Cake Labs

All Things Technical / Twitch / Infosec / Sysadmin in Recovery.

Managing Multiple SSH Keys on Windows with KeePassXC and OpenSSH

Dec 7, 2021
7 min read
Linux SSH Security

Intro

Over the years I have found myself becoming a collector of SSH keys used for different systems and clients of mine. Many people will start using SSH keys having 1 key for their system, I was there myself. While SSH keys offer greater security compared to passwords, they do not offer perfect security, no solution ever will (not even this one!). There is a chance you will leak private keys (accidental GitHub commit, bad filesystem permissions, show it on a livestream, malware/trojans, and so on). Because of this, I choose to generate SSH keys for specific purposes or clients, thus limiting the “blast radius” a leaked key will have. It’s a lot easier to re-key 2 servers than 25.

Everything, we have compartmentalized our keys but now you have a new problem. Now you have become the digital equivalent of a building superintendent with massive ring of keys and no easy way to keep track of their usage.

I’m going to talk about what I’ve done to solve this issue with security in mind, without disrupting my existing workflow and using the now native Windows OpenSSH service. I wrote this guide with Windows 10/11 in mind, but should also work on Linux and MacOS OpenSSH agents.

Enter KeePass

I’ve been using KeePass for almost a decade now. I love the personal control I have over my vault, and when I started using it in 2012 most of the cloud password managers were not as strong as they are today. Nothing against them, I still advocate for Bitwarden for anyone looking at a simple cloud and free password manager. If you are debating on switching to a password manager (which by the way, you should) and don’t need crazy SSH key control, Bitwarden is fine.

KeePass being Open Source has many forks and client implementations, my favorite and choice for this being KeePassXC.

KeePassXC offers a built-in SSH Agent capable of storing your private keys inside your

Cake Labs

All Things Technical / Twitch / Infosec / Sysadmin in Recovery.

encrypted vault, and only presenting keys to the agent when requested. KeePassXC also offers a nice interface for auto fill browser plugins, but that is out of scope here.

Benefits

I've been using SSH key authentication for a while now, but had limited control over my keys, defaulting to:

1. Using PuTTY + Pageant + Plink
2. Keeping some SSH keys in `~/.ssh/` and configuring ssh-agent to read them
3. Manually selecting a SSH key on every authentication `ssh -i ~/.ssh/id_client cake@clientserver.dev`

These solutions worked, but had some flaws:

1. Required to keep SSH keys lying around on computers, some without passphrases.
2. Mental gymnastics keeping track of keys + locations
3. Not easily portable or synced across multiple systems. I once found myself 1000's of miles away from home without my SSH keys to a degraded production system.

Moving to KeePassXC for SSH key management allowed me to do the following:

- Keys are centralized in 1 database. This is now 1 file to keep track of and keep safe.
- Encrypted at rest, irregardless of if passphrases are on the keys or not.
- Portable. Depending how you sync your KeePass Database, you can have all your keys on multiple machines and always up to date.
- Automation of adding/removing the keys from your systems memory when not needed.

Wait! Here Be Danger

Before we begin, a few things to keep in mind.

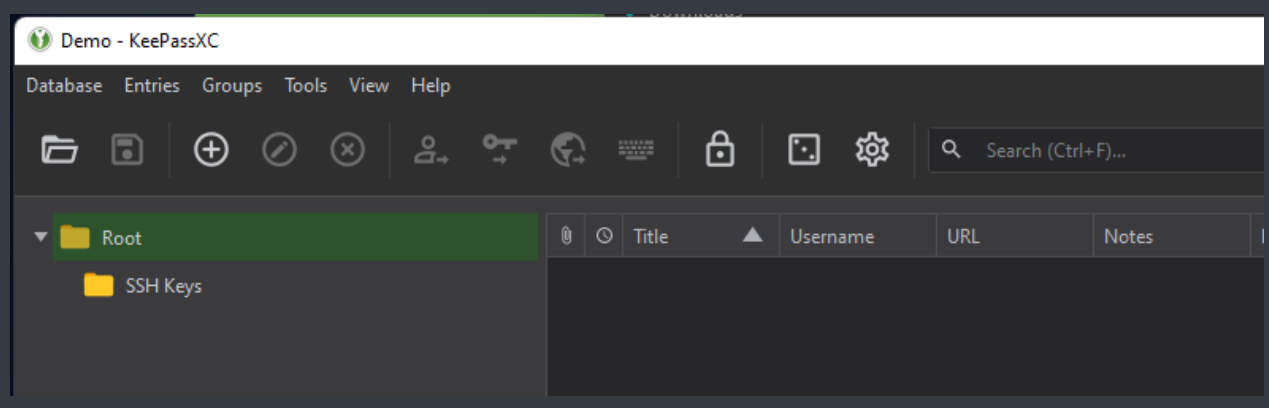
I will be generating new and disposable SSH keys for demonstration. **NEVER show or give your private keys to anyone.**

Great care should be put into protecting and backing up your KeePass database file. Unlike cloud password managers if you lose this file (or access to download it) you are hosed. You should treat this file with a 3-2-1 backup mentality. 3 Copies, 2 Locations, 1 Offline.

Setup

Download the latest KeePassXC and install.

if you already use a KeePass derivative, open your existing vault. New users can create a new vault.



Open Services in Windows. Use Search or the Run box to open `Services.msc`.

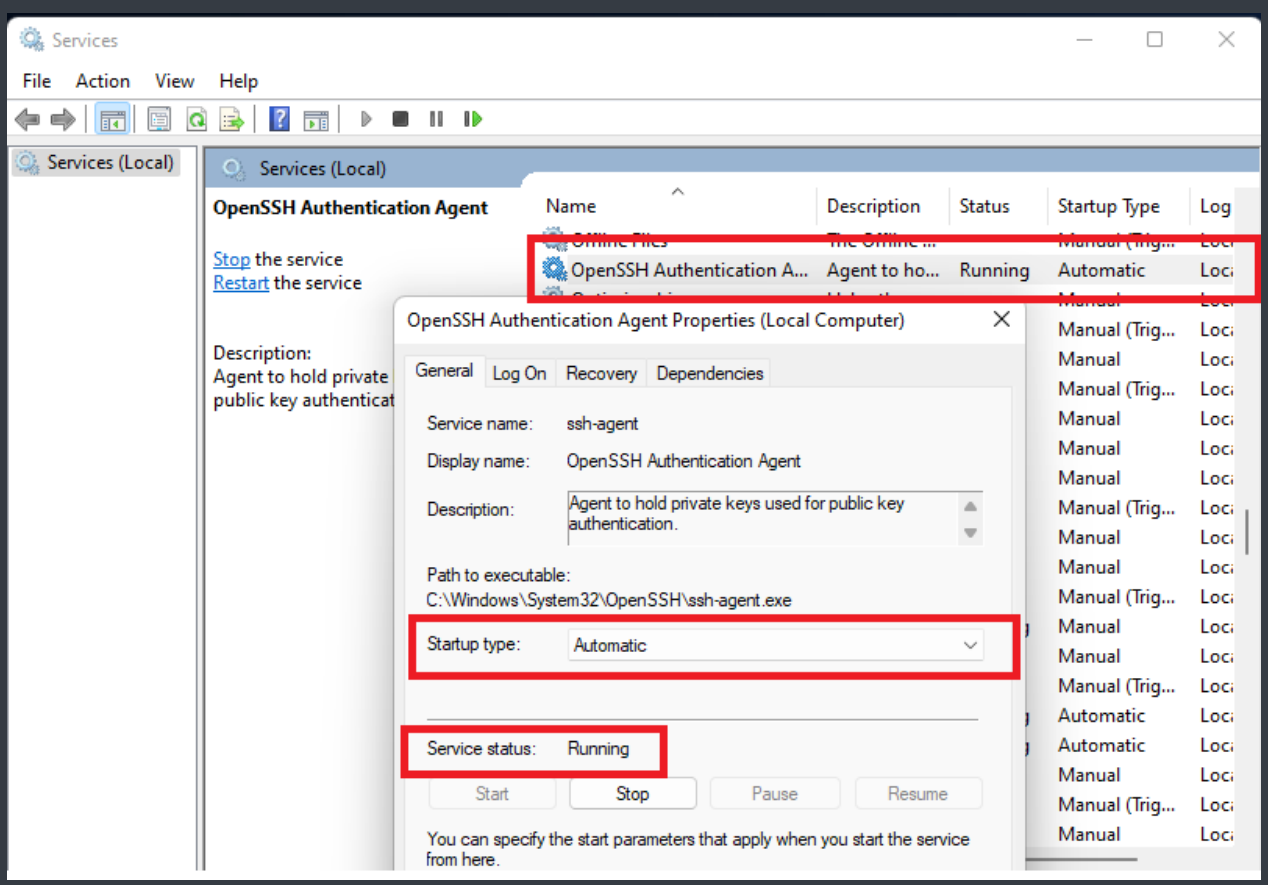
Look for "OpenSSH Authentication Agent" and set Startup type to `Automatic` and start

Cake Labs

All Things Technical / Twitch / Infosec / Sysadmin in Recovery.

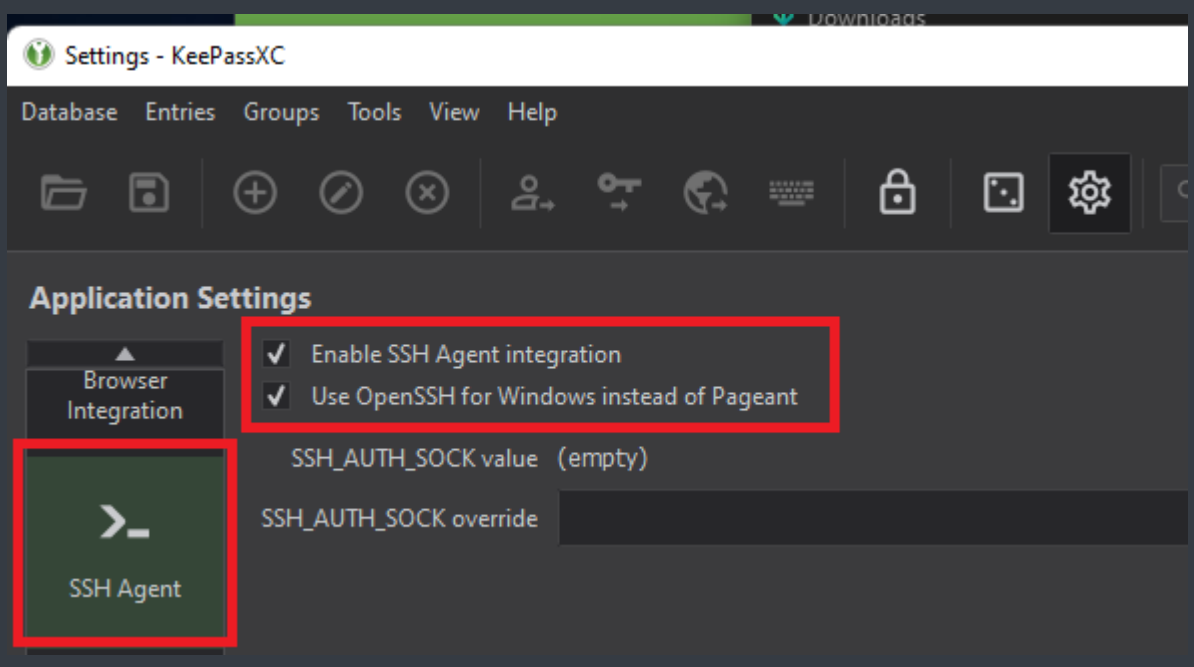
the service.

***If you Don't have the OpenSSH Agent service, you may need to install it first.**



Open the KeePassXC Settings Menu (Tools -> Settings). Select SSH Agent and enable both:

- Enable SSH Agent Integration
- Use OpenSSH for Windows instead of Pageant

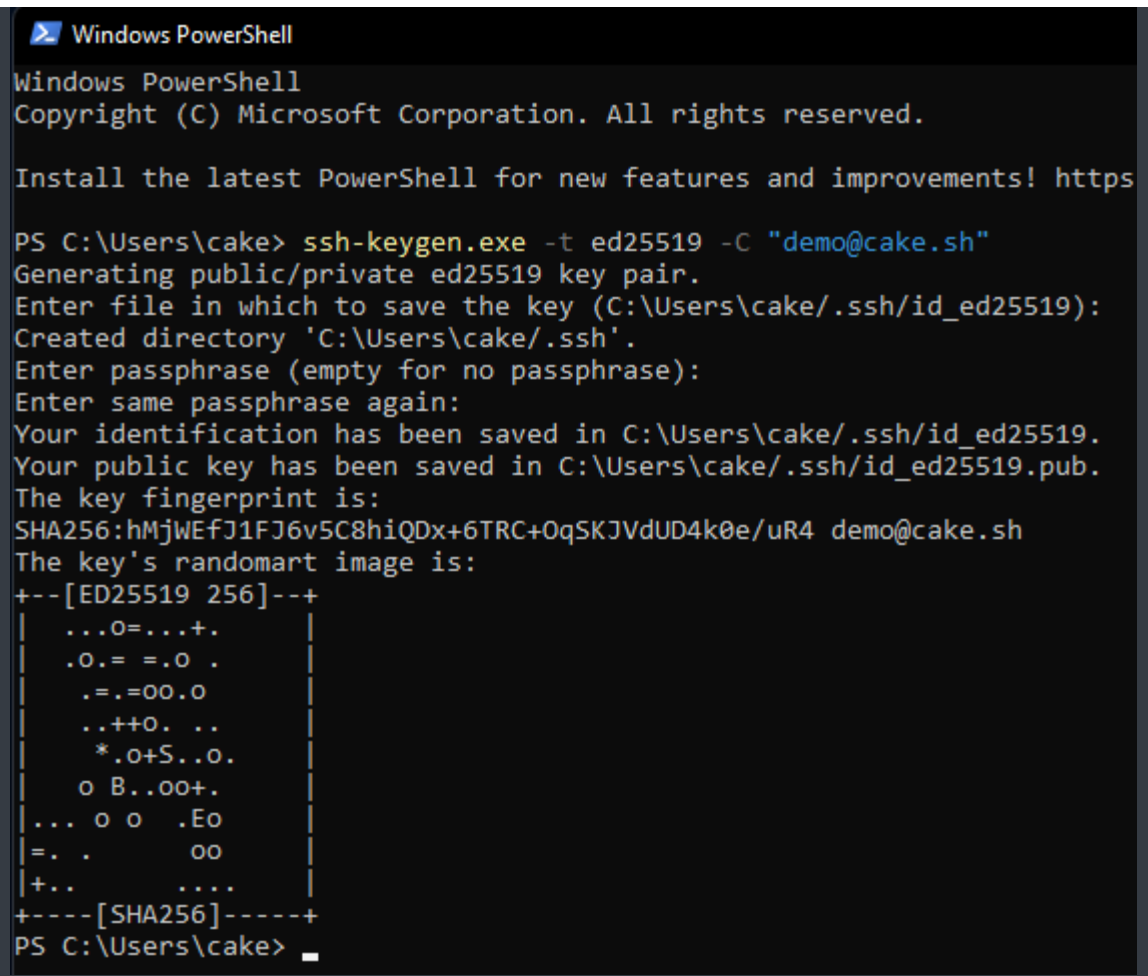


For our example, let's make a new SSH key in a CLI prompt. You can use an existing private key.

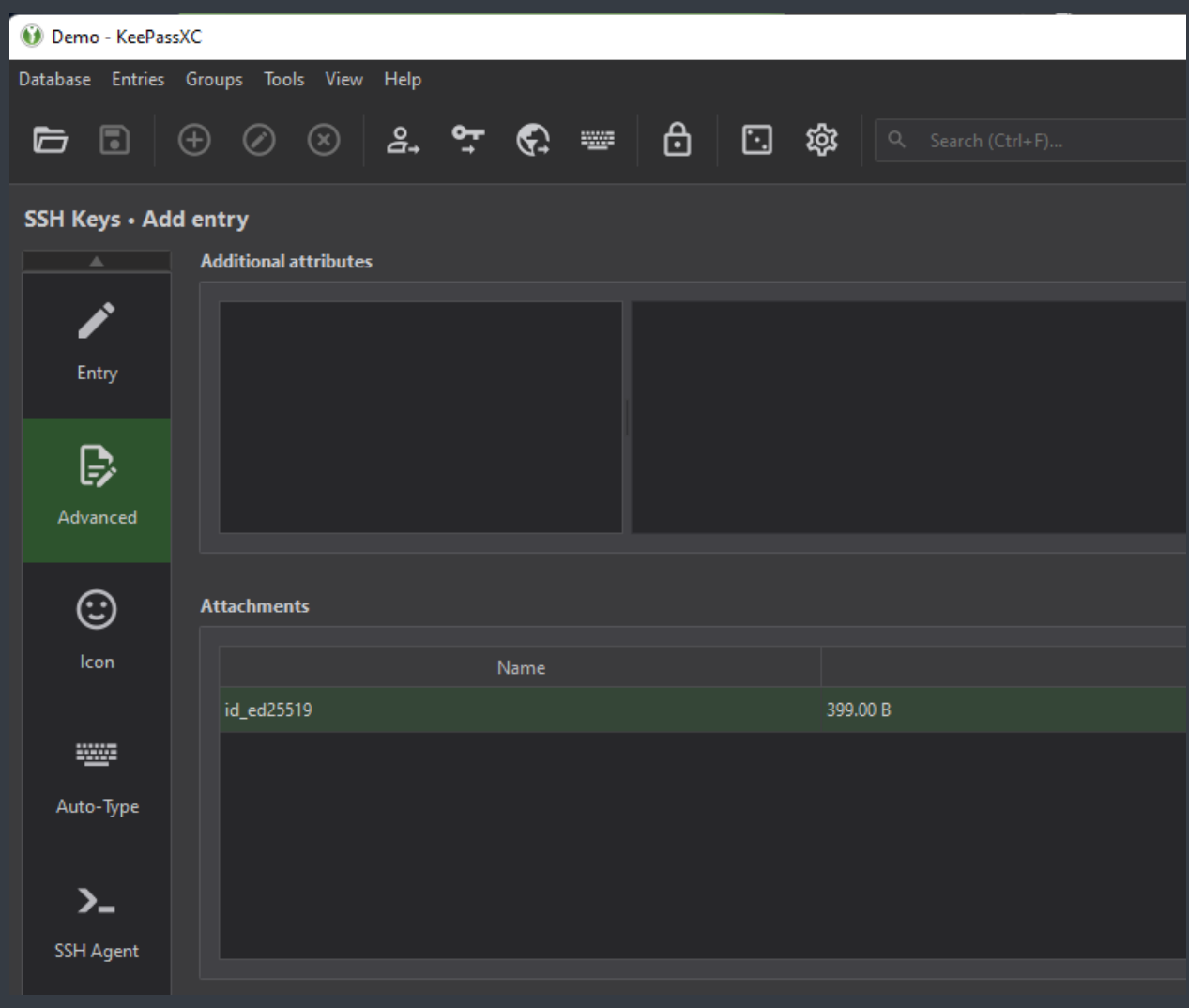
```
ssh-keygen.exe -t ed25519 -C "demo@cake.sh"
```

Cake Labs

All Things Technical / Twitch / Infosec / Sysadmin in Recovery.



Back in KeePassXC, Create a new vault entry and select `Advanced`. Attach your private key to the entry.

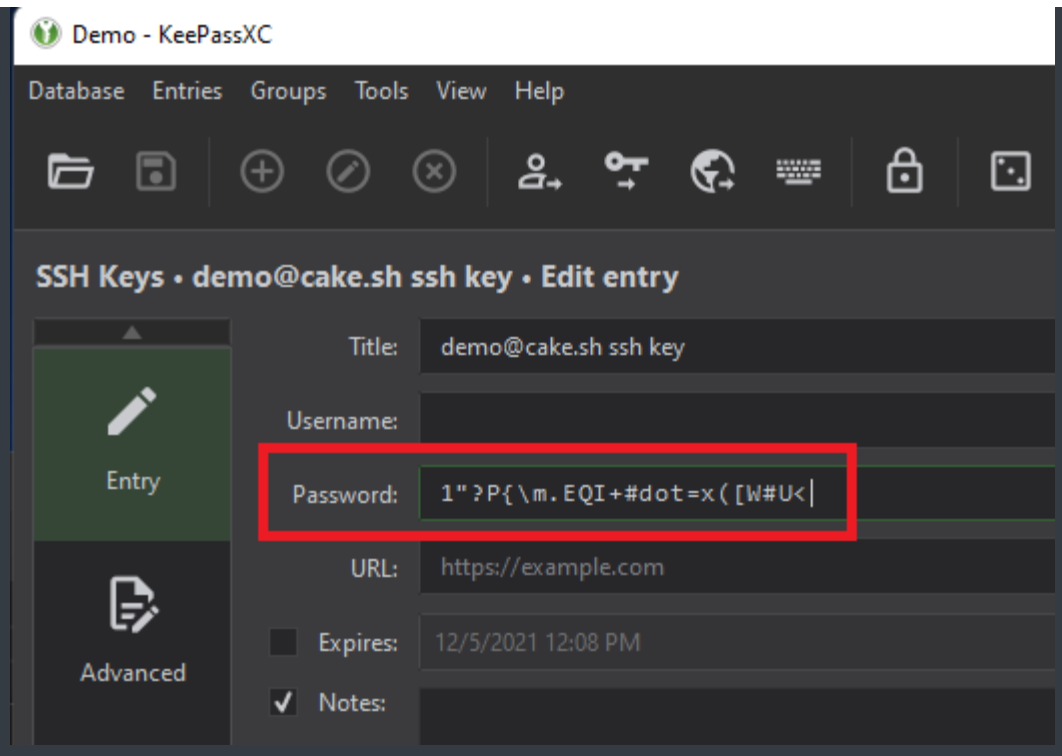


Once attached, go to `SSH Agent` and select your private key from `Attachment`. The public key should now be displayed below if properly attached. Save your entry.

If your private key has a passphrase fill the password in the “Password” field, otherwise keep it blank.

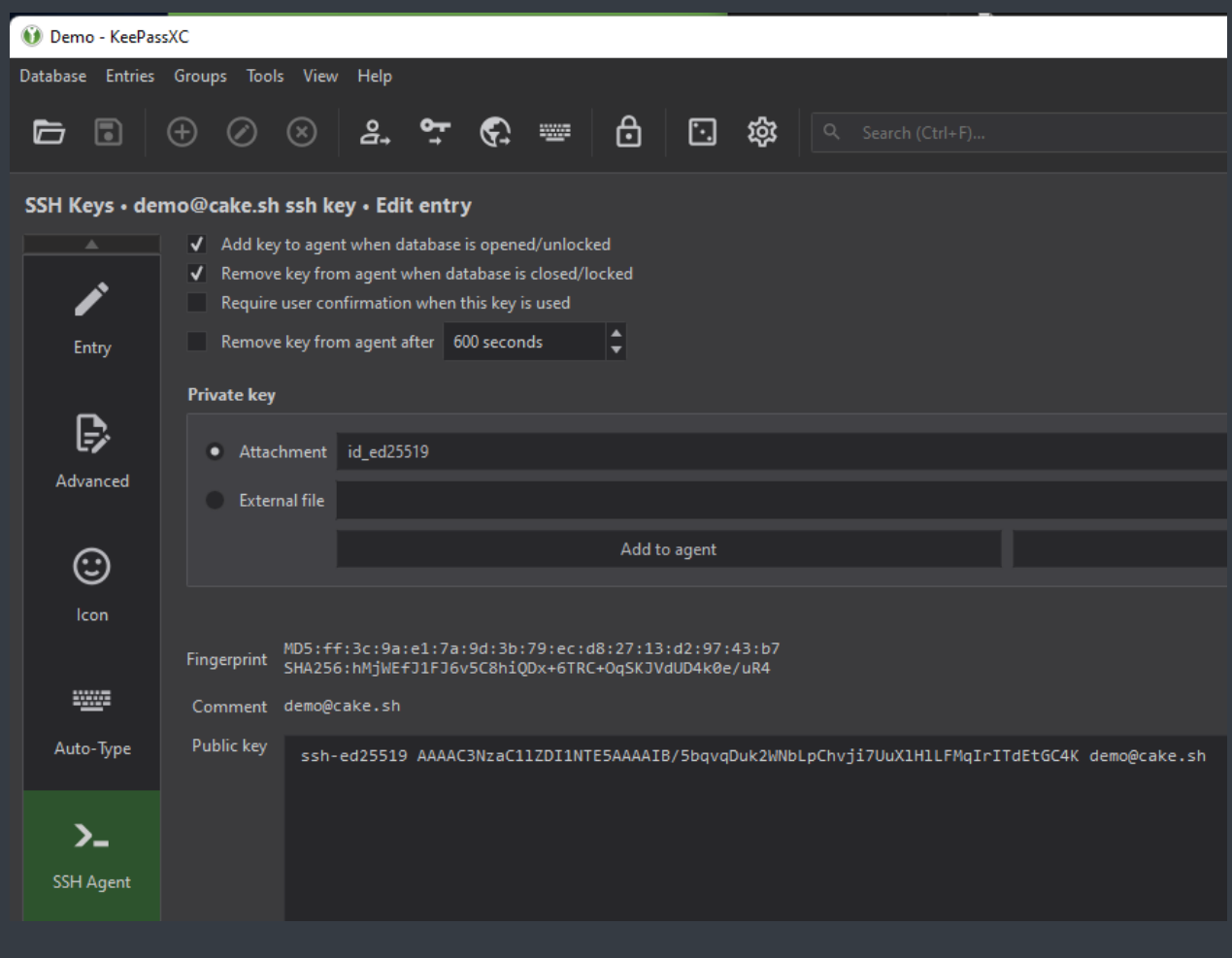
Cake Labs

All Things Technical / Twitch / Infosec / Sysadmin in Recovery.



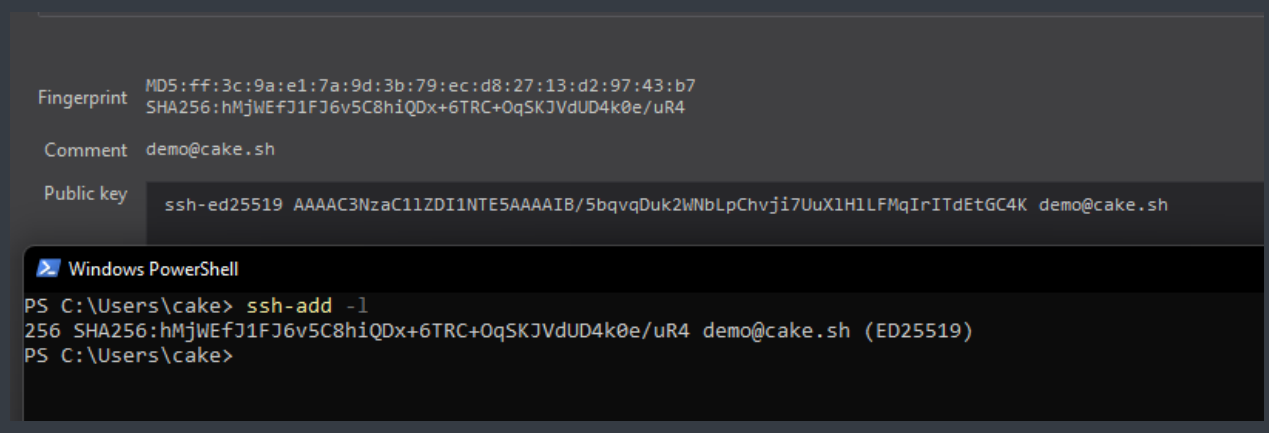
For our test, we are going to check off the following options, ensuring our SSH key is only available when our password vault is unlocked.

- Add key to agent when database is opened/unlocked
- Remove key from agent when database is locked/closed



Let's Use It

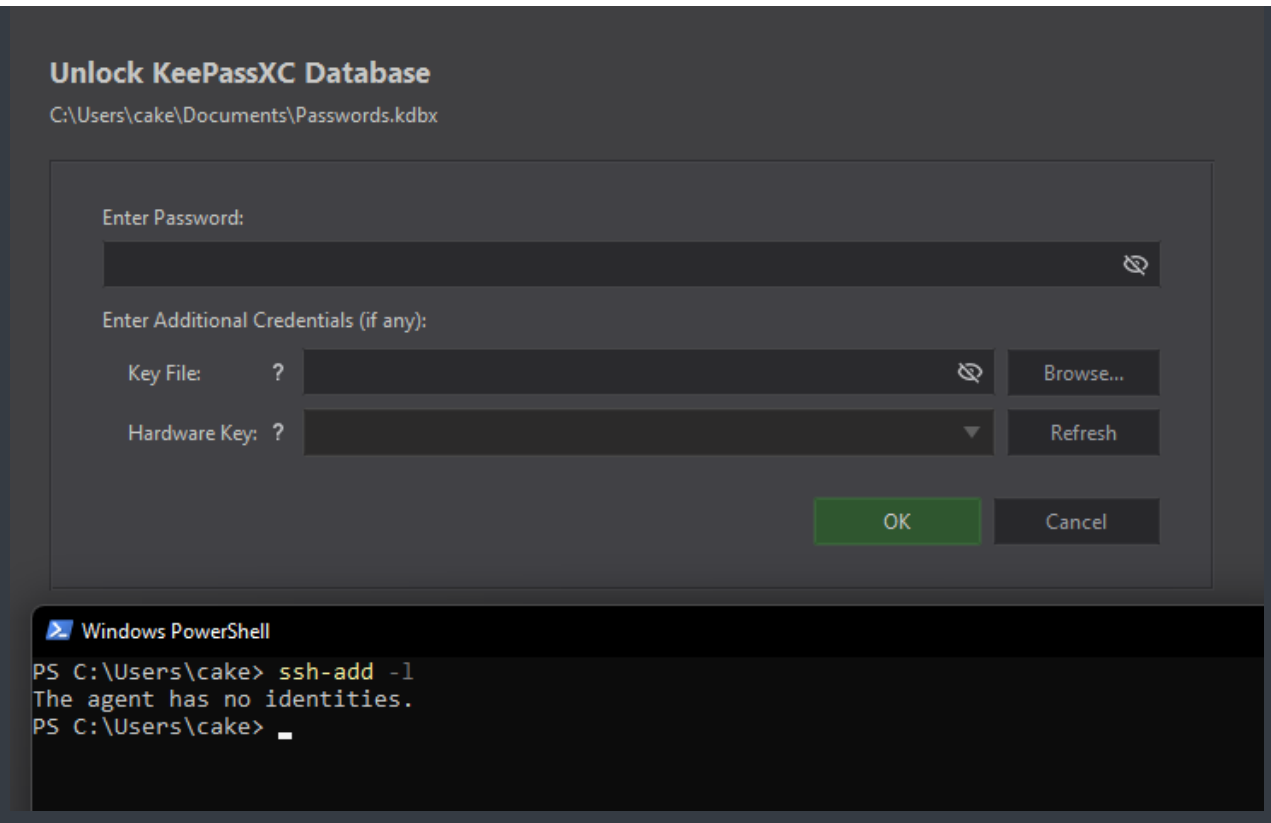
With the KeePass DB unlocked, open a command prompt and run `ssh-add -l` to display keys currently in the SSH agent. You should see your public key with a matching fingerprint in KeePassXC.



Lock your KeePass database and run `ssh-add -l` again. The key has been removed from your SSH agent, no longer available for use until you unlock your database again.

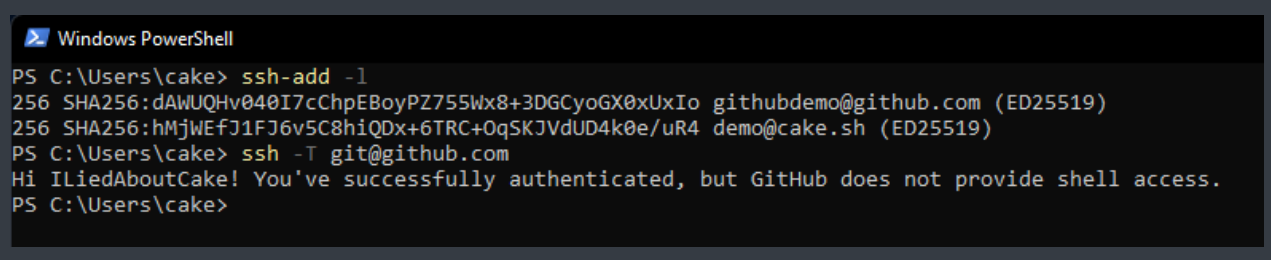
Cake Labs

All Things Technical / Twitch / Infosec / Sysadmin in Recovery.



Multiple Identities

Following in my use case I've added a second key, which is also on my GitHub account. We can test our access to GitHub over SSH with `ssh -T git@github.com` and see I am authenticated through my KeePass keys.



You can keep multiple keys active in your SSH agent, but without proper configuration all active keys will be tried against a host, leading to slow authentication time or fail2ban timeouts. It's best practice to only keep your keys you need active in the SSH agent when you need them. If you only have 1-2 keys, this is not really a problem but can quickly get out of hand. Luckily, we can use our SSH config to control this.

Enter ~/.ssh/config

The SSH client config file allows for easy standardization around commonly used connections. Lets take this example `ssh cake@gateway-us-east1.prod.evilcorp.dev -A -p 41020 -L 9906:127.0.0.1:3306` (SSH to a server as a different user, on a nonstandard port, forward the SSH agent, and tunnel MySQL back to the client). Nobody wants to type this every connection, so instead we will use the SSH config file to simplify things.

```
File: ~/.ssh/config

host evilcorp
    HostName gateway-us-east1.prod.evilcorp.dev
    User cake
    ForwardAgent yes
    Port 41020
    LocalForward 9906 127.0.0.1:3306
```

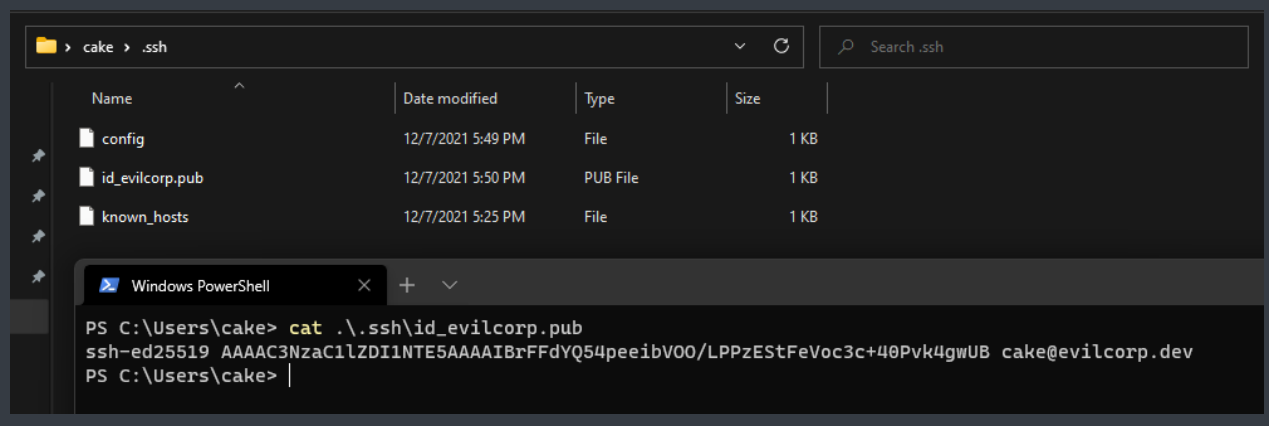
If we connect with `ssh -v evilcorp` we get verbose output of the client/server exchange and can quickly see that our SSH agent is offering all of our keys and even looking for more, which discussed above can cause problems.

Cake Labs

All Things Technical / Twitch / Infosec / Sysadmin in Recovery.

```
debug1: Will attempt key: githubdemo@github.com ED25519 SHA256:dAWUQHv040I7cChpEBoyPZ755Wx8+3DGCyoGX0xUxIo agent
debug1: Will attempt key: demo@cake.sh ED25519 SHA256:hMjWEfJ1FJ6v5C8hiQDx+6TRC+OqSKJVdUD4k0e/uR4 agent
debug1: Will attempt key: cake@evilcorp.dev ED25519 SHA256:wPxKx352U9QbTCSYVOc/iUSlmUlbHMnwKw2IEldp/rM agent
debug1: Will attempt key: C:\\Users\\cake/.ssh/id_rsa
debug1: Will attempt key: C:\\Users\\cake/.ssh/id_dsa
debug1: Will attempt key: C:\\Users\\cake/.ssh/id_ecdsa
debug1: Will attempt key: C:\\Users\\cake/.ssh/id_ed25519
debug1: Will attempt key: C:\\Users\\cake/.ssh/id_xmss
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,ssh-rsa,rsa-sha2-256,rsa-sha2-512,ssh-dss,ecdsa-sha2-nistp256
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering public key: githubdemo@github.com ED25519 SHA256:dAWUQHv040I7cChpEBoyPZ755Wx8+3DGCyoGX0xUxIo agent
debug1: Authentications that can continue: publickey,password
debug1: Offering public key: demo@cake.sh ED25519 SHA256:hMjWEfJ1FJ6v5C8hiQDx+6TRC+OqSKJVdUD4k0e/uR4 agent
debug1: Authentications that can continue: publickey,password
debug1: Offering public key: cake@evilcorp.dev ED25519 SHA256:wPxKx352U9QbTCSYVOc/iUSlmUlbHMnwKw2IEldp/rM agent
debug1: Server accepts key: cake@evilcorp.dev ED25519 SHA256:wPxKx352U9QbTCSYVOc/iUSlmUlbHMnwKw2IEldp/rM agent
debug1: Authentication succeeded (publickey).
Authenticated to gateway-us-east1.prod.evilcorp.dev ([                ]:41020).
```

Expanding on our config above, we can add the `IdentityFile` and `IdentitiesOnly` directive to guide the exchange to the correct key in our agent. To do this, copy your public key out of KeePassXC to the ~/.ssh/ folder as a file. You only need to include your public key and **do not put your private key in this export**, or you have defeated this entire blog post.



And update your config:

```
File: ~/.ssh/config

host evilcorp
    HostName gateway-us-east1.prod.evilcorp.dev
    User cake
    ForwardAgent yes
    Port 41020
    LocalForward 9906 127.0.0.1:3306
    IdentityFile ~/.ssh/id_evilcorp.pub
    IdentitiesOnly yes
```

Now if we `ssh -v evilcorp` we can see a single key offered and accepted in the exchange. No more login failed noise! Your Security Operations Center thanks you :)

```
debug1: Will attempt key: C:\\Users\\cake/.ssh/id_evilcorp.pub ED25519 SHA256:wPxKx352U9QbTCSYVOc/iUSlmUlbHMnwKw2IEldp/rM explicit agent
debug1: SSH2_MSG_EXT_INFO received
debug1: kex_input_ext_info: server-sig-algs=<ssh-ed25519,ssh-rsa,rsa-sha2-256,rsa-sha2-512,ssh-dss,ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ec
debug1: SSH2_MSG_SERVICE_ACCEPT received
debug1: Authentications that can continue: publickey,password
debug1: Next authentication method: publickey
debug1: Offering public key: C:\\Users\\cake/.ssh/id_evilcorp.pub ED25519 SHA256:wPxKx352U9QbTCSYVOc/iUSlmUlbHMnwKw2IEldp/rM explicit agent
debug1: Server accepts key: C:\\Users\\cake/.ssh/id_evilcorp.pub ED25519 SHA256:wPxKx352U9QbTCSYVOc/iUSlmUlbHMnwKw2IEldp/rM explicit agent
debug1: Authentication succeeded (publickey).
```

It's worth nothing that even when limiting your SSH identity to a single key when signing in, if `ForwardAgent yes` or `ssh -A` is used, you will still forward all keys in your agent. You should only use SSH agent forwarding on servers you trust as rogue admins or compromised servers can use your agent to impersonate you while connected.

Final Thoughts

Windows native support for SSH has come a long way, and the days of needing PuTTY + Pageant + Plink are long gone. Coupling KeePassXC for secure private key storage with native OpenSSH tooling is a strong beast, allowing quick workflows to develop without compromising on security. Fumbling through KeePass documentation and StackOverflow posts on SSH configs was frustrating, I hope my guide changes that.