yubico

The GitHub guide to commit signing

Secure your software supply chain with GitHub and YubiKeys.

Part one: Overview

For many organizations today, the question isn't whether you're using open source code, but how much. It's common for projects to use hundreds of open source dependencies, or code and other software components that are free for anyone to use. These components help make up your "software supply chain." Like a physical supply chain, a **software supply chain** is anything that goes into or affects your code from development, through your CI/CD pipeline, until it gets deployed into production. It includes both open source components and their source—including repositories and package managers.

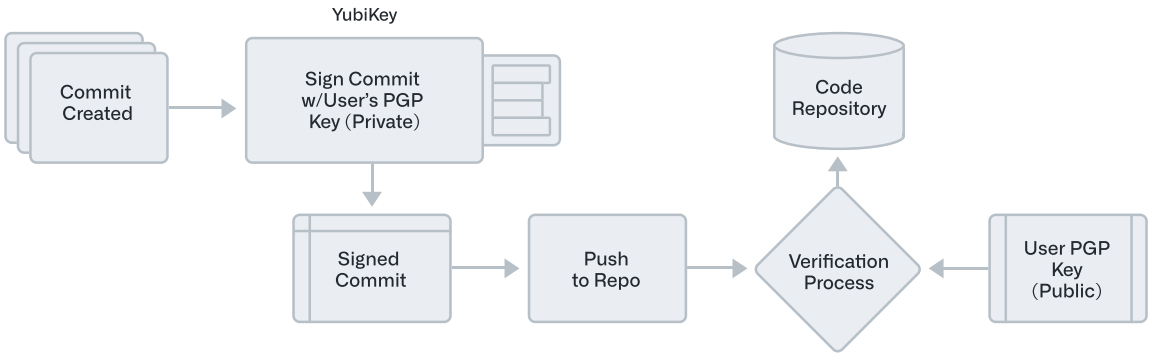
Understanding what's in your supply chain and where it came from is a critical part of application security. Organizations approach this challenge using a variety of authentication tools and processes, starting with commit signing, code signing, verified commits. All three are tied to a developer's unique "digital signature," which can be authenticated using GPG keys stored on physical devices like YubiKeys. In case of a breach, you can quickly filter the impacted code, trace who introduced it to your code base, and review it.

It all comes down to trust. Signed commits and tags can give you, your teams, and your business peace of mind. In this guide, we'll walk through how to set up commit and code signing with GitHub. You'll also learn how to use digital signatures with the additional security layer of a portable YubiKey, which can increase the validity of the signature and confidence in your code. Lastly, we'll show you how to secure your GitHub account by moving the sensitive part of your SSH key from your computer to an external key, so you can access your GitHub account and the GitHub API without supplying a username and personal access token.

Team tips: SSH with FIDO YubiKeys requires OpenSSH 8.4p1, supported by macOS, Linux, and Windows. To meet the minimum requirements, we recommend Windows users install Git Bash. Learn more in the latest OpenSSH release notes.

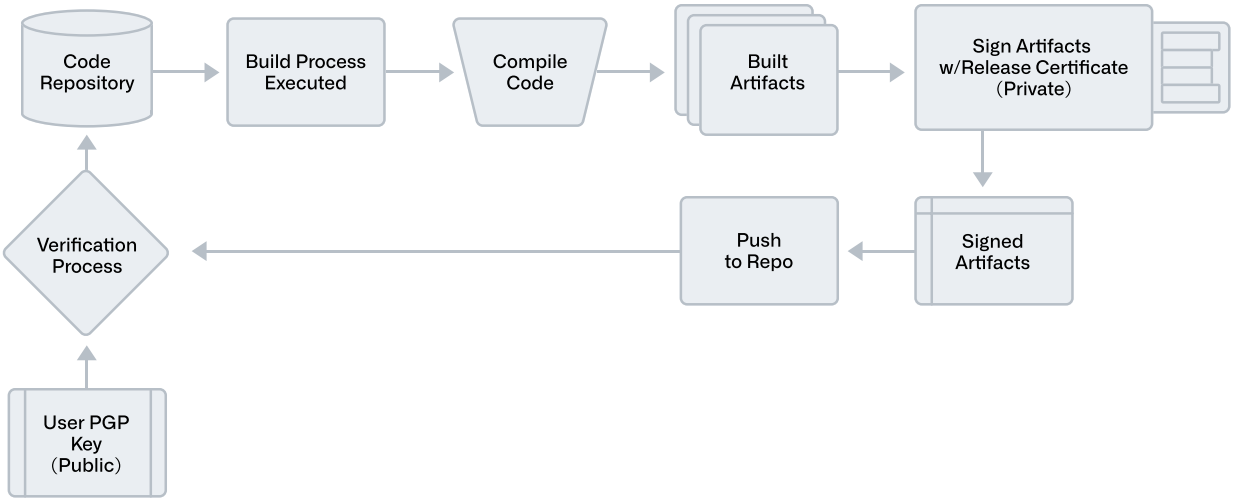
I. Commit signing

Commit signing is when developers digitally sign code commits to a code repository. When a developer signs a commit, they create an auditable trail of information that helps establish provenance of commits. This type of signature is generally done with GPG keys, but can be done with other types of key-pairs like x509.



II. Code signing

Code signing happens several times during the software release process. This is the action of signing build artifacts (exe, dll, jar, war, apk) that will be deployed. It's generally done with a x509 code signing certificate with code signing attributes set, and allows client machines to verify the code with a trusted root authority by verifying the signature.



III. Verification process

To make commit signing part of your security workflow, **start by setting some branch protection rules**. This way contributors and bots can only push commits that have been signed and verified to the branch. Using branch protection rules and requiring commit signing provides an additional layer of security and can increase confidence in your code's integrity.

As a best practice, it's important to verify your code at different times during the software development process. When and how often depends on your organization. Some have more security and compliance requirements than others, like financial institutions or government agencies. Multifactor authentication adds yet another layer of verification when securing the software supply chain is critical.

Part two: Configuration

I. GPG key creation

The first step in the commit signing process is to create a set of GPG keys on your security key. (You can find **a helpful how-to on Yubico's website**.) As you follow the directions, make sure your personal information is all correct—if it's wrong, it'll impact GitHub's verification process.

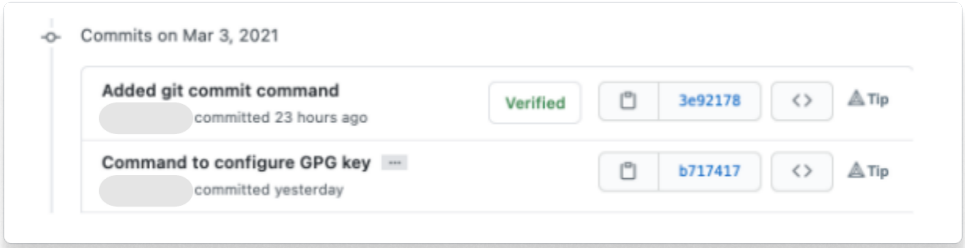
GPG keys are referenced in the key ID. For our purposes, let's say the key ID is C6298BB96A78D4F8.

Team tips: Email addresses configured on your GPG key must match at least one email address on your GitHub account.

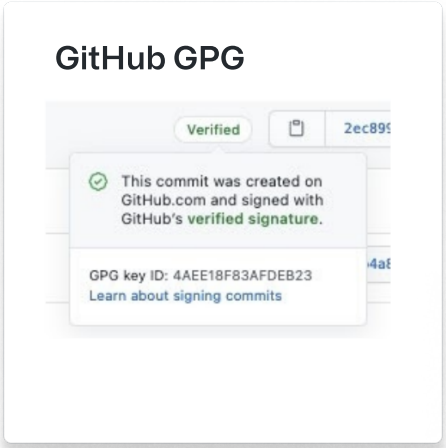
II. GitHub account settings

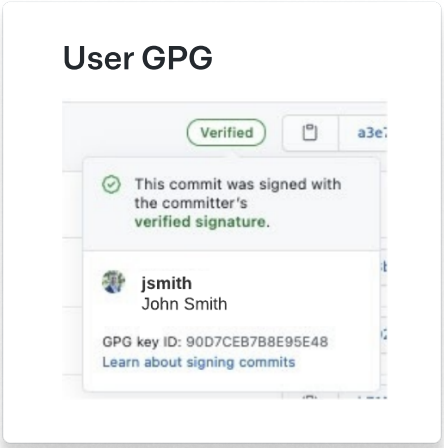
On GitHub, you'll know if code committed was successfully signed by a GitHub user because it'll have a green "Verified" icon. Here's an example of one commit with the icon and one without. Next, we'll show you how to get your commits verified.

Team tips: If you use the GitHub web interface to edit your code or use squash commits, those commits will also show as "Verified" even if you don't have your GPG key configured. This is because GitHub creates a signature for you using our own GPG key—and your application authentication provides you access to this "Verified" icon. In this case the "Verified" indicator only verifies that the commit came from you, not that the code was signed by your personal GPG key.



Another tip: If you click on the "Verified" button it will tell you the source of the verification.





Need to create a GitHub account? Here's how to set it up.

Step 1

Go into the Account Settings page and click on **SSH and GPG keys**

Step 2

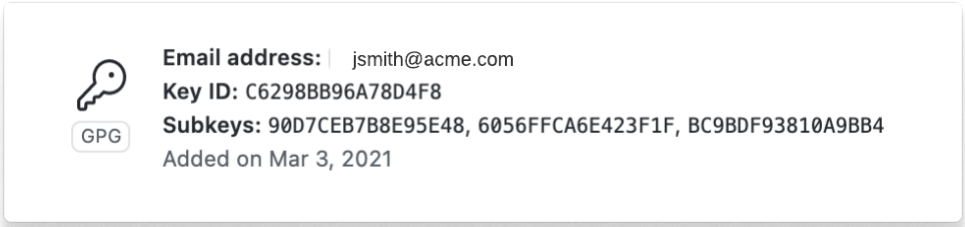
Click on the **New GPG key**

Step 3

Upload the public key that was created from your GPG keys

Step 4

Verify the information that was uploaded



Once you set this up, your commits with signatures will be verified with our registered GPG public key.

III. Local Git configuration

Team tips: This section assumes you have already installed the required applications to utilize your GPG key. To see if you have, open the command line and type “gpg —card-status” with your YubiKey plugged in. If the data returned provides information on your YubiKey, you’re all set.

To use the GPG key, your local Git configuration needs to be updated.
To add it to the global configuration for Git, use the following command.

```
git config --global user.signingkey C6298BB96A78D4F8
```

Another optional command you can use is the gpgsign true. This will make sure all commits are signed by default.

```
git config --global commit.gpgsign true
```

Team tips: This guide is only focusing on command line Git implementations. If you’re using a GUI tool to interface with Git, please check that tool’s documentation for configuration updates.

IV. Signing commits

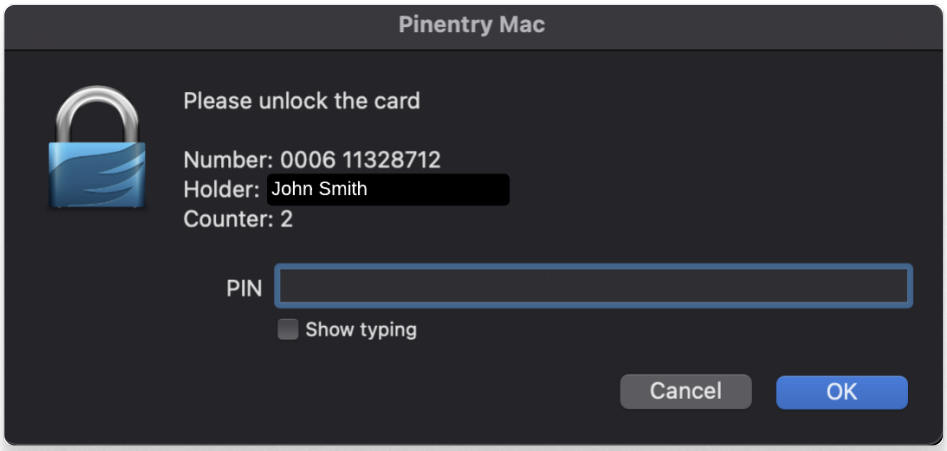
Once you're done, you'll be ready to sign a code commit. Let's create a branch called `gpgsign` to test this out:

```
git checkout -b gpgsign
```

If you didn't set "commit.gpgpsign" to "true" you'll need to add the "-S" switch to each of your Git commit commands.

```
git commit -a -S -m "Signing my first commit"
```

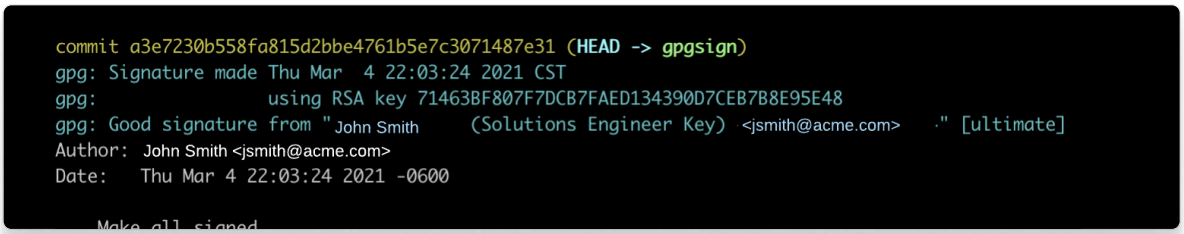
At this point, if you haven't unlocked your GPG key with a pin, you'll be prompted to submit your pin.



From there, the screen will show your commit message. This is also a good time to review the logs to verify everything is correct.

```
git log --show-signature -1
```

The information returned will show details about the signer of the commit and will indicate the commit is ready to be pushed to GitHub.



The following command will then push the updates to a branch named gpgsign in GitHub.

```
git push origin gpgsign
```

Now, your commit should have a “Verified” icon. If this worked correctly, you should be able to return to your main branch and sign commits for it!

V. Using security keys for SSH

Team tips: SSH with FIDO YubiKeys requires OpenSSH 8.4p1, supported by macOS, Linux, and Windows. For Windows users, we recommend using Git Bash.

While storing GPG keys in your YubiKey is great for securely signing your commits, you can also use your YubiKey to quickly and securely authenticate when using Git. This moves the sensitive part of your SSH key from your computer to an external key, so you can access Git data without supplying a username and personal access token every time. Here’s how to get set up.

Getting started

1. First generate a key pair

```
$ ssh-keygen -t ecdsa-sk
```

2. This will create two files in your SSH directory. The first is id_ecdsa_sk.pub, which is a normal OpenSSH public key file whose contents you'll need to paste into the new SSH key form on GitHub. The second is id_ecdsa_sk, which would usually contain the corresponding private key, but in this case it'll contain a "key handle" that references the security key. You'll need to copy the id_ecdsa_sk file to each computer where you want to use this SSH key. Or, if your security key supports it, you can use a FIDO2 resident key (also known as a discoverable credential).

Using resident keys

3. If your security key supports FIDO2 resident keys, like the YubiKey 5 Series, you can use it when creating your SSH key:

```
$ ssh-keygen -t ecdsa-sk -O resident
```

This works the same as before, except a resident key is easier to import to a new computer because it can be loaded directly from the security key.

4. To use the SSH key on a new computer, make sure you have ssh-agent running and simply run:

```
$ ssh-add -K
```

This will load a "key handle" into the SSH agent and make the key available for use on the new computer. This works great for short visits, but it won't last forever–you'll need to run ssh-add again if you reboot the computer, for example. To import the key permanently, instead run:

```
$ ssh-keygen -K
```

This will write two files into the current directory: id_ecdsa_sk_rk and id_ecdsa_sk_rk.pub. Now you just need to rename the private key file to id_ecdsa_sk and move it into your SSH directory:

```
$ mv id_ecdsa_sk_rk ~/.ssh/id_ecdsa_sk
```

Part three: Conclusion

Now that you know how to secure SSH access and sign and validate commits, you are ready to level up and head on over to GitHub Advanced Security to learn how to scan your application code and create policies for your software dependencies. (Don't forget to document best practices for your team!)

> Questions about GitHub integrations or security?
> We're here to help. Contact our Sales Team to learn more.