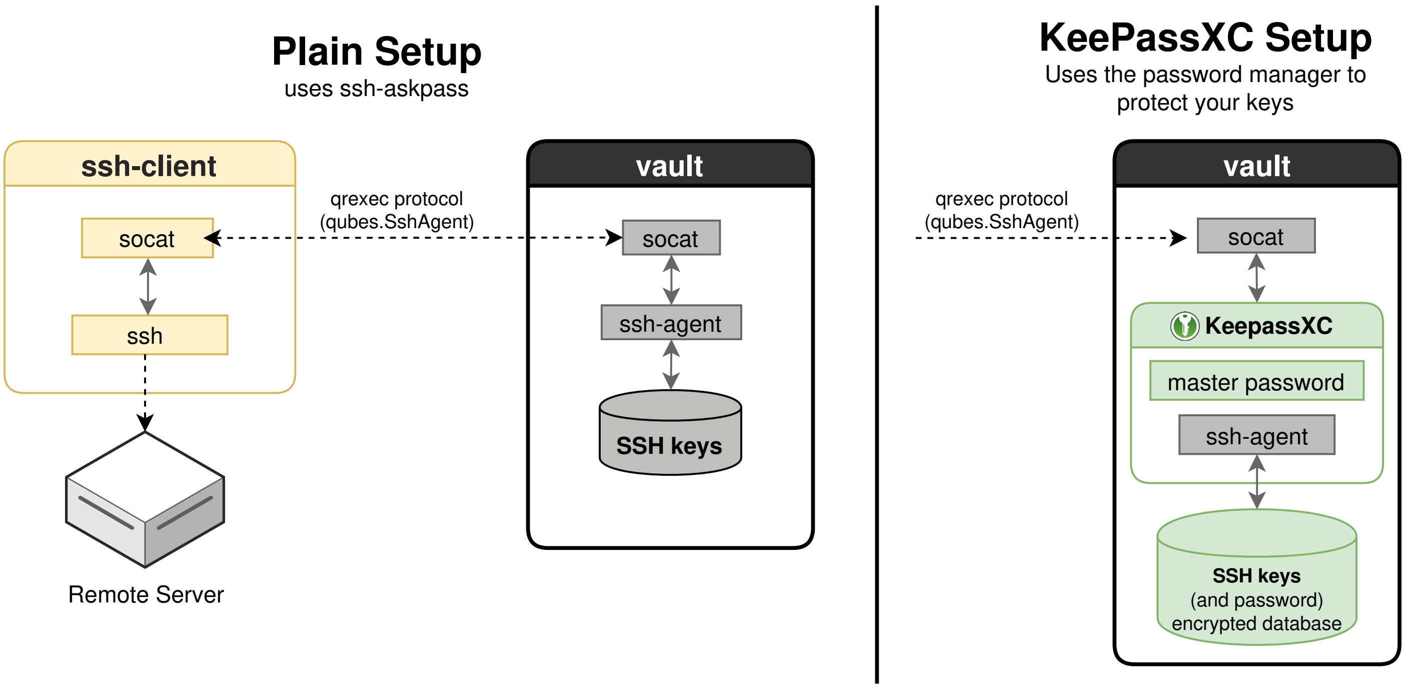
forum.qubes-os.org /t/split-ssh/19060/6

Split SSH - Community Guides - Qubes OS Forum

14-17 minutes ⋮ 5/12/2024

Split SSH implements a concept similar to having a smart card with your private SSH keys, except that the role of the “smart card” is played by another Qubes AppVM. This Qubes setup allows you to keep your SSH private keys in a vault VM (`vault`) while using an SSH Client VM (`ssh-client`) to access your remote server. This is done by using Qubes’s qrexec 8 framework to connect a local SSH Agent socket from your SSH Client VM to the SSH Agent socket within the vault VM. This way the compromise of the domain you use to connect to your remote server does not allow the attacker to automatically also steal all your keys. (We should make a rather obvious comment here that the so-often-used passphrases on private keys are pretty meaningless because the attacker can easily set up a simple backdoor which would wait until the user enters the passphrase and steal the key then.)


Plain Setup
uses ssh-askpass
ssh-client
socat
ssh
qrexec protocol (qubes.SshAgent)
vault
socat
ssh-agent
SSH keys
Remote Server
KeePassXC Setup
Uses the password manager to protect your keys
qrexec protocol (qubes.SshAgent)
vault
socat
KeepassXC
master password
ssh-agent
SSH keys (and password) encrypted database

Security Benefits

In the setup described in this guide, even an attacker who manages to gain access to the `ssh-client` VM will not be able to obtain the user’s private key since it is simply not there. Rather, the private key remains in the `vault` VM, which is extremely unlikely to be compromised if nothing is ever copied or transferred into it. In order to gain access to the vault VM, the attacker would require the use of, e.g., a general Xen VM escape exploit or a signed, compromised package which is already installed in the TemplateVM upon which the vault VM is based.

Overview

1. Make sure the TemplateVM you plan to use is up to date.
2. Create `vault` and `ssh-client` AppVMs.
3. Create an ssh key in your `vault` AppVM and set up automatic key adding prompt.

4. Set up VM interconnection
5. (Strongly Encouraged) Create a KeePassXC Database and set up SSH Agent Integration in KeePassXC.

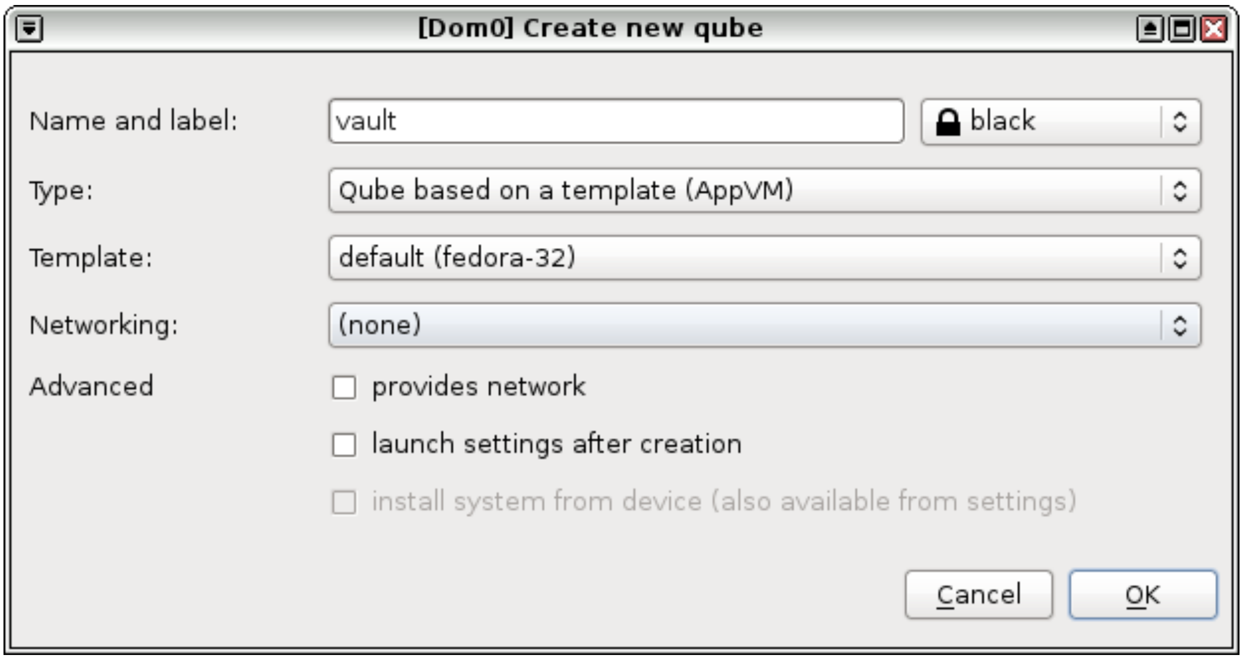
Preparing your system

Make sure the templates you plan to base your AppVMs on are up-to-date 2.

Creating AppVMs 1

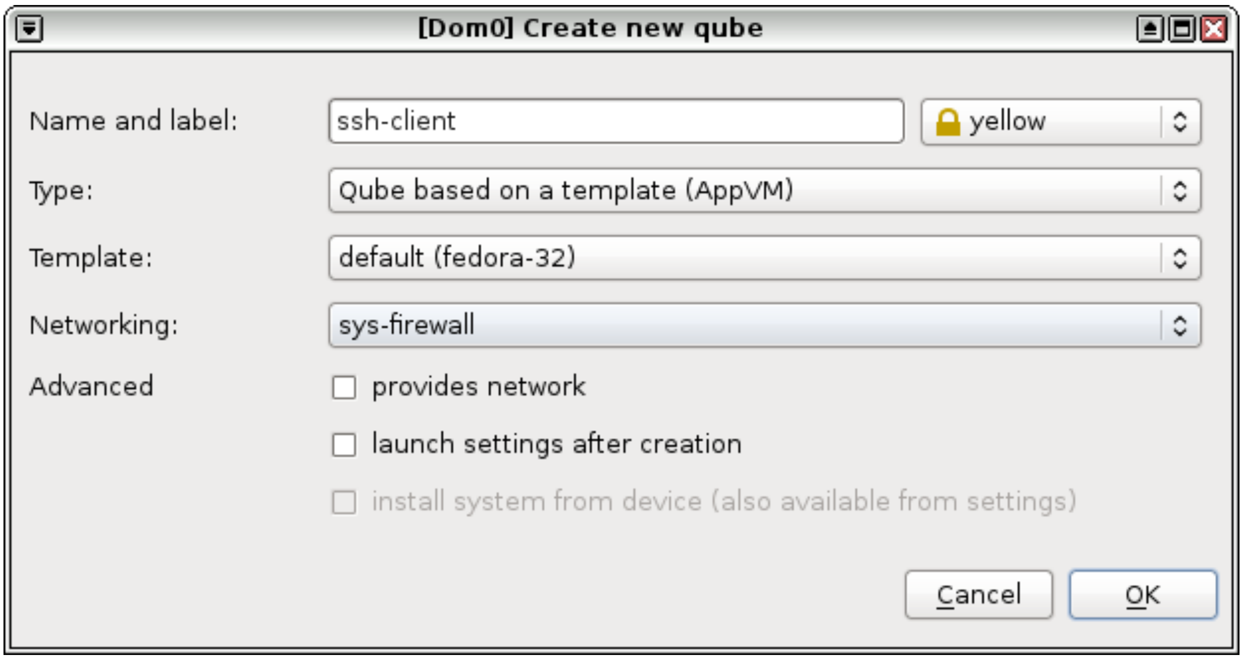
If you've installed Qubes OS using the default options, a few qubes including a vault AppVM has been created for you. Skip the first step if you don't wish to create another vault.

1. Create a new vault AppVM (`vault`) based on your chosen template. Set networking to `(none)`.



Name and label: vault | black
Type: Qube based on a template (AppVM)
Template: default (fedora-32)
Networking: (none)
Advanced
☐ provides network
☐ launch settings after creation
☐ install system from device (also available from settings)

2. Create a SSH Client AppVM (`ssh-client`). This VM will be used to make SSH connections to your remote machine.



Name and label: ssh-client yellow
Type: Qube based on a template (AppVM)
Template: default (fedora-32)
Networking: sys-firewall
Advanced
☐ provides network
☐ launch settings after creation
☐ install system from device (also available from settings)

Setting up SSH

Install `ssh-askpass` in the template of your `vault` VM. It will be used by `ssh-agent` to ask for confirmation, for keys added using `ssh-add -c`.

For Fedora templates:

```
sudo dnf install openssh-askpass
```

For Debian templates:

```
sudo apt-get install ssh-askpass-gnome
```

Perform the next steps in the AppVM `vault`.

1. Generate an SSH key pair. Skip this step if you already have your keys. Note that it is *okay* to not enter a password for your private keys since the `vault` AppVM has no networking. If you still want to encrypt your keys you must refer to the Securing Your Private Key section.

   ```
   [user@vault ~]$ ssh-keygen -t ed25519 -a 500
   Generating public/private ed25519 key pair.
   Enter file in which to save the key (/home/user/.ssh/id_ed25519):
   Created directory '/home/user/.ssh'.
   Enter passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved in /home/user/.ssh/id_ed25519
   Your public key has been saved in /home/user/.ssh/id_ed25519.pub
   The key fingerprint is:
   SHA256:DBxSxZcp16d1NSVSid3m8HRipUDM2INghQ4Sx3jPEDo user@vault
   The key's randomart image is:
   ```

```
+--[ED25519 256]--+
|    o==+++.@++o=*|
|    o==o+ B BoOoB|
|    Eoo* +   *.O.|
|     . o+    .   o|
|        S        |
|                 |
|                 |
|                 |
|                 |
+----[SHA256]-----+
```

-t: type
-a: num_trials

Please note that the key fingerprint and the randomart image will differ.

For more information about `ssh-keygen`, run `man ssh-keygen`.

Notice: Skip the following steps if you plan on using KeePassXC.

2. Make a new directory `~/.config/autostart`

   ```
   mkdir -p ~/.config/autostart
   ```

3. Create the file `~/.config/autostart/ssh-add.desktop`

   - Open the file with e.g. `gedit`

     ```
     gedit ~/.config/autostart/ssh-add.desktop
     ```

   - Paste the following contents:

     ```
     [Desktop Entry]
     Name=ssh-add
     Exec=ssh-add -c
     Type=Application
     ```

   Note: If you've specified a custom name for your key using *-f*, you should adjust `Exec=ssh-add` to `Exec=ssh-add <path-to-your-key-file>`.

Setting Up VM Interconnection

In `dom0`:

To control which VM is allowed as a client, which may act as the server and how we want this interaction to happen, we have to write a policy file for qrexec in `dom0`.

1. Create and edit `/etc/qubes/policy.d/50-ssh.policy`.

   - Open the file with e.g. nano.

     ```
     sudo nano /etc/qubes/policy.d/50-ssh.policy
     ```

   - If you want to explicitly allow only this connection, add the following line:

     ```
     qubes.SshAgent * ssh-client vault ask target=vault
     ```

   - If you want all of your VMs to potentially be an `ssh-client` or a `vault`, add the following line:

     ```
     qubes.SshAgent * @anyvm @anyvm ask
     ```

   - If you want the input field to be "prefilled" by your `vault` VM, append `target=vault` so it looks like for example:

     ```
     qubes.SshAgent * @anyvm @anyvm ask target=vault
     ```

▸ Old policy file for Qubes R4.1

Note: There are many ways to fine-tune this policy. For more details see the Qubes qrexec documentation 32.

In the Template of Your AppVM `vault`:

We now need to write a small script that handles connection requests from `ssh-client` and forwards them to the SSH agent in your `vault`. As we are using qrexec as communication method we have to place it in a special location and have to name it just as the policy file we just created in `dom0`.

1. Create and edit `/etc/qubes-rpc/qubes.SshAgent`.

   - Open the file with e.g. `gedit`

     ```
     sudo gedit /etc/qubes-rpc/qubes.SshAgent
     ```

   - Paste the following contents:

     ```
     #!/bin/sh
     # Qubes App Split SSH Script

     # safeguard - Qubes notification bubble for each ssh request
     ```

```
notify-send "[$(qubesdb-read /name)] SSH agent access from:
$QREXEC_REMOTE_DOMAIN"

# SSH connection
socat - "UNIX-CONNECT:$SSH_AUTH_SOCK"
```

2. Make it executable

```
sudo chmod +x /etc/qubes-rpc/qubes.SshAgent
```

In the AppVM `ssh-client`

Theoretically, you can use SSH in any AppVM. However, if you are considering split-SSH as an additional security layer it is probably reasonable to also think about which VMs you will be using SSH in. For instance, you might want a dedicated `admin` domain for these purposes. Depending on how many systems you plan to access and where they are located, it could also be preferable to have different VMs with different firewall rules for Intranet and Internet administration.

We want to make sure that our `ssh-client` is prepared to use split-ssh right after the VM has started. Therefore, we add a script in `rc.local` (Which will run at VM startup) to listen for responses from `vault` and make SSH use this connection by modifying the user's `.bashrc`.

1. Edit `/rw/config/rc.local`.
   - Open the file with e.g. `gedit`.

     ```
     sudo gedit /rw/config/rc.local
     ```

   - Add the following to the bottom of the file:

     ```
     # SPLIT SSH CONFIGURATION >>>
     # replace "vault" with your AppVM name which stores the ssh
     private key(s)
     SSH_VAULT_VM="vault"

     if [ "$SSH_VAULT_VM" != "" ]; then
       export SSH_SOCK="/home/user/.SSH_AGENT_$SSH_VAULT_VM"
       rm -f "$SSH_SOCK"
       sudo -u user /bin/sh -c "umask 177 && exec socat 'UNIX-LISTEN:
     $SSH_SOCK,fork' 'EXEC:qrexec-client-vm $SSH_VAULT_VM
     qubes.SshAgent'" &
     fi
     # <<< SPLIT SSH CONFIGURATION
     ```

2. Edit `~/.bashrc` and add the following to the bottom of the file:

- Open the file with e.g. `gedit`

  ```
  gedit ~/.bashrc
  ```

- Add the following to the bottom of the file:

  ```
  # SPLIT SSH CONFIGURATION >>>
  # replace "vault" with your AppVM name which stores the ssh
  private key(s)
  SSH_VAULT_VM="vault"

  if [ "$SSH_VAULT_VM" != "" ]; then
    export SSH_AUTH_SOCK="/home/user/.SSH_AGENT_$SSH_VAULT_VM"
  fi
  # <<< SPLIT SSH CONFIGURATION
  ```

Securing Your Private Key

Although passwords wouldn't protect you against a full system compromise (attacker could place a keylogger), it's possible for an adversary to gain read-only access to some of your files (e.g., file shares or offline backups of data). This becomes even more likely if you plan to also use your data outside of Qubes and not be able to modify anything. Passwords are advisable for mitigating these threats .

You can either use the built-in password utility of your private key combined with a graphical prompt or prefer to use KeePassXC. Please note that since `ssh-askpass` prompt is displayed on `vault` VM boot, it is not possible to use both configurations simultaneously.

Using the Built-in Password Utility and `ssh-askpass`

You should have added `ssh-askpass` to your vault template earlier when setting up SSH.

1. Either add a password to an existing private key with `ssh-keygen -p` or directly create a key pair with a password (enter password when prompted during the creation process, see above). Note that the location and name of your private key may differ.

   ```
   [user@vault ~]$ ssh-keygen -p
   Enter file in which the key is (/home/user/.ssh/id_rsa): /home/
   user/.ssh/id_ed25519
   Key has comment 'user@vault'
   Enter new passphrase (empty for no passphrase):
   Enter same passphrase again:
   Your identification has been saved with the new passphrase.
   ```

2. Shutdown the template and restart your `vault` VM.

With this configuration you'll be prompted for entering your password every time you start your vault VM to be able to make use of your SSH key.

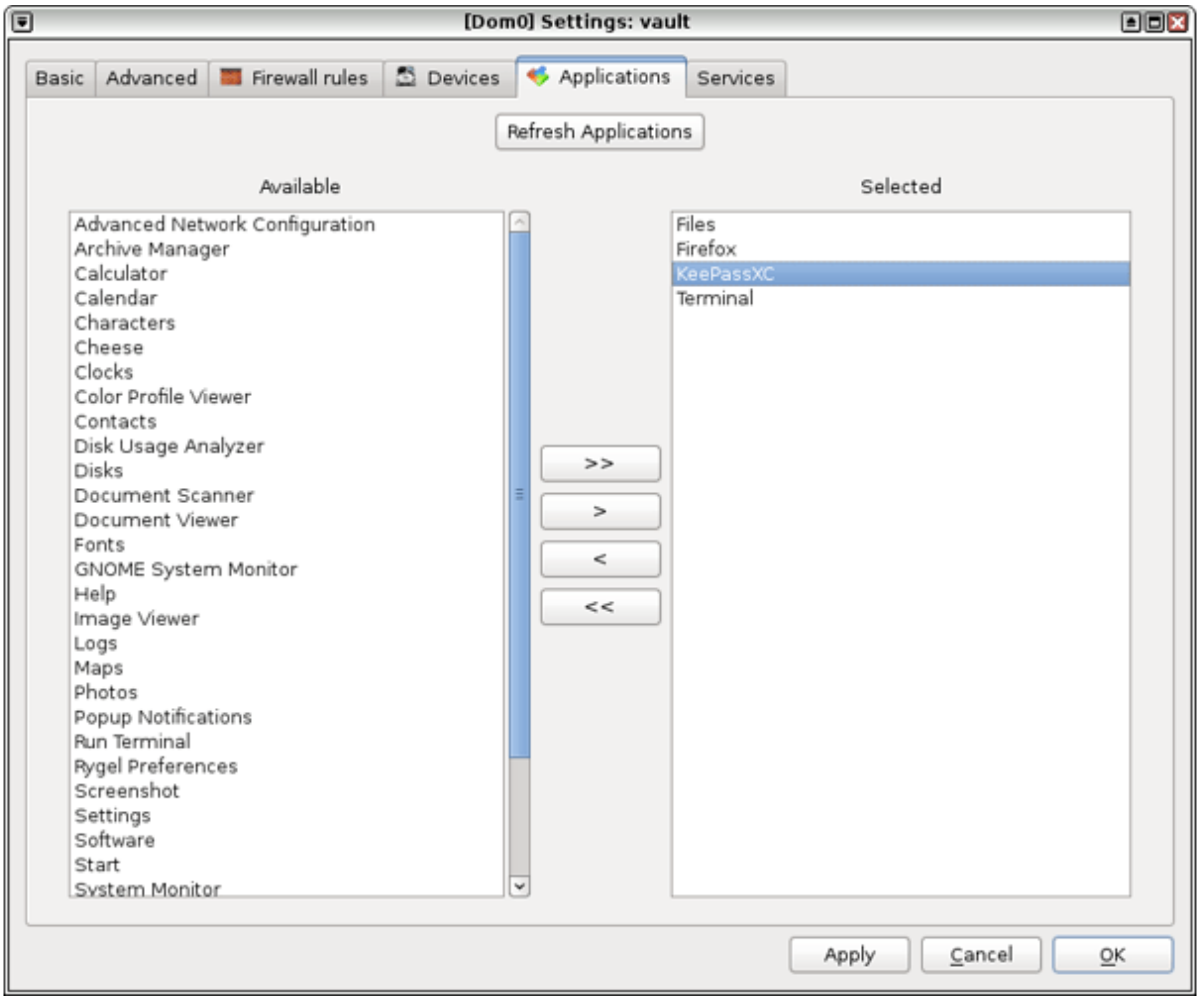
Using KeePassXC 1

Note: This part is for setting up *KeePassXC*, not KeePassX or KeePass. See the KeePassXC FAQ 2.

KeePassXC should be installed by default in both Fedora and Debian TemplateVMs. If it's not or you're using another template, you can install it manually.

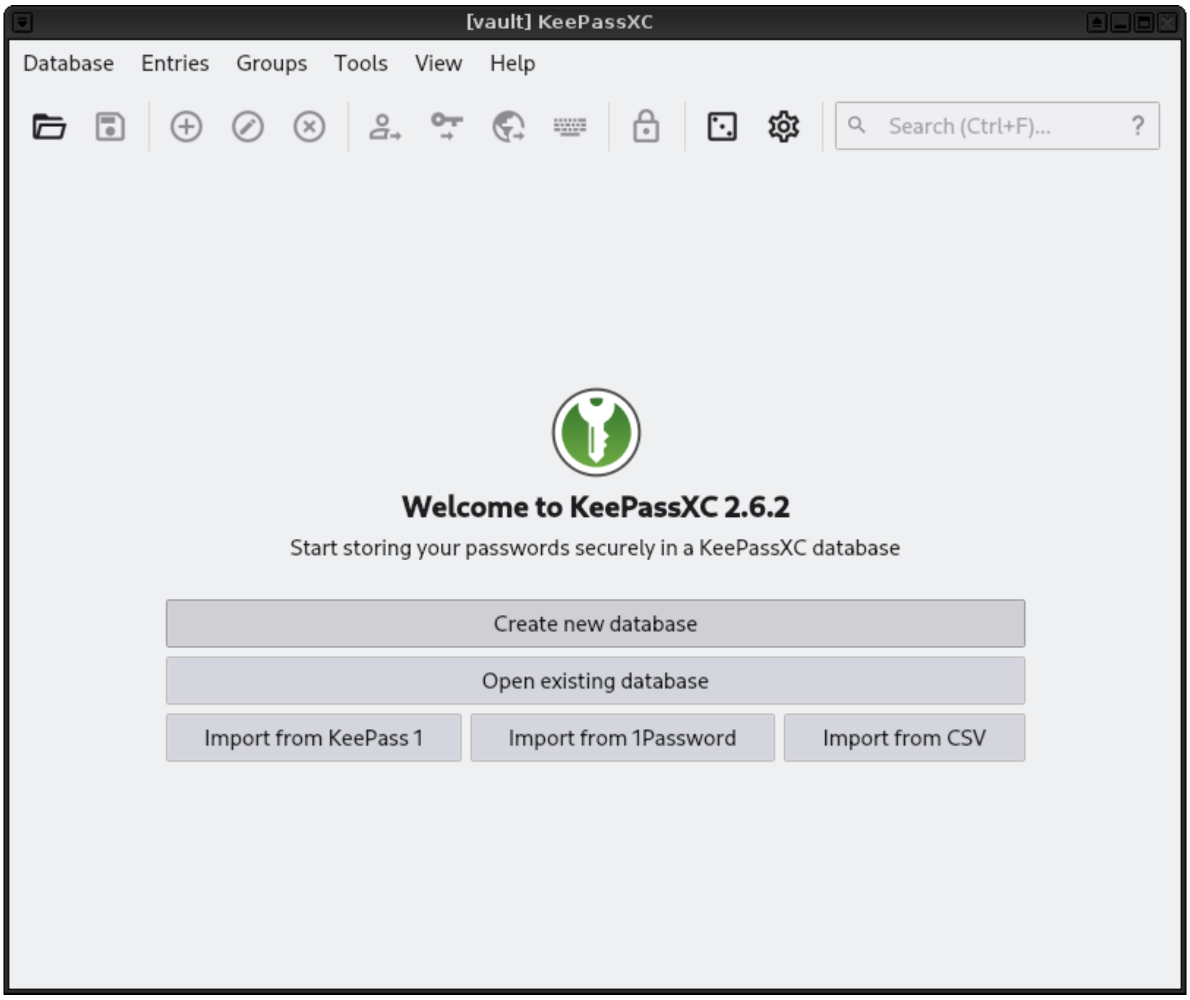
A note on managing your Qubes domains: You might already be using KeePassXC to store your passwords. This guide explains how to set up a new KeePass database and use it exclusively for SSH keys. However, it is also possible to mix your passwords and SSH keys in one big database. Others might have dedicated databases for different kinds of passwords, potentially even in different `vault` VMs. There is no right or wrong here. Feel free to skip the first five steps if you want to use an existing database.

1. Add KeepasXC to the Applications menu of the newly created AppVM for ease of access and launch it.

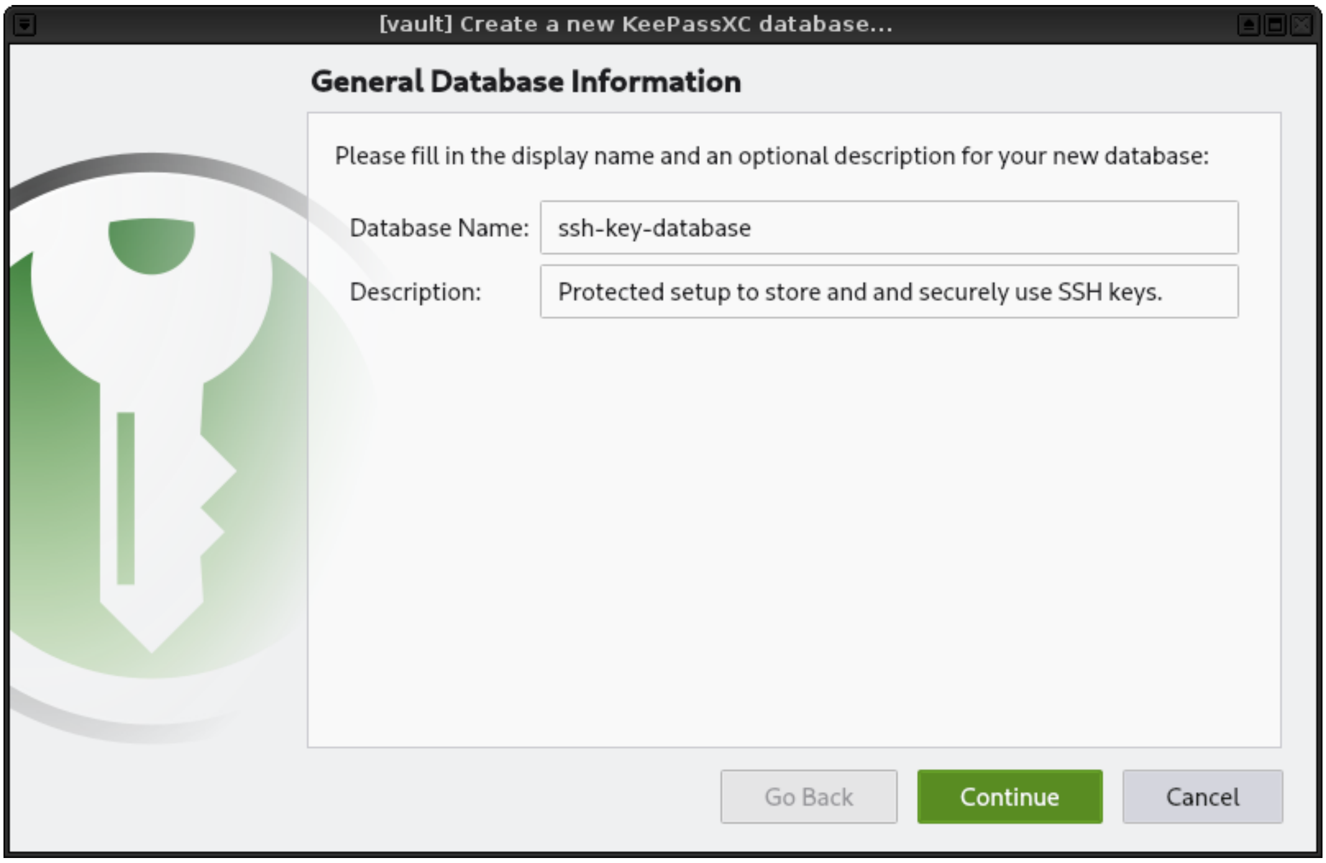


   Note: Since the vault VM has no internet connection, you can safely deny automatic updates if prompted.

2. Create a new database.



3. Enter a name for your database and continue.


[vault] Create a new KeePassXC database...
General Database Information
Go Back
Continue
Cancel

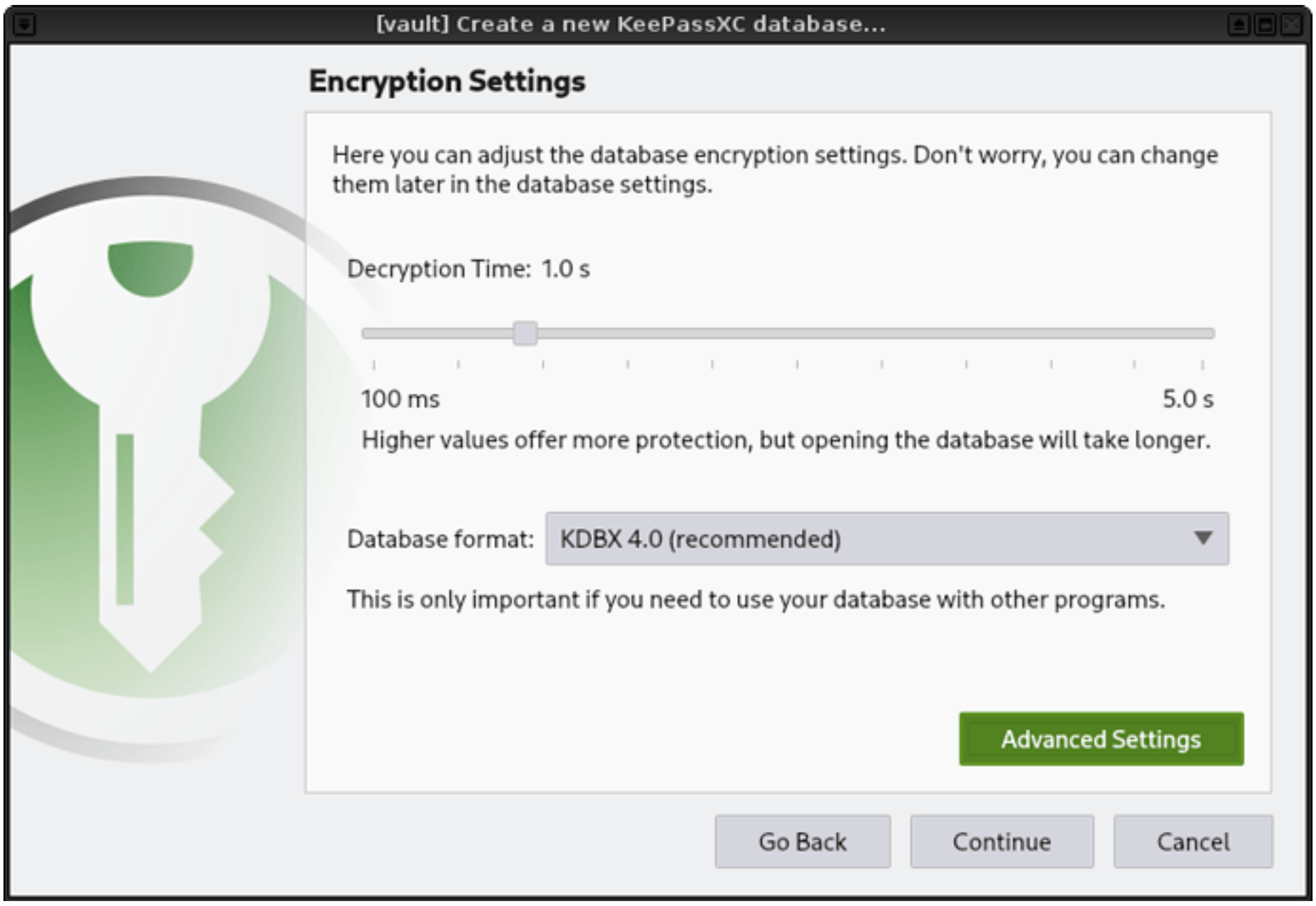
Please fill in the display name and an optional description for your new database:

Database Name: ssh-key-database

Description: Protected setup to store and and securely use SSH keys.

4. Adjust the encryption settings. Check the KeePassXC User Guide 2 for more information about

these settings.


[vault] Create a new KeePassXC database...

Encryption Settings

Here you can adjust the database encryption settings. Don't worry, you can change them later in the database settings.

Decryption Time: 1.0 s

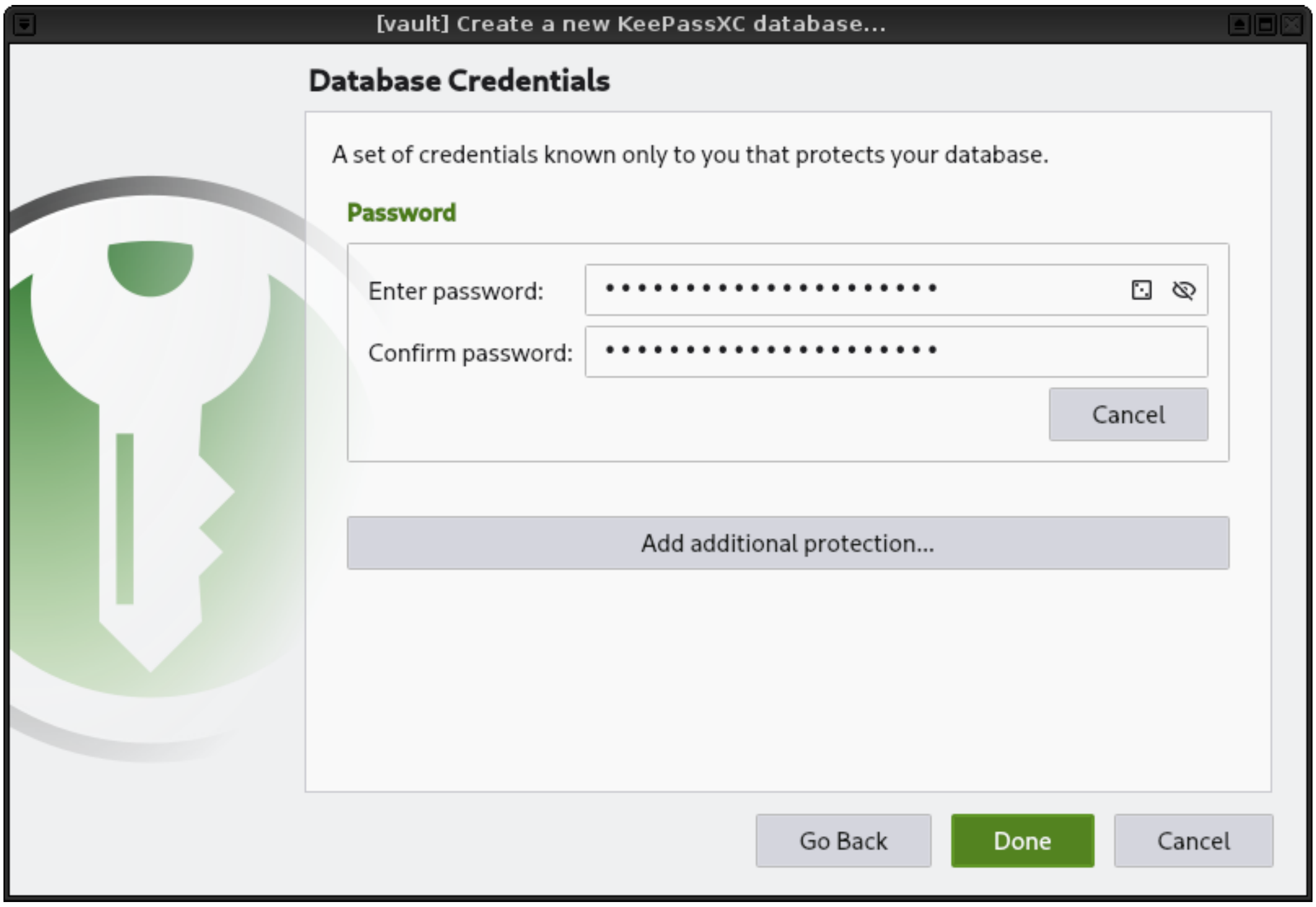
100 ms 5.0 s

Higher values offer more protection, but opening the database will take longer.

Database format: KDBX 4.0 (recommended)

This is only important if you need to use your database with other programs.

5. Enter a password for your database. Take your time make a secure but also rememberable password. (hint 8)


[vault] Create a new KeePassXC database...

Database Credentials

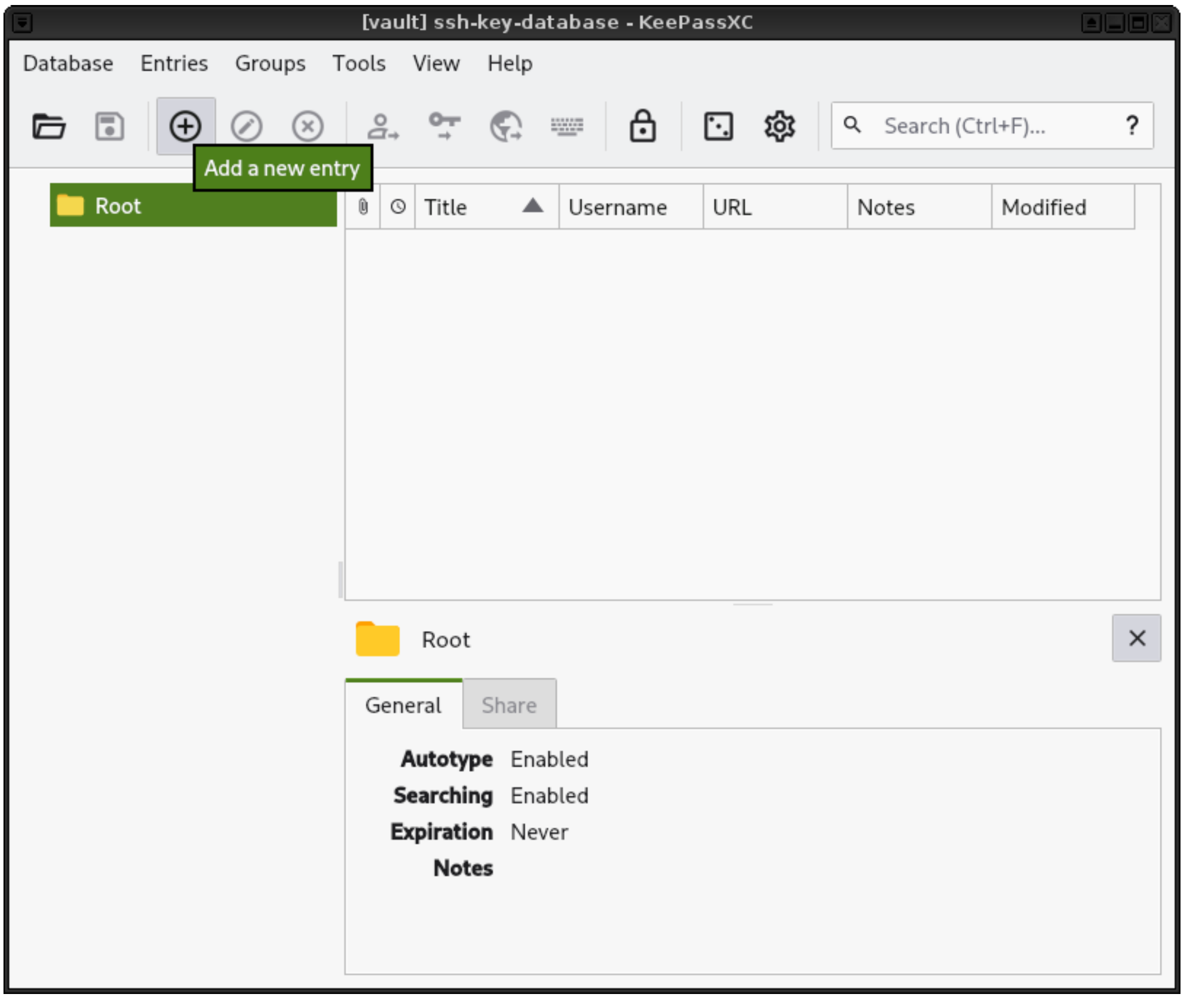
A set of credentials known only to you that protects your database.

Password

Enter password: ••••••••••••••••••••••

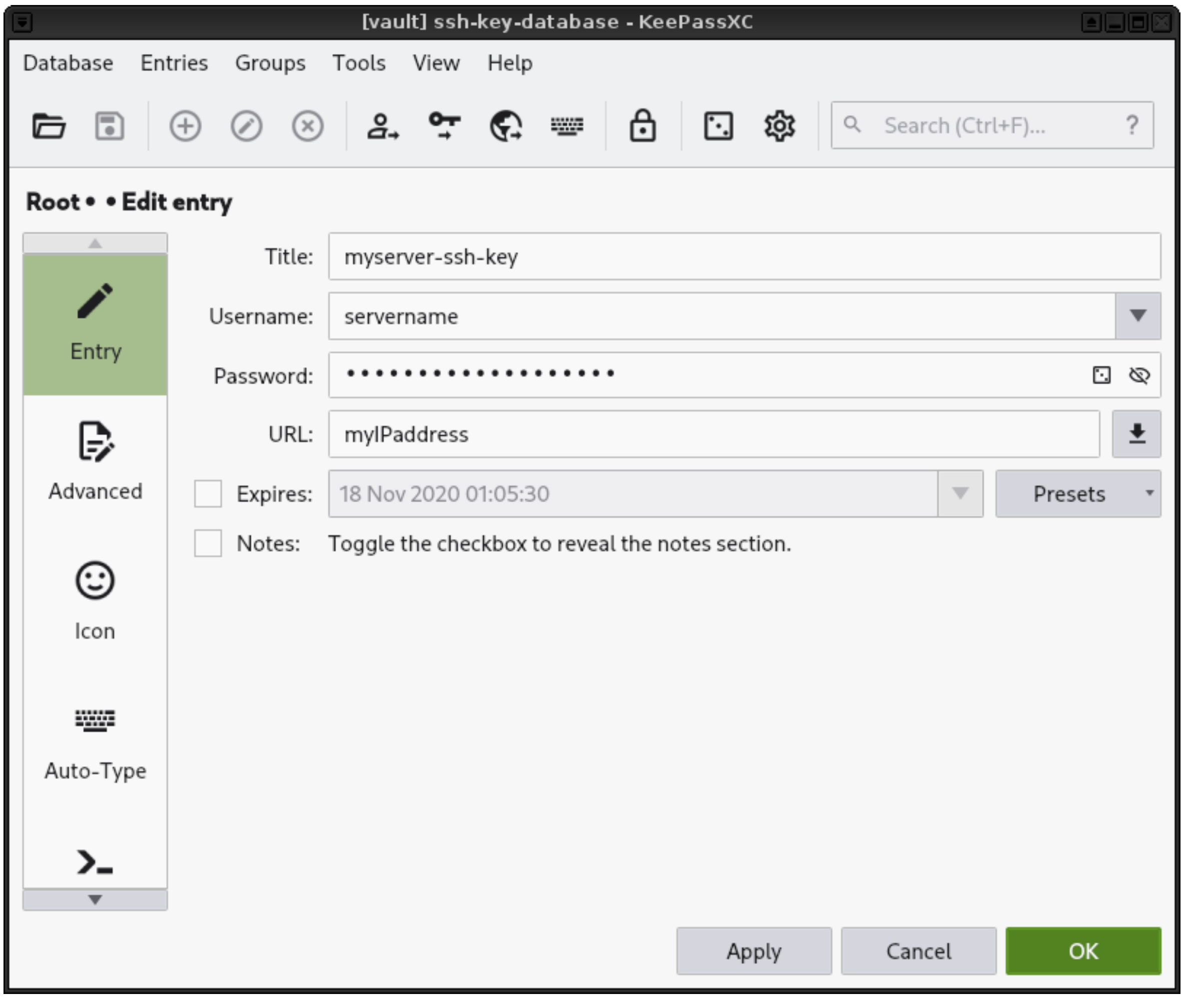
Confirm password: ••••••••••••••••••••••

6. Add a new entry.



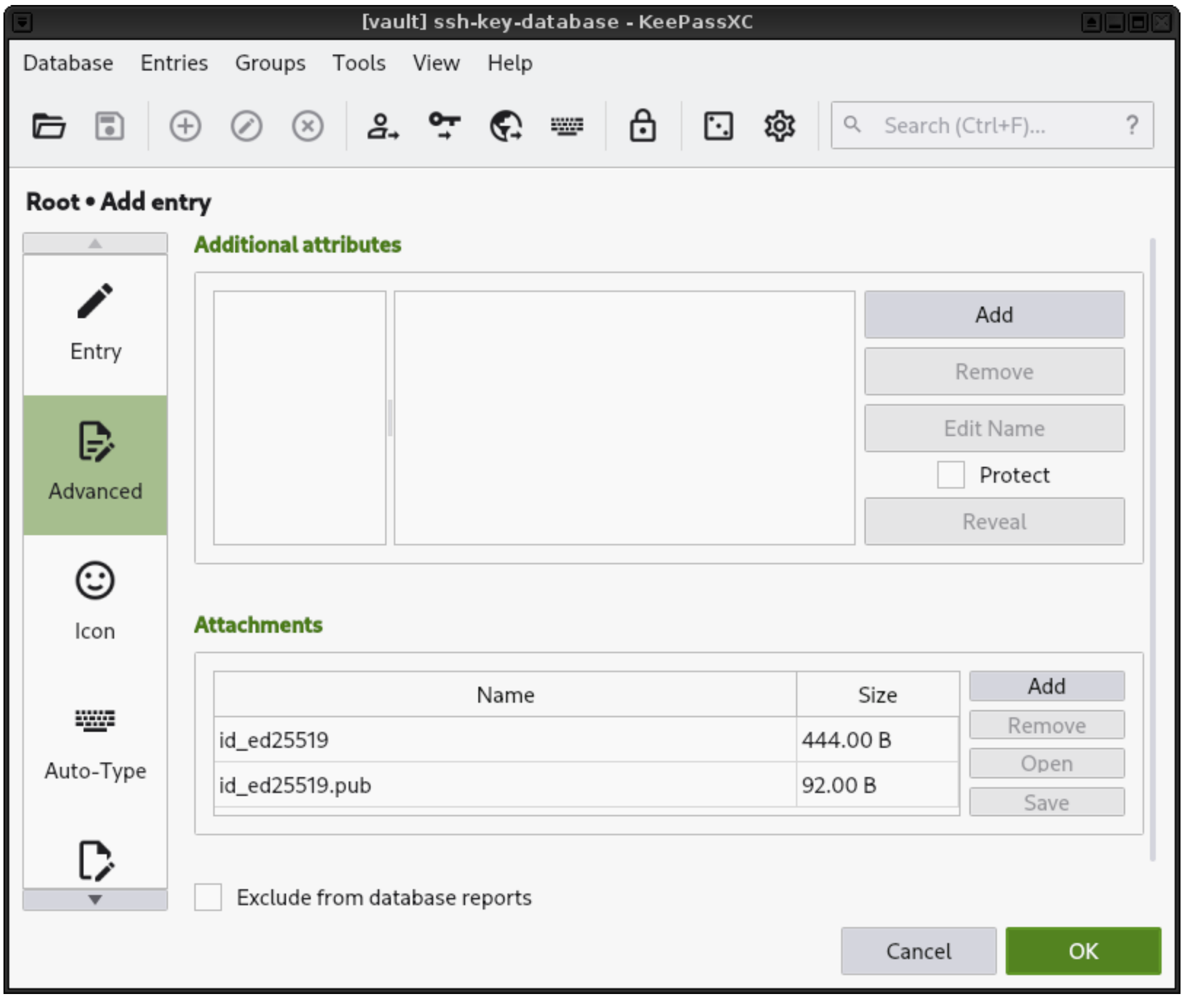
Autotype Enabled
Searching Enabled
Expiration Never
Notes

7. Set password to your SSH key passphrase.



Title: myserver-ssh-key
Username: servername
Password: •••••••••••••••••••
URL: myIPaddress
Expires: 18 Nov 2020 01:05:30
Notes: Toggle the checkbox to reveal the notes section.

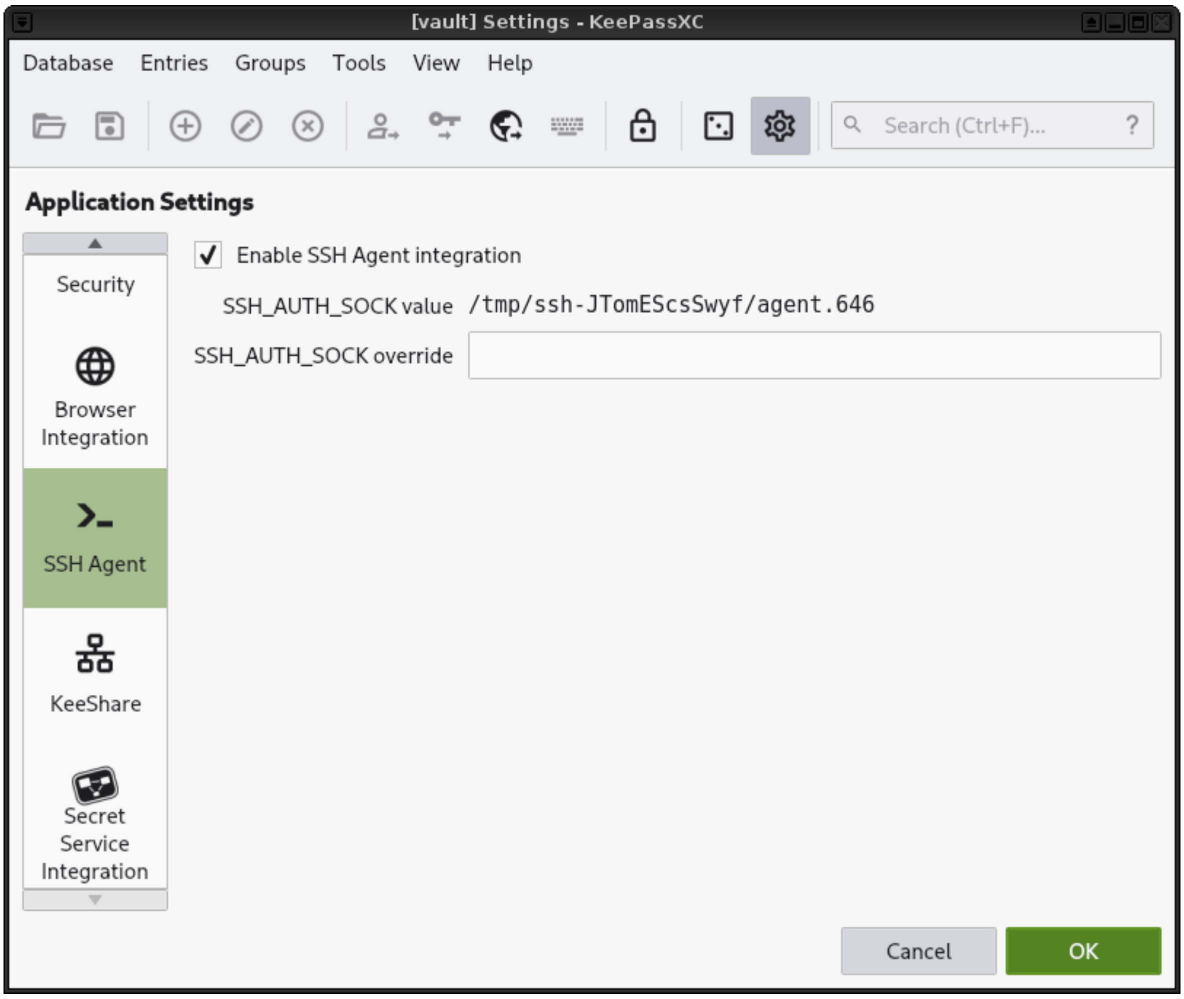
8. Go into the Advanced section and add your keys.



Name	Size
id_ed25519	444.00 B
id_ed25519.pub	92.00 B

Note: Technically, you only need to add the private key (`id_25519`) for the following steps to work. If you add the public key here, too, you can later on backup your kdbx file and have everything in one place. You can even delete your keys (`id_25519` and `id_25519.pub`) from your file system if you like.

9. Enable "SSH Agent Integration" within the Application Settings.


[vault] Settings - KeePassXC
Database Entries Groups Tools View Help
Search (Ctrl+F)...
Application Settings
Security
Browser Integration
SSH Agent
KeeShare
Secret Service Integration
Cancel
OK

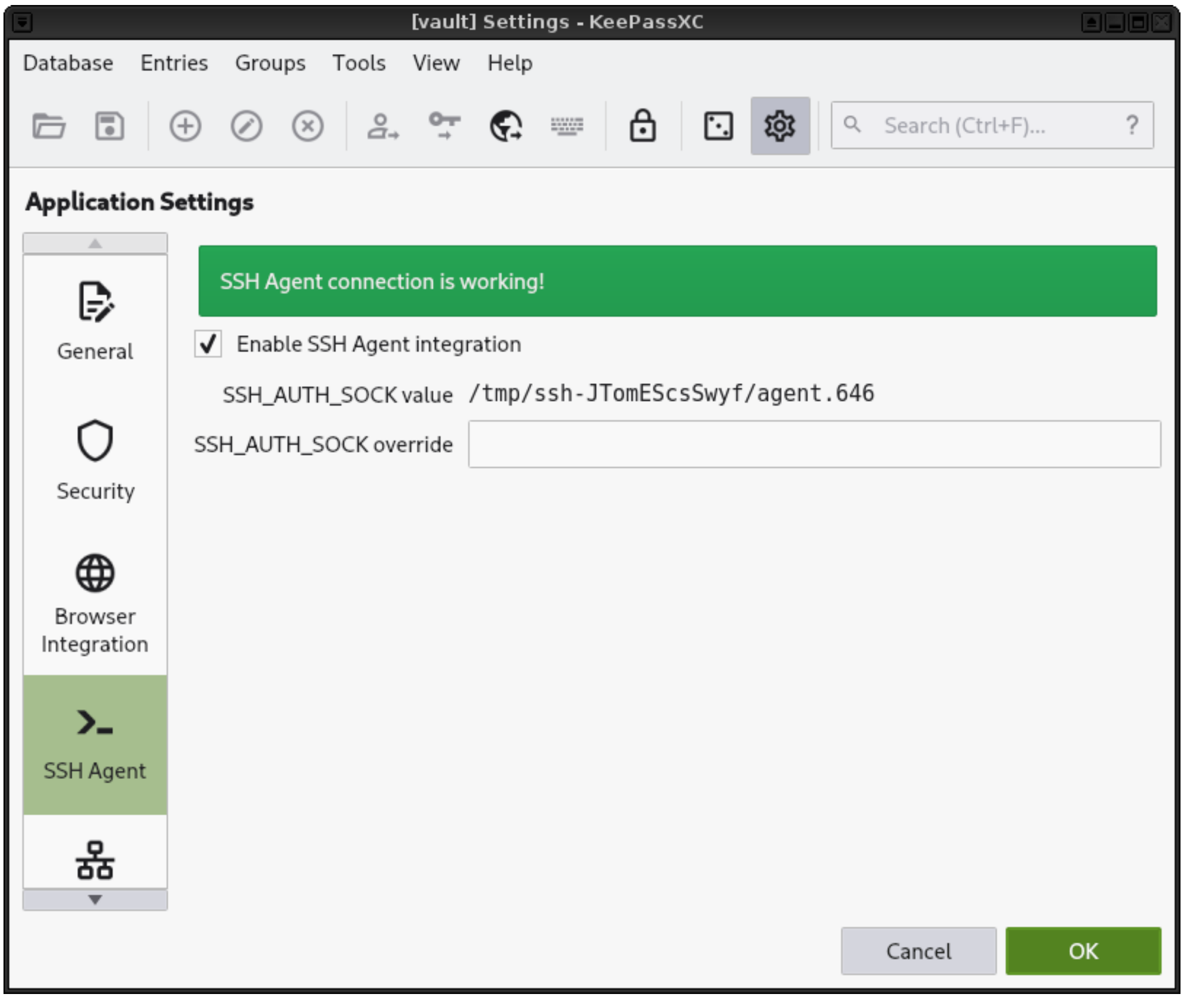
☑ Enable SSH Agent integration

SSH_AUTH_SOCK value `/tmp/ssh-JTomEScsSwyf/agent.646`

SSH_AUTH_SOCK override

10. Restart KeePassXC

11. Check the SSH Agent Integration status.



Application Settings

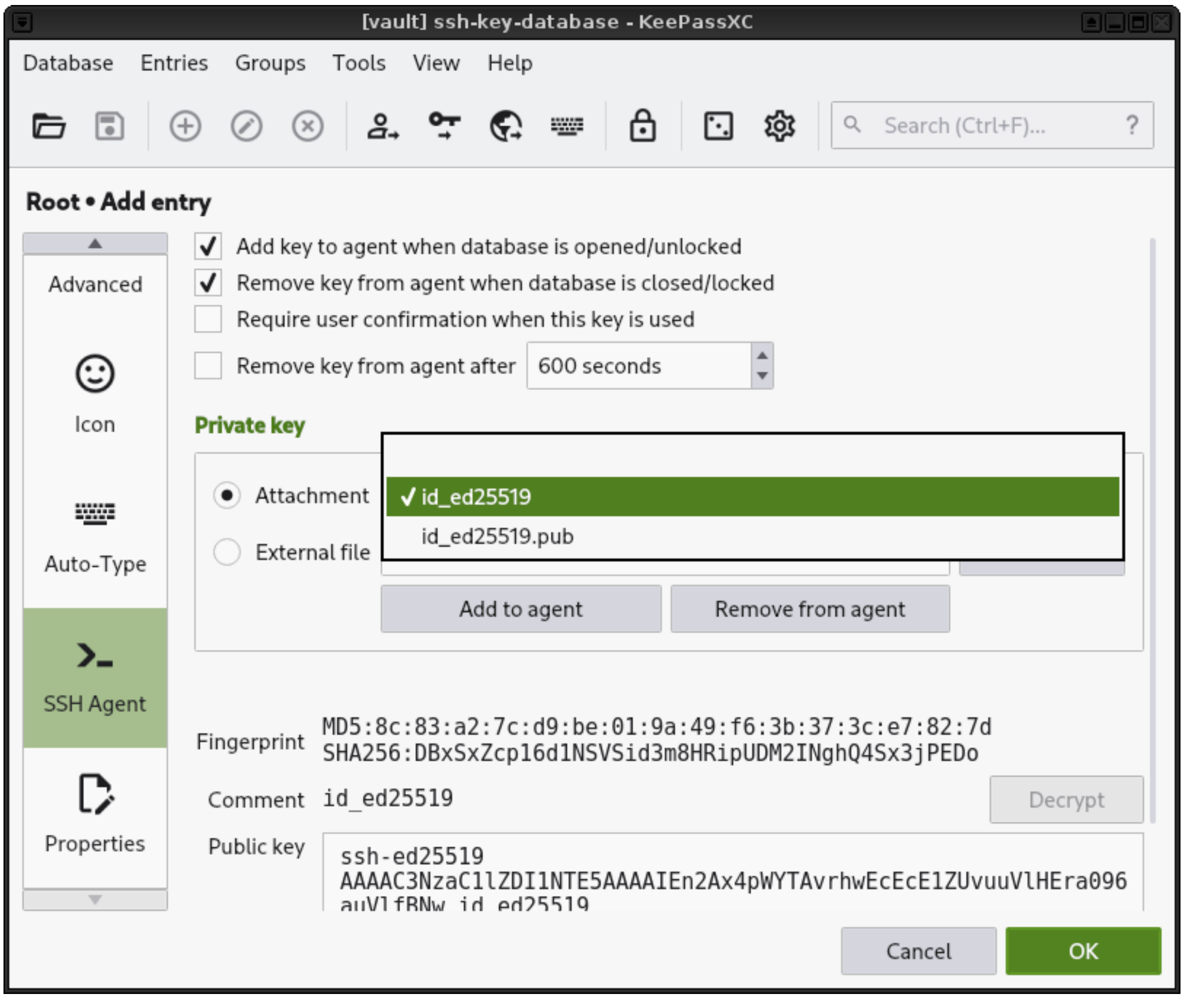
SSH Agent connection is working!

✓ Enable SSH Agent integration

SSH_AUTH_SOCK value `/tmp/ssh-JTomEScsSwyf/agent.646`

SSH_AUTH_SOCK override

12. Open the entry you created and select your private key in the “SSH Agent” section. Don’t forget to also check the first two options.



[✓] Add key to agent when database is opened/unlocked
[✓] Remove key from agent when database is closed/locked
[ ] Require user confirmation when this key is used
[ ] Remove key from agent after 600 seconds

Private key

[● Attachment] ○ External file

✓ id_ed25519
id_ed25519.pub

Fingerprint MD5:8c:83:a2:7c:d9:be:01:9a:49:f6:3b:37:3c:e7:82:7d
SHA256:DBxSxZcp16d1NSVSid3m8HRipUDM2INghQ4Sx3jPEDo

Comment id_ed25519

Public key
```
ssh-ed25519
AAAAC3NzaC1lZDI1NTE5AAAAIEn2Ax4pWYTAvrhwEcEcE1ZUvuuVlHEra096
auVlfBNw id_ed25519
```

Testing the KeePassXC Setup

1. Close your KeePassXC database and run `ssh-add -L` in a `vault` VM terminal. It should return `The agent has no identities.`

```
[user@vault ~]$ ssh-add -L
The agent has no identities.
```

2. Unlock your KeePassXC database and run `ssh-add -L` again. This time it should return your public key.

```
[user@vault ~]$ ssh-add -L
ssh-ed25519 <public key string> user@vault-keepassxc
```

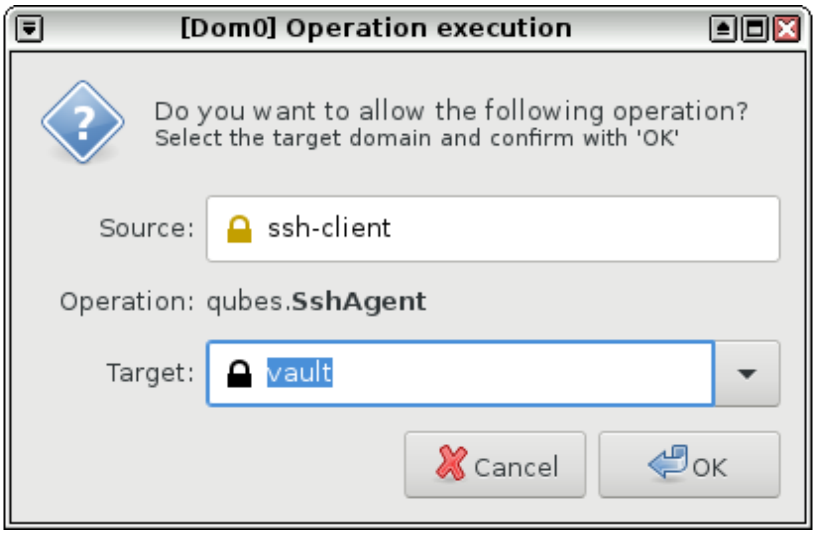
Test Your Configuration

1. If you're using KeePassXC, shutdown KeePassXC on your vault VM. If not, make sure your private key is not added to the ssh-agent in your vault VM (Check with `ssh-add -L`). If it is, restart your vault VM and do not enter your password when it asks you to.

2. Try fetching your identities on the SSH Client VM.

```
ssh-add -L
```

3. Allow operation execution. (If you don't see the below prompt, check your VM interconnection setup.)



[Dom0] Operation execution

Do you want to allow the following operation?
Select the target domain and confirm with 'OK'

Source: ssh-client

Operation: qubes.SshAgent

Target: vault

It should return `The agent has no identities.`.

If you're getting an error (e.g. `error fetching identities: communication with agent failed`), make sure your vault VM is running and check your VM interconnection setup.

4. Launch KeePassXC and unlock your database.

5. Repeat steps 1 and 2.

Check if it returns `ssh-ed25519 <public key string>`

If you're getting an error (e.g. `error fetching identities: communication with agent failed`), make sure your vault VM is running and check your VM interconnection setup.

Current limitations

- It is possible for a malicious VM to hold onto an ssh-agent connection for more than one use. Therefore, if you authorize usage once, assume that a malicious VM could then use it many more times. In this case, though, the SSH Agent will continue to protect your private keys; only usage of it would be available to the malicious VM until it is shut down.
- The two actions below should allow the user to know what is requested before the operation gets approved, but has not been fully verified:
  - configuring the vault to use `ssh-add -c` as described above
  - configuring KeepassXC to "Require user confirmation when this key is used"

Want more Qubes split magic? Check out Split-GPG 37.

This guide has been inspired by:
Qubes Split SSH (Github: Jason Hennessey - henn) https://github.com/henn/qubes-app-split-ssh 10
Using split ssh in QubesOS 4.0 (Kushal Das) https://kushaldas.in/posts/using-split-ssh-in-

qubesos-4-0.html 3
Using Split-SSH in Qubes 4 (Denis Zanin) https://deniszanin.com/using-split-ssh-gpg-in-qubes-os/ 9
R.I.S.K.S. https://19hundreds.github.io/risks-workflow/ssh-split-setup 11

▸ This document was migrated from the qubes-community project