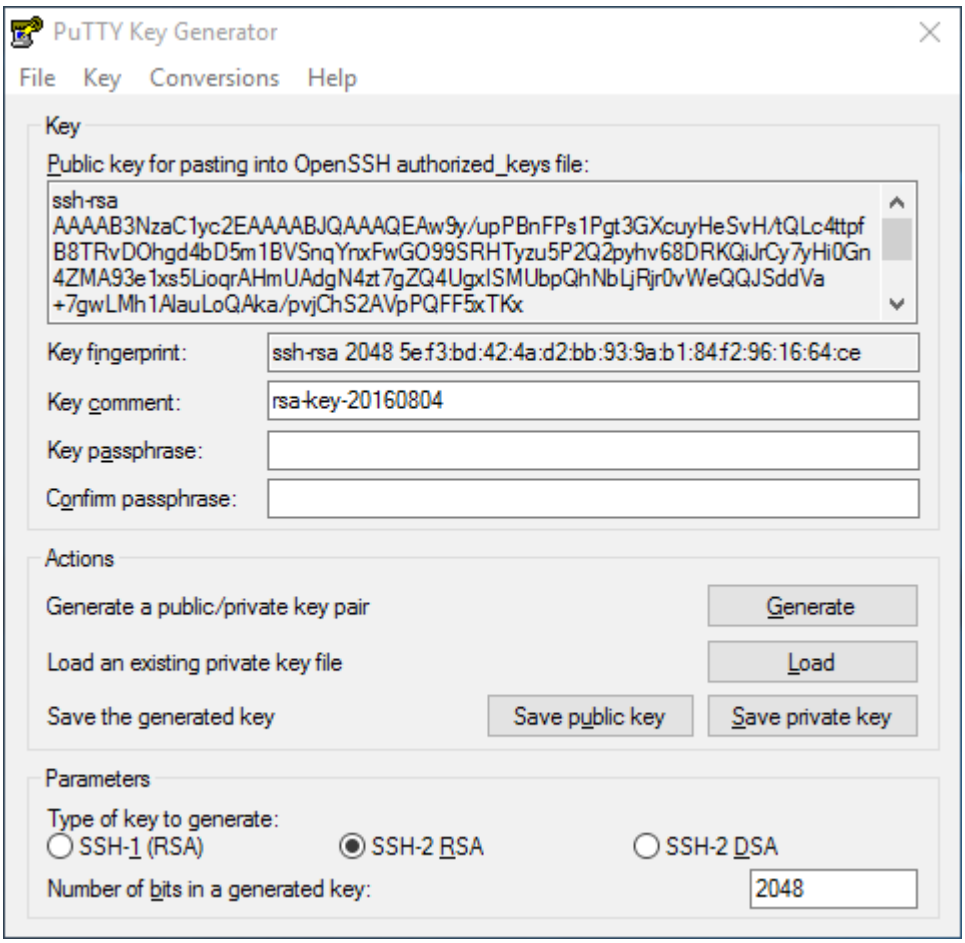
For our example, our client is a Windows 10 host. We are using puttygen.exe to create the key-pair. Entering puttygen.exe on the command line or clicking on an icon for the program opens the dialog shown below. By default, the type of key is set to SSH-2 RSA. If not, select this type. The "Number of bits in a generated key" should be 2048.



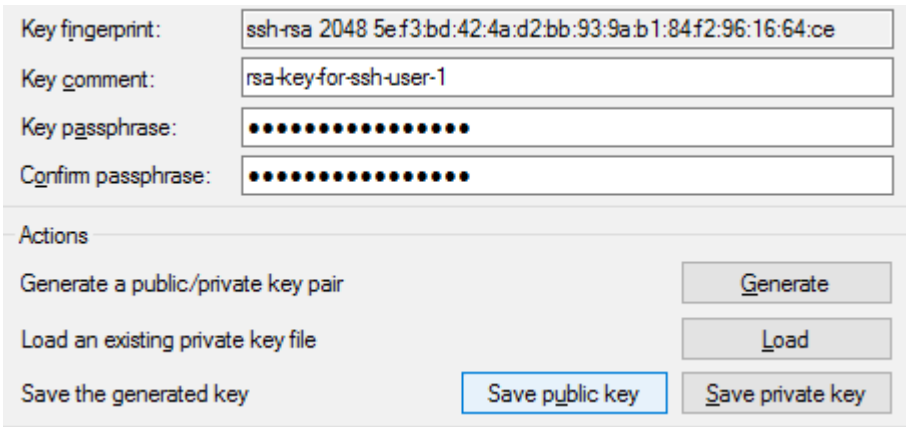
Clicking the "Generate" button starts the key generation process. In order to complete the process, the user must move the mouse around the dialog box to generate some pseudo-random bits.



After the key pair is generated, we need to add additional data in the dialog that appears. First, change the "Key comment:" field value so the date the key was created is no longer part of the comment value. This comment will appear whenever the SSH user connects to the server. We do not want to advertise the date our key pair was created.

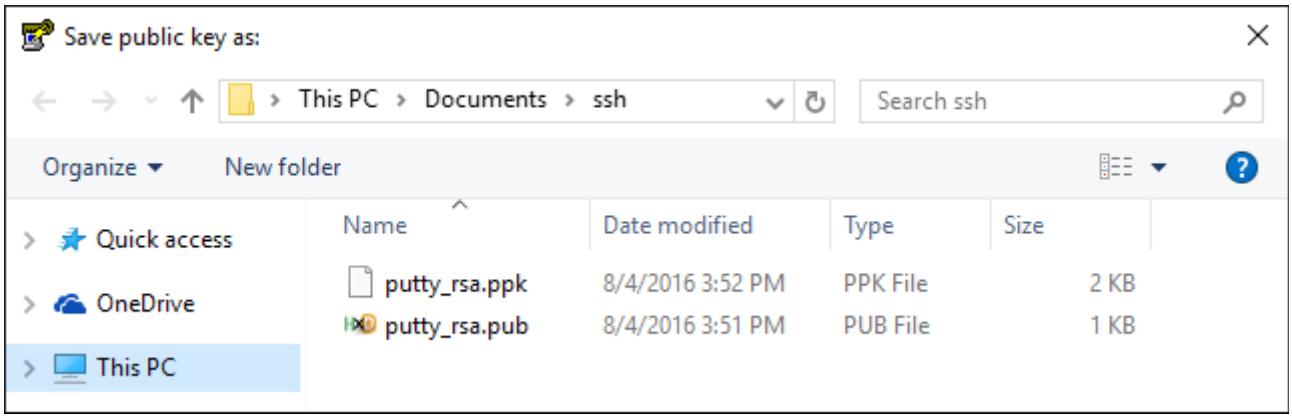


Update the comment with enough information for you to know which key is being used, but include no information that would identify the key to others. Also, enter a passphrase in both the "Key passphrase" and "Confirm passphrase" fields. This is usually longer than a typical password. It can include punctation and spaces. Again, try to use a passphrase you will remember without having to write it down. If you are forgetful and just need to write down passwords and passphrases, consider writing them down in a small notebook you can keep with you or lock in your desk drawer.

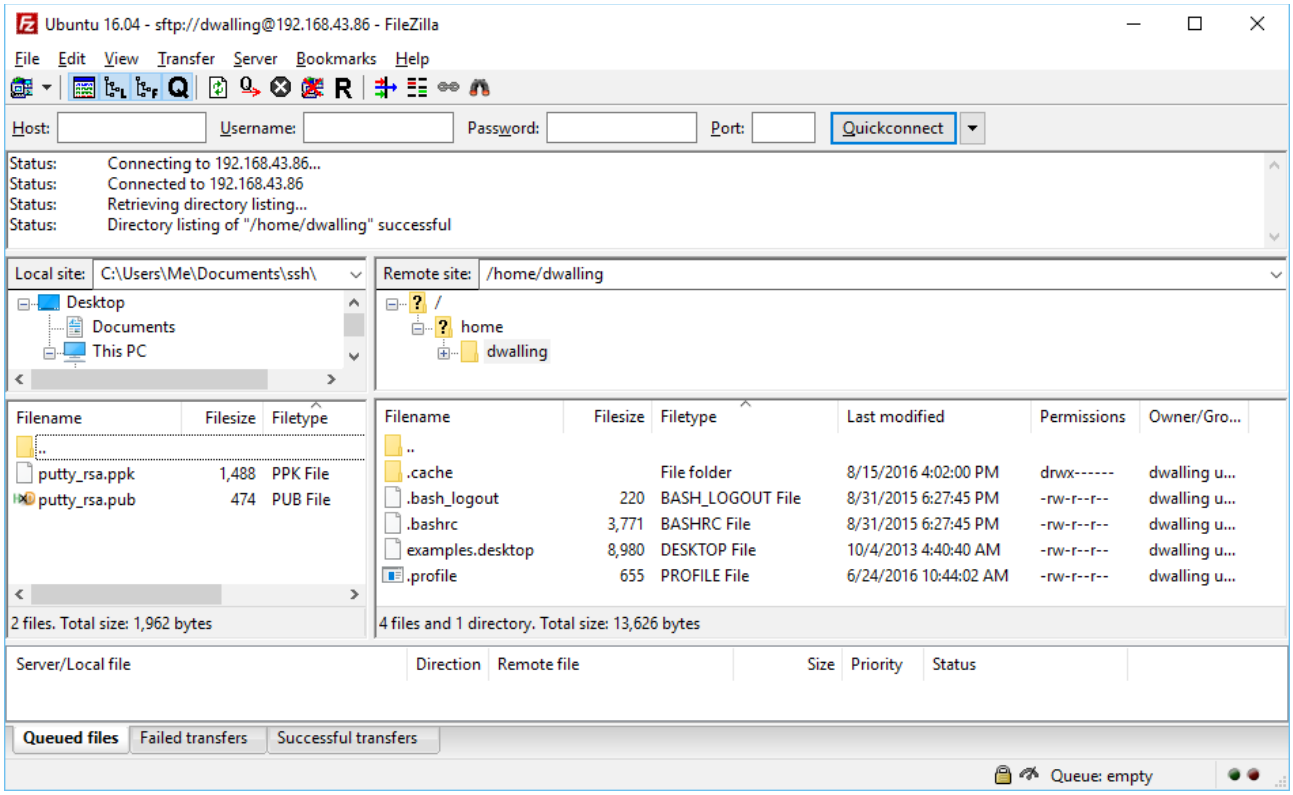


Don't confuse the public key file created here with a digital certificate. They are not the same! One of the biggest differences is that digital certificates expire. The key pair you have just created has NO expiration date. You are responsible for managing the life-cycle of the key pair. There is also no concept of following an SSH public key's signer to a trusted root issued by any third-party. Authenticating using an SSH key pair only proves that you (1) possess the private key and (2) know the passphrase.

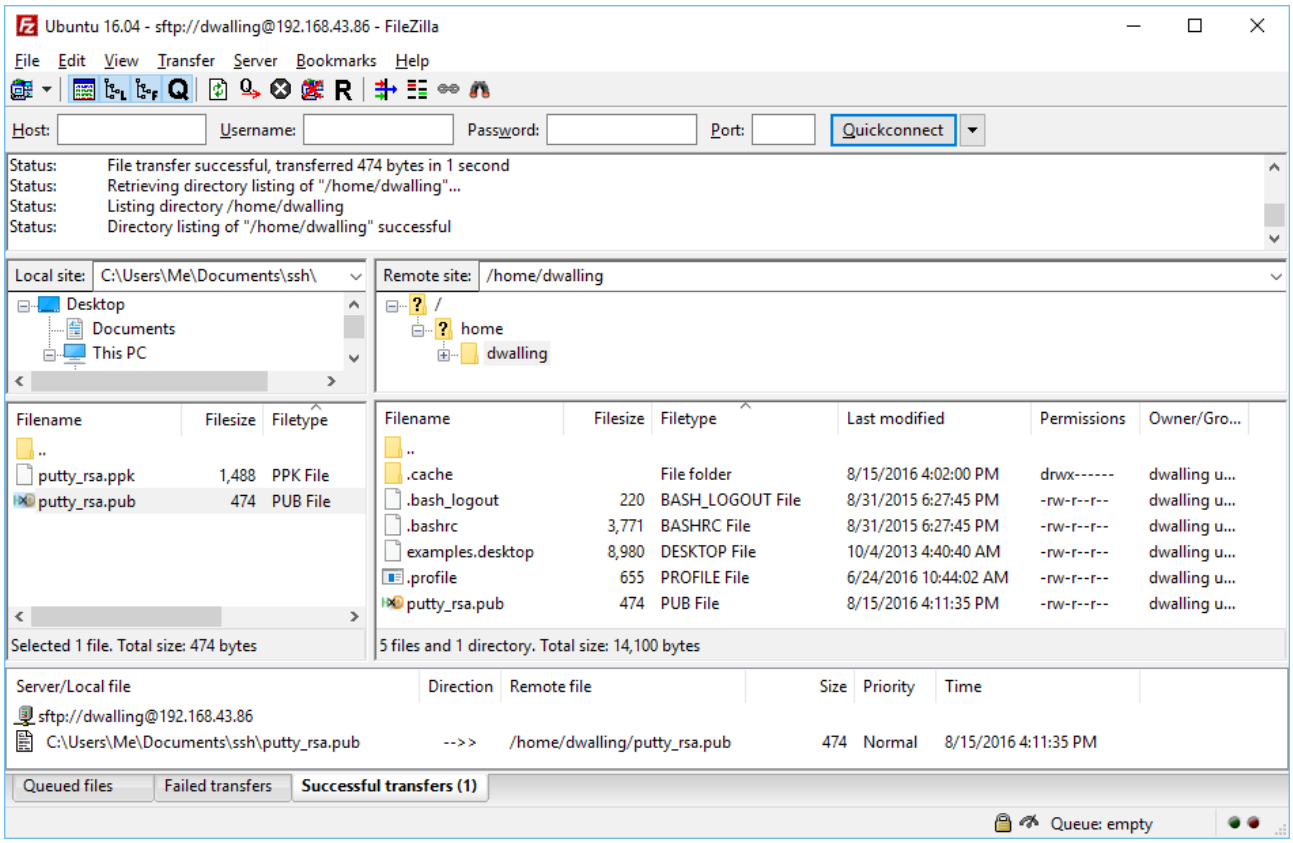
Use the "Save public key" and "Save private key" buttons to save the key pair. The private and public key files should be stored in a location only accessible by the user or the system administrator. I typically save the private key as putty_rsa.ppk and the public key as putty_rsa.pub. But the names of these files is up to you.



Next, we will use SFTP (secure file trasnfer using SSH) to upload the public key we just created to Ubuntu. Be sure to connect to Ubuntu using the new user's logon, "dwalling" in our example. This is because the default user, osboxes, will not have write authority to the new user's home directory.



In FileZilla, drag only the putty_rsa.pub file to the new user's home directory as shown below.



Again, only the putty_rsa.pub file was copied to Ubuntu. Close FileZilla after transferring the file.

Next, we need to convert the public key file we just uploaded to Ubuntu, putty_rsa.pub to OpenSSH format. SSH and PuTTY use different file format. Once converted, the public key content is appended to the file ~/.ssh/authorized_keys. Also, the public key file is copied into ~/.ssh.

Starting out, we log on to Ubuntu as our new user, dwalling. Or, we can log on as osboxes and switch to our new users with the su -l dwalling command. Use the ls -al command to display the contents of the new user's home directory, including the public key we just uploaded.

```
dwalling@osboxes:~$ ls -al
total 40
drwxr-xr-x 3 dwalling users 4096 Aug  4 22:18 .
drwxr-xr-x 4 root     root  4096 Aug  4 21:34 ..
-rw-r--r-- 1 dwalling users  220 Sep  1  2015 .bash_logout
-rw-r--r-- 1 dwalling users 3771 Sep  1  2015 .bashrc
drwx------ 2 dwalling users 4096 Aug  4 22:01 .cache
-rw-r--r-- 1 dwalling users 8980 Oct  4  2013 examples.desktop
-rw-r--r-- 1 dwalling users  655 Jun 24 16:44 .profile
-rw-r--r-- 1 dwalling users  474 Aug  4 22:18 putty_rsa.pub
dwalling@osboxes:~$
```

Use the ssh-keygen command to convert the PuTTY formatted public key, putty_rsa.pub to the OpenSSH formatted file, id_rsa.pub. After the OpenSSH file is created, we delete the PuTTY formatted public key file and create a .ssh folder for the new user.

```
dwalling@osboxes:~$ ssh-keygen -i -f putty_rsa.pub >> id_rsa.pub
dwalling@osboxes:~$ rm -f putty_rsa.pub
dwalling@osboxes:~$ mkdir .ssh
dwalling@osboxes:~$ ls -al
total 44
drwxr-xr-x 4 dwalling users 4096 Aug  4 22:26 .
drwxr-xr-x 4 root     root  4096 Aug  4 21:34 ..
-rw-r--r-- 1 dwalling users  220 Sep  1  2015 .bash_logout
-rw-r--r-- 1 dwalling users 3771 Sep  1  2015 .bashrc
drwx------ 2 dwalling users 4096 Aug  4 22:01 .cache
-rw-r--r-- 1 dwalling users 8980 Oct  4  2013 examples.desktop
-rw-r--r-- 1 dwalling users  381 Aug  4 22:26 id_rsa.pub
-rw-r--r-- 1 dwalling users  655 Jun 24 16:44 .profile
drwxr-xr-x 2 dwalling users 4096 Aug  4 22:26 .ssh
dwalling@osboxes:~$
```

Now we will put two files into the new .ssh folder. We use the cat command to append the content of the id_rsa.pub file to a file called authorized_keys in the .ssh folder. Although in this instance a simple mv command would have worked because there was not authorized_keys files already in .ssh, it is better in practice to always use this cat command format whenever appending new public key data. This will ensure that you do not accidentally overwrite any existing public key data you previously stored in authorized_keys.

We also move the id_rsa.pub file into .ssh. Having this file in .ssh is not required. I like to keep it here, though, just in case authorized_keys gets deleted or corrupted. I can rebuild it from the public keys I have copies of.

```
dwalling@osboxes:~$ cat id_rsa.pub >> ~/.ssh/authorized_keys
dwalling@osboxes:~$ mv id_rsa.pub ~/.ssh
dwalling@osboxes:~$ ls -al ~/.ssh
total 16
drwxr-xr-x 2 dwalling users 4096 Aug  4 22:31 .
drwxr-xr-x 4 dwalling users 4096 Aug  4 22:31 ..
-rw-r--r-- 1 dwalling users  381 Aug  4 22:30 authorized_keys
-rw-r--r-- 1 dwalling users  381 Aug  4 22:26 id_rsa.pub
dwalling@osboxes:~$
```

Now, there are some file permissions that should be changed on the .ssh folder, the new user's home directory and, for that matter, any other user's home directories, to avoid unwarranted access.

```
dwalling@osboxes:~$ chmod 700 .ssh
dwalling@osboxes:~$ chmod 600 .ssh/authorized_keys
dwalling@osboxes:~$ chmod 600 .ssh/id_rsa.pub
dwalling@osboxes:~$ chmod 750 .
dwalling@osboxes:~$ ls -al
total 40
drwxr-x--- 4 dwalling users 4096 Aug  4 22:31 .
drwxr-xr-x 4 root     root  4096 Aug  4 21:34 ..
-rw-r--r-- 1 dwalling users  220 Sep  1  2015 .bash_logout
-rw-r--r-- 1 dwalling users 3771 Sep  1  2015 .bashrc
drwx------ 2 dwalling users 4096 Aug  4 22:01 .cache
-rw-r--r-- 1 dwalling users 8980 Oct  4  2013 examples.desktop
-rw-r--r-- 1 dwalling users  655 Jun 24 16:44 .profile
drwx------ 2 dwalling users 4096 Aug  4 22:31 .ssh
dwalling@osboxes:~$ ls -al .ssh
total 16
drwx------ 2 dwalling users 4096 Aug  4 22:31 .
drwxr-x--- 4 dwalling users 4096 Aug  4 22:31 ..
-rw------- 1 dwalling users  381 Aug  4 22:30 authorized_keys
-rw------- 1 dwalling users  381 Aug  4 22:26 id_rsa.pub
dwalling@osboxes:~$
```

What we are doing here is:

1. Allow ONLY the user dwalling to access the .ssh folder.
2. Allow ONLY the user dwalling to read or write to the files in .ssh
3. Allow ONLY the user dwalling to write in the user's home folder
4. Allow ANY user in the "users" group to read or execute files in the user's home folder.

This last permission change is really up to you. Allowing group members to read and execute from each other's home directories can facilitate collaboration.

What might be surprising is that these permissions do not prevent SSH from accessing the user's private key to authenticate the user's SSH logon as we will see below. This is because OpenSSH's authentication occurs in a process having higher privileges than the unprivileged child process.

We also want to change the permission on the osboxes user's home folder, but the user dwalling does not have permission to do that. So, instead, we will su to the osboxes user and change permission.

```
dwalling@osboxes:/home$ su -l osboxes
Password:
osboxes@osboxes:~$ chmod 750 .
osboxes@osboxes:~$ ls -al ..
total 16
drwxr-xr-x  4 root     root    4096 Aug  4 21:34 .
drwxr-xr-x 24 root     root    4096 Aug  4 01:34 ..
drwxr-x---  4 dwalling users   4096 Aug  4 22:31 dwalling
drwxr-x--- 15 osboxes  osboxes 4096 Aug  4 21:29 osboxes
osboxes@osboxes:~$
```

Now, our users' home directories are protected from unauthorized user activity.

Next, we will edit the sshd_config file to place restrictions on how SSH operates. We need to be logged in as osboxes to make changes to the /etc/ssh/sshd_config file.

```
osboxes@osboxes:~$ sudo cp /etc/ssh/sshd_config /etc/ssh/sshd_config.original
osboxes@osboxes:~$ sudo vi /etc/ssh/sshd_config
```

Here is a summary of the settings we will make. Note that it is always a good practice to copy the original version of the sshd_config file to a backup, such as sshd_config.original, before editing it. Also, it is a good practice to comment out the original setting values and add new lines with the changed values.

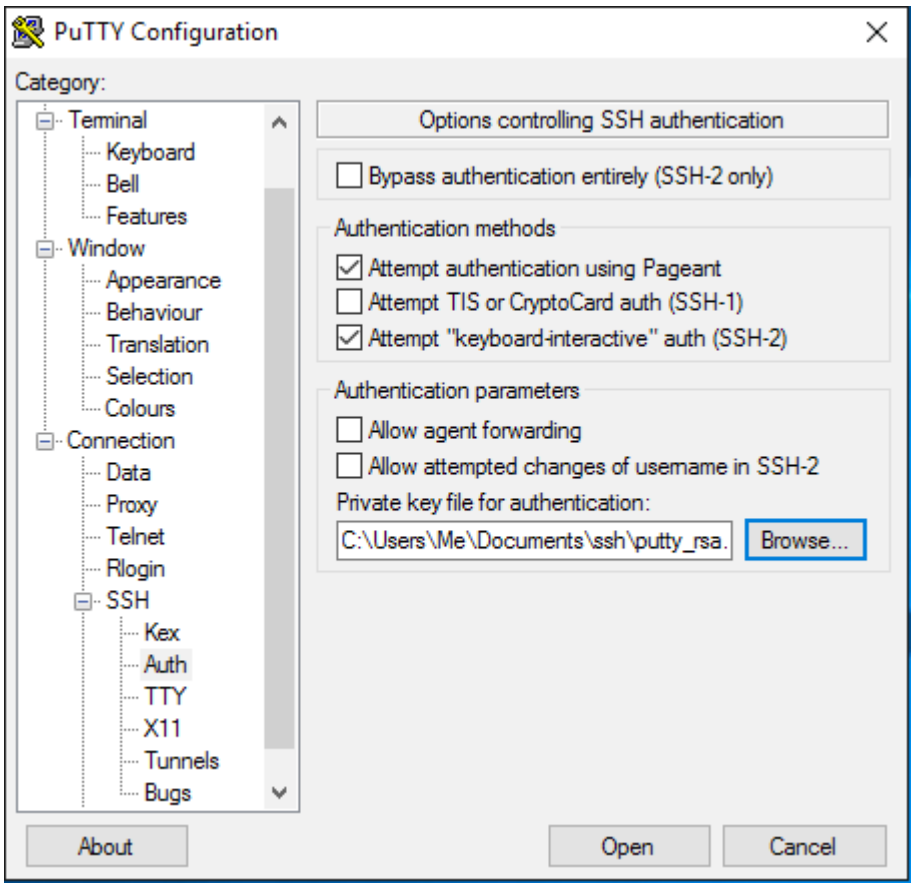
```
ChallengeResponseAuthentication no    # only public key authentication is allowed
ClientAliveCountMax 0                 # do not warn before timing out the SSH session
ClientAliveInterval 900               # timeout client after 15 minutes of inactivity
GSSAPIAuthentication no               # only public key authentication is allowed
ListenAddress 192.168.43.86           # only listen for connections on one interface
LoginGraceTime 30                     # timeout client after 30 seconds if no login
PasswordAuthentication no             # only public key authentication is allowed
PermitEmptyPasswords no               # users without passwords are not allowed
PermitRootLogin no                    # the root user may not connect over SSH
Port 8642                             # use a non-standard port
Protocol 2                            # allow only SSH version 2 connections
PubkeyAuthentication yes              # only public key authentication is allowed
SyslogFacility AUTHPRIV               # restrict access to logs
AllowUsers dwalling                   # allow only this user to access using SSH
X11Forwarding no                      # do not forward to X11 displays
```

After editing sshd_config, restart the ssh service and use netstat to confirm that SSH is listening on the non-standard port defined in sshd_config.

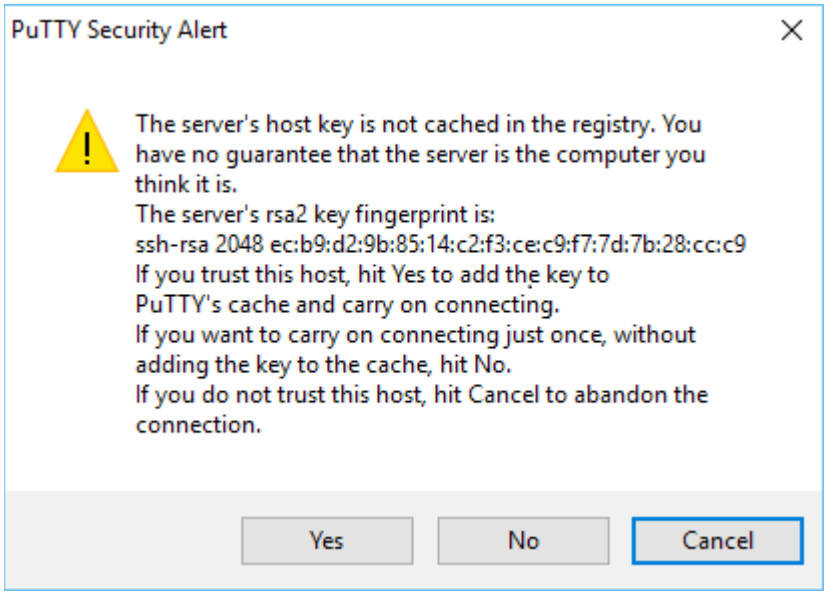
```
osboxes@osboxes:~$ sudo service ssh restart
osboxes@osboxes:~$ netstat -an | grep "LISTEN "
tcp        0      0 127.0.1.1:53            0.0.0.0:*               LISTEN
tcp        0      0 192.168.43.86:8642      0.0.0.0:*               LISTEN
osboxes@osboxes:~$
```

Note: Setting the sshd_config settings above on Ubuntu 16.04.1 will cause the ssh service to fail to start on reboot. We will step through why this is the case and how to fix that issue below.

Now, we will use the PuTTY client to attempt an SSH connection to Ubuntu using credentials for the user dwalling. But first, the PuTTY configuration needs to be setup for Ubuntu. Make sure that the PuTTY configuration references the non-standard port and the private key we created above. Also make sure that SSH-2 is used, not SSH-1.

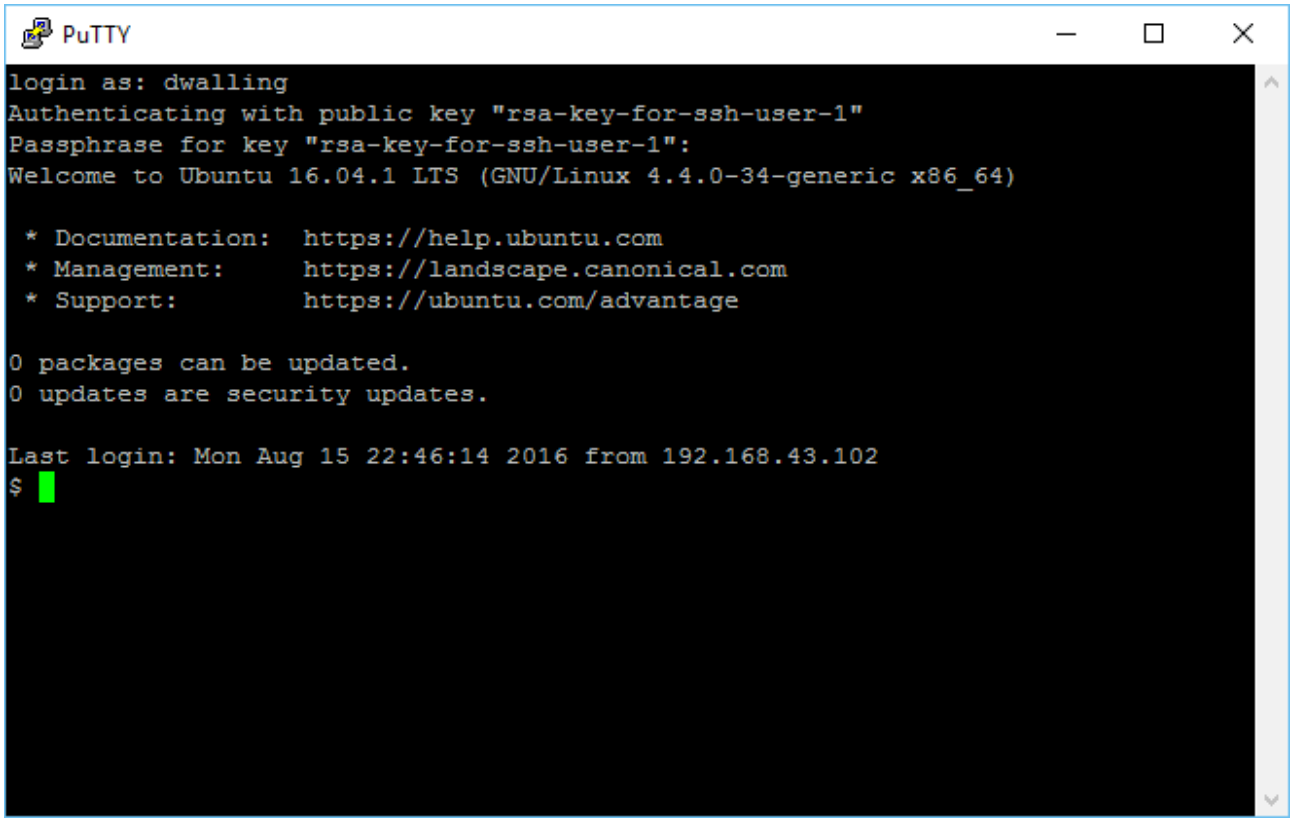


When the SSH connection is made for the first time, it will display the following dialog to confirm the acceptance of the server's key.



Depending on how you are managing server keys on your Windows 10 client, select either the "Yes" or "No" option.

The SSH authentication process will now continue. It will prompt for the user name. When the user name is entered, SSH will search that user's ~/.ssh directory for the authorized keys for the user. Choosing the public key we uploaded, SSH will next request that the user enter the passphrase to unlock the private key stored on the client. Enter the passphrase and hit Enter. If the passphrase is correct, the SSH negotiation will complete and the secure shell session will begin.



Using "netstat" on the Ubuntu host will show that our SSH session is connected from our Windows client.

```
osboxes@osboxes:~$ netstat -an | grep 8642
tcp        0      0 192.168.43.86:8642      0.0.0.0:*               LISTEN
tcp        0      0 192.168.43.86:8642      192.168.43.102:59905    ESTABLISHED
osboxes@osboxes:~$
```

There is one more change we will make to the sshd_config file. When a user connects, we want to display a banner that states our server's policy as to monitoring activity on the server. This both informs the user that their activities may be logged and may dissuade a user from misbehaving.

Edit /etc/issue.net to contain something similar to the following. I include a blank line at the top and bottom so that when it is displayed it will leave a blank line after the login: prompt and before the name of the key as shown below.

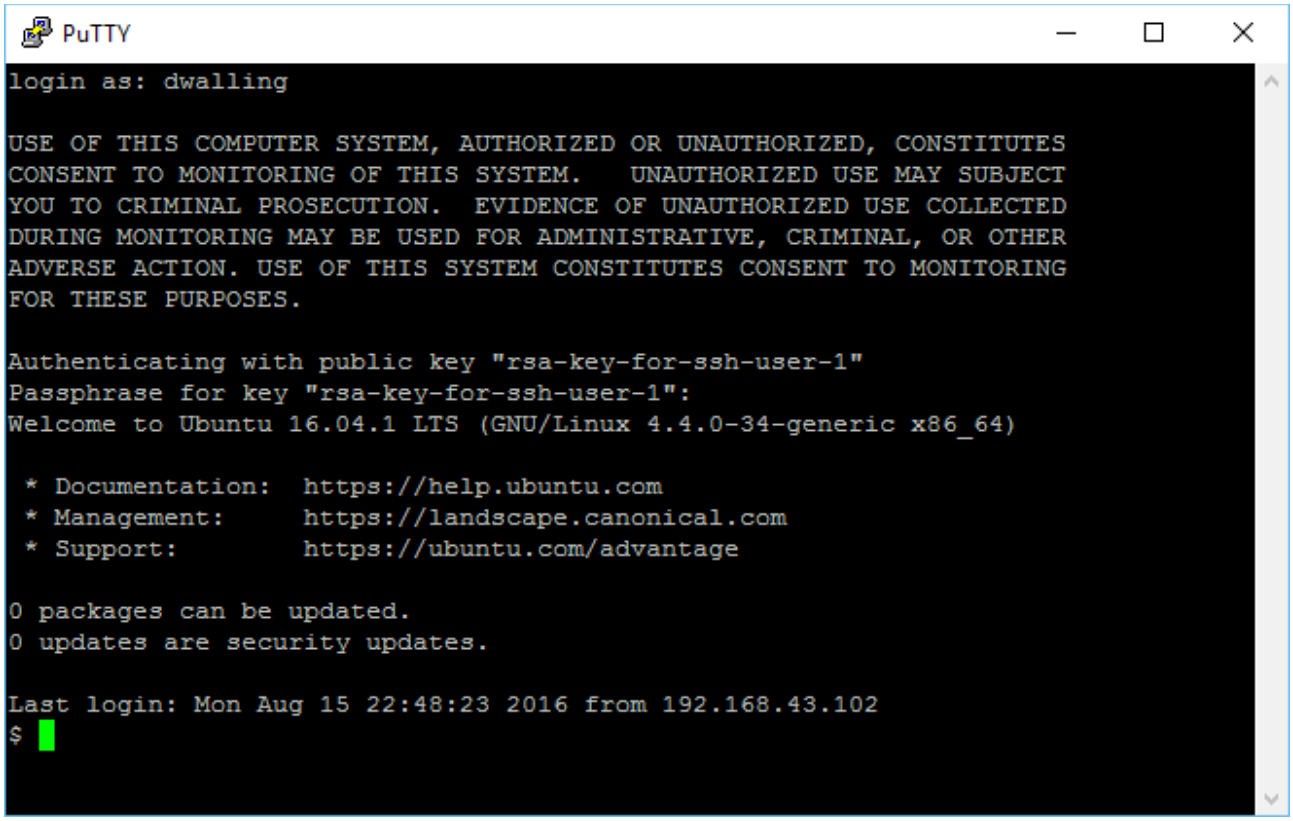
```
USE OF THIS COMPUTER SYSTEM, AUTHORIZED OR UNAUTHORIZED, CONSTITUTES
CONSENT TO MONINTORING OF THIS SYSTEM.  UNAUTHORIZED USE MAY SUBJECT
YOU TO CRIMINAL PROSECUTION.  EVIDENCE OF UNAUTHORIZED USE COLLECTED
DURING MONITORING MAY BE USED FOR ADMINISTRATIVE, CRIMINAL, OR OTHER
ADVERSE ACTION. USE OF THIS SYSTEM CONSTITUTES CONSENT TO MONITORING
FOR THESE PURPOSES.

~
```

Then edit /etc/ssh/sshd_config to uncomment the line:

```
Banner /etc/issue.net
```

Restart the ssh service. Now, when an SSH connection is started, the banner appears prior to requesting the passphrase. Due to our other changes, only the user dwalling can connect and the user must enter the passphrase within 30 seconds.



There is additional information displayed at the start of this SSH terminal session after the authentication is completed. The Ubuntu version is shown, including the patch level, in this case, 16.04.1 LTS. Helpful web-sites are shown. Counts of update packages are shown and a line indicating the last login date and time and source IP address.

What determines the information shown here are the files in /etc/update-motd.d

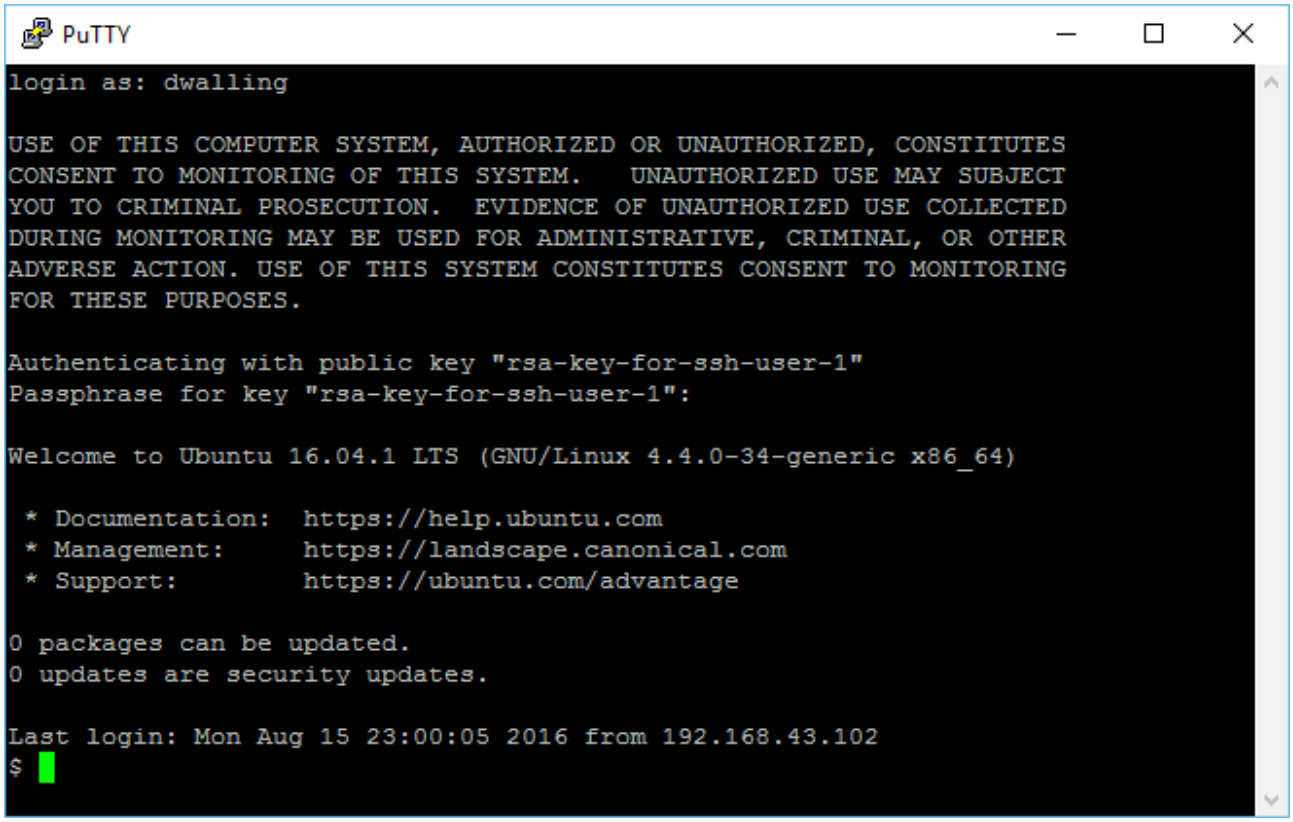
If we list the files in /etc/update-motd.d, we see file that have two-digit numerals leading their file names. This interesting naming convention determines the order (ascending) in which each file is executed to emit text to the screen, the total of all output comprising our message-of-the-day (motd).

```
osboxes@osboxes:~$ ls -al /etc/update-motd.d
total 40
drwxr-xr-x   2 root root  4096 Aug  4 01:34 .
drwxr-xr-x 131 root root 12288 Aug  5 17:34 ..
-rwxr-xr-x   1 root root  1220 Oct 22  2015 00-header
-rwxr-xr-x   1 root root  1157 Jun 14 13:16 10-help-text
-rwxr-xr-x   1 root root    97 Apr 12 11:25 90-updates-available
-rwxr-xr-x   1 root root   299 Feb  8 22:32 91-release-upgrade
-rwxr-xr-x   1 root root   142 Apr 12 11:25 98-fsck-at-reboot
-rwxr-xr-x   1 root root   144 Apr 12 11:25 98-reboot-required
osboxes@osboxes:~$
```

We will leave our message almost unchanged. I will insert a single blank line at the top of the header message, so separate it from the previous authentication messages.

```
sudo vi /etc/update-motd.d/00-header
```

```
#printf "Welcome to %s (%s %s %s)\n" "$DISTRIB_DESCRIPTION" "$(uname -o)" "$(uname -r)" "$(uname -m)"
printf "\nWelcome to %s (%s %s %s)\n" "$DISTRIB_DESCRIPTION" "$(uname -o)" "$(uname -r)" "$(uname -m)"
~
```

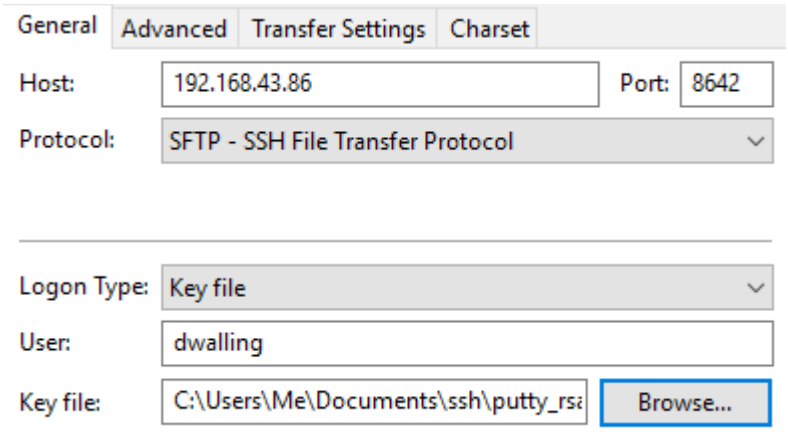


Did you notice that motd displays the Ubuntu release and patch level, "16.04.1"? This information comes from the lsb_release utility. You can also "cat" the /etc/issue file.

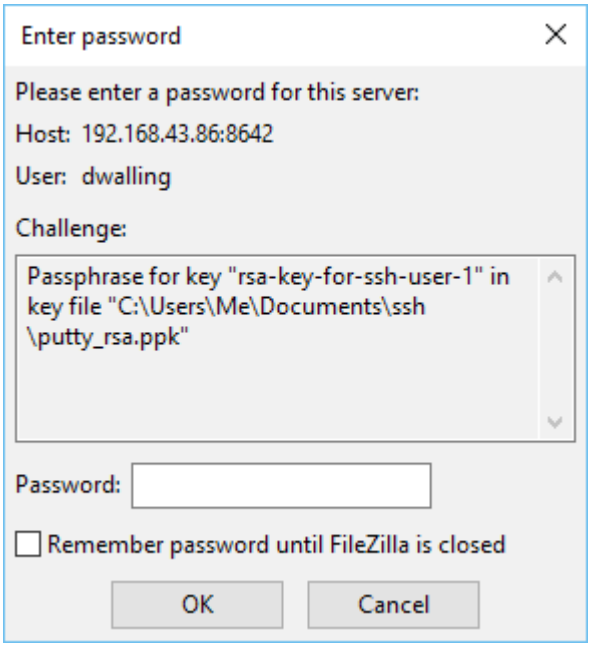
```
osboxes@osboxes:~$ lsb_release -d
Description:    Ubuntu 16.04.1 LTS
osboxes@osboxes:~$ cat /etc/issue
Ubuntu 16.04.1 LTS \n \l
```

Next, to setup FileZilla to connect to Ubuntu over SFTP using the dwalling account, we need to update the Site Settings for our Ubuntu host. Make sure you are using a recent version of FileZilla that supports the "Key file" logon type.

Note that we have specified our non-standard port, a Logon Type of "Key file", our authorized user name and the private key file that will be used for authentication.



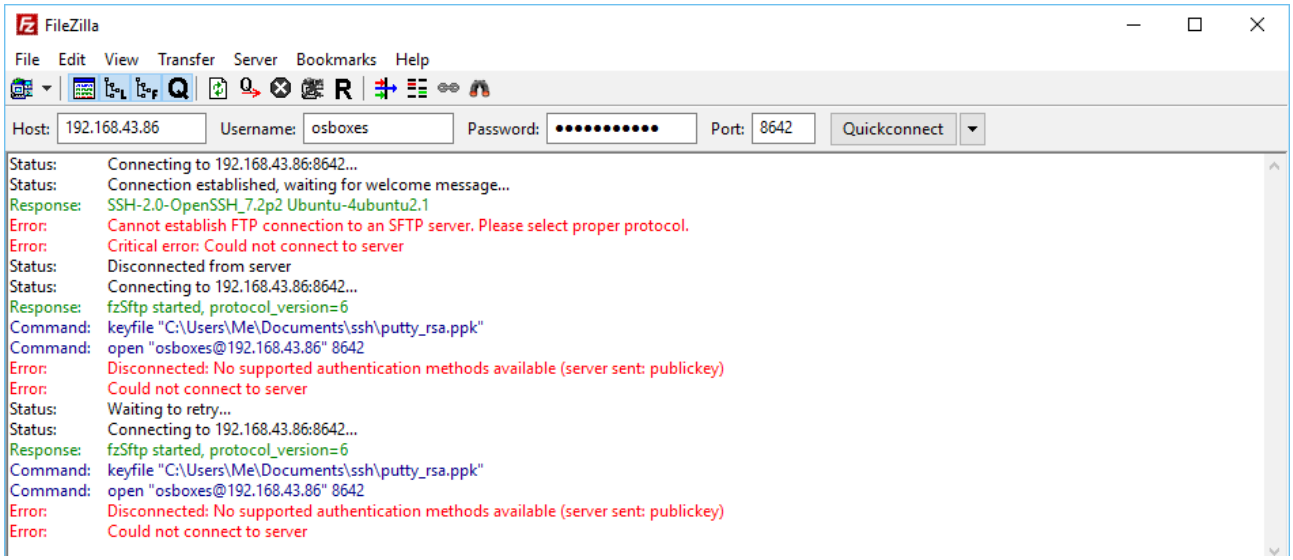
Using this "Key file" logon type, FileZilla will prompt you to enter the private key's passphrase at the start of the SFTP session.


Enter password
Please enter a password for this server:
Host: 192.168.43.86:8642
User: dwalling
Challenge:
Passphrase for key "rsa-key-for-ssh-user-1" in key file "C:\Users\Me\Documents\ssh \putty_rsa.ppk"
Password:
Remember password until FileZilla is closed
OK
Cancel

For improved security, UNCHECK the "Remember password until FileZilla is closed" check-box. If you leave this checked, you will NOT be prompted to re-enter the passphrase each time you connect until exiting FileZilla. Disconnecting from the Ubuntu server will not cause FileZilla to forget the passphrase!

We are able to connect over SFTP to our Ubuntu server now using public-key authentication. Now, let's make sure we CANNOT use the default user, osboxes, to connect using SFTP or SSH.

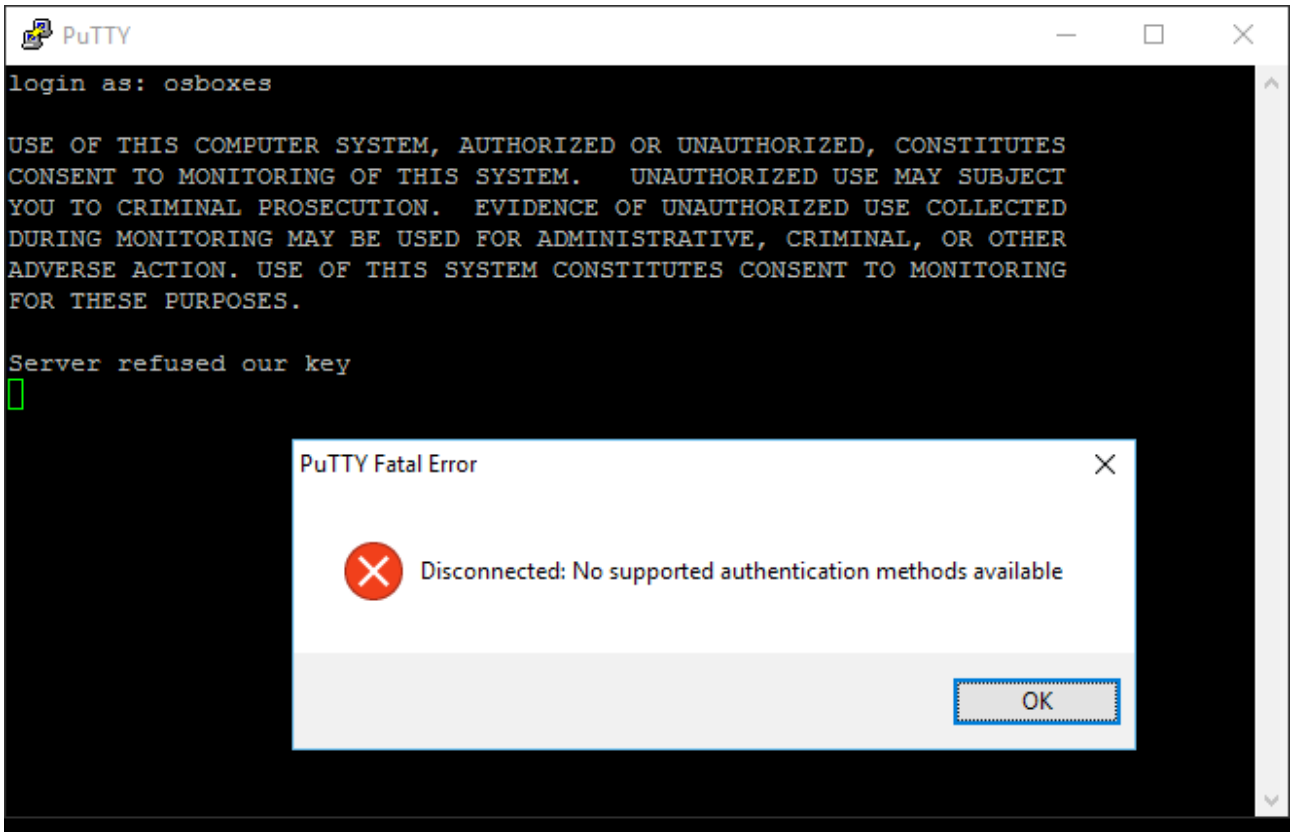
First, SFTP with FileZilla ...



We've tried two different things here. First, using the FileZilla "Quickconnect" option, we specify information - correct IP address and port and correct user name and password. However, FileZilla doesn't recognize the port as valid for any known protocol, hence the "Please select proper protocol error."

Next we opened the File|Site Manager dialog and attempted a "Key file" login using the private-key for dwalling but using the osboxes user name. This attempt failed with the "No supported authentication methods avaiable" error because only the dwalling user has logon privileges.

Next, SSH using PuTTY ...



So, we have demonstrated that we can restrict SSH access to our server to a single user with limited access privileges on our system.

Now, we have added some restrictions to help control access to our Ubuntu server using SSH. As I mentioned earlier, however, there is a problem with Ubuntu's SSH server when it is configured to listen on only one network interface. This problem presents itself on reboot, when SSH fails to restart.

After restarting Ubuntu, issuing a "sudo service ssh status" command reveals the error that occured during boot.

```
osboxes@osboxes:~$ sudo service ssh status
[sudo] password for osboxes:
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: failed (Result: exit-code) since Fri 2016-08-05 17:08:39 BST; 2min 44s ago
  Process: 1958 ExecStart=/usr/sbin/sshd -D $SSHD_OPTS (code=exited, status=255)
 Main PID: 1958 (code=exited, status=255)

Aug 05 17:08:38 osboxes systemd[1]: Starting OpenBSD Secure Shell server...
Aug 05 17:08:39 osboxes systemd[1]: ssh.service: Main process exited, code=exited, status=255/n/a
Aug 05 17:08:39 osboxes systemd[1]: Failed to start OpenBSD Secure Shell server.
Aug 05 17:08:39 osboxes systemd[1]: ssh.service: Unit entered failed state.
Aug 05 17:08:39 osboxes systemd[1]: ssh.service: Failed with result 'exit-code'.
```

What is causing this is Ubuntu attempting to bind our SSH port before our network interfaces are fully initialized.

We have two options here. We can either relax the restriction and let SSH clients connect on our SSH port on any server interface, or we can manually configure our interface, enp0s3, to start SSH when the network interface is up. The second option sounds good - it lets us keep our single-interface restriction. But, there is a catch. Once we update /etc/network/interfaces, we'll lose some ability to configure the interface using the Network settings applet in Ubuntu.

Let's see what this looks like.

Before we edit /etc/network/interfaces, the Network settings window looks like this:

Next, we update /etc/network/interfaces adding the highlighted text below.

```
sudo vi /etc/network/interfaces
```

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
auto enp0s3
iface enp0s3 inet dhcp
  up service ssh start
```

Save this change and restart Ubuntu.

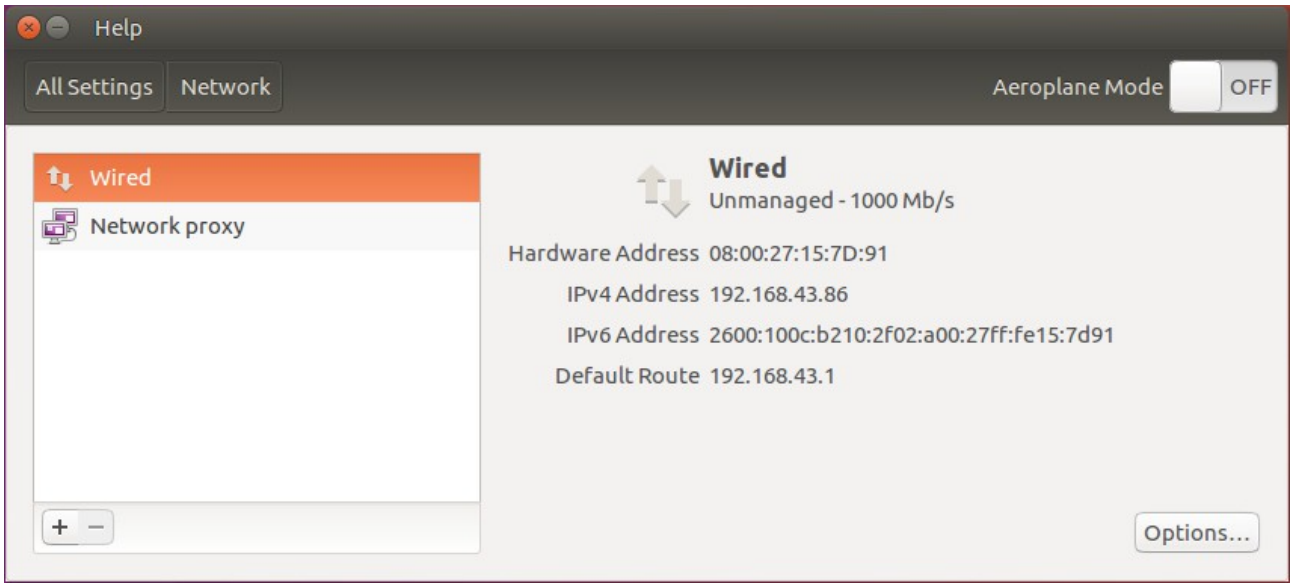
Again, logon and check the status of our SSH service.

```
osboxes@osboxes:~$ sudo service ssh status
[sudo] password for osboxes:
● ssh.service - OpenBSD Secure Shell server
   Loaded: loaded (/lib/systemd/system/ssh.service; enabled; vendor preset: enabled)
   Active: active (running) since Mon 2016-08-15 23:22:59 BST; 1min 49s ago
  Process: 2163 ExecReload=/bin/kill -HUP $MAINPID (code=exited, status=0/SUCCESS)
 Main PID: 2066 (sshd)
   CGroup: /system.slice/ssh.service
           └─2066 /usr/sbin/sshd -D

Aug 15 23:22:58 osboxes systemd[1]: Starting OpenBSD Secure Shell server...
Aug 15 23:22:59 osboxes sshd[2066]: Server listening on 192.168.43.86 port 8642.
Aug 15 23:22:59 osboxes systemd[1]: Started OpenBSD Secure Shell server.
Aug 15 23:22:59 osboxes systemd[1]: Reloading OpenBSD Secure Shell server.
Aug 15 23:23:00 osboxes sshd[2066]: Received SIGHUP; restarting.
Aug 15 23:23:00 osboxes systemd[1]: Reloaded OpenBSD Secure Shell server.
Aug 15 23:23:00 osboxes sshd[2066]: Server listening on 192.168.43.86 port 8642.
osboxes@osboxes:~$
```

Netstat confirms that our server is listening on port 8642 and we are able to connect over SSH from a PuTTY client.

But, if we now open the Network settings window from System Settings, we see some differences. First, the "Wired" network connection appears as "Unmanaged" instead of "Connected". Also, clicking on the "Options..." button now returns an error indicating that no UUID is defined for this connection.



Ultimately, we can hope that an update to Ubuntu will allow us to configure SSH to a single interface and not require manual edits to /etc/network/interfaces to get SSH to start automatically on reboot. We will leave this work-around in place for now.

That concludes this "How To". We have updated our Ubuntu configuration to restrict SSH connections (terminal and SFTP) to using public-key authentication with private-key passphrase for only one authorized account on one network interface.

In the next "How To", we will look at how to use iptables to establish firewall rules on our Ubuntu server to further improve security by restricting open ports, limiting which source IP addresses can connect on our SSH port, and rejecting known bad packets.