Securing Production Config Files

Steve Hord edited this page on 25 Jan · 3 revisions

Introduction

Once you have split out your production config, you may want to encrypt any files that contain sensitive information (API Keys, passwords etc). This page will get you started with git-crypt , which works well with node-config. With git-crypt, you can keep your production config files under version control with git, while also keeping them secure via encryption.

Here is the scenario we'll be configuring:

- You, Paul and Ringo are all working on a project, using git. There are server deployments involved.
- Ringo is a contributor, and doesn't need or want to see production passwords, API secret keys etc.
- You and Paul both maintain the servers and need access to the sensitive info.

Prerequisites:

- git-crypt is installed.
  - On OS X : `brew install git-crypt`
  - On Ubuntu / Debian (since 16.04 LTS) `sudo apt-get install git-crypt`
  - On other systems: https://github.com/AGWA/git-crypt/blob/master/INSTALL.md
- Your project is using git, your working copy is clean, and you haven't committed any sensitive information to it yet.
  - If you *have* accidentally committed sensitive information to your repository, you'll need to clean it using bfg repo cleaner.
- You have GnuPG installed.
  - On OS X, the GPG Keychain that is part of GPG Suite is very easy to use.
  - On Windows, GPG4Win looks pretty good.
  - On Unix/Linux, GPA provides a front-end.
- You have a `config` folder where your configuration files will be kept.

Steps

Initial Setup

We're going to set up git-crypt so that the `production.json` file is encrypted for Ringo, but the same file is plain JSON for yourself and Paul.

1. Initialise the repository to use git-crypt

   ```
   git-crypt init
   ```

2. Add your own public GPG key (already on your GPG keychain) as a trusted user

   ```
   git-crypt add-gpg-user john@example.com
   ```

3. Import Paul's public GPG key to your own GPG keychain (Paul probably emailed this to You)

   ```
   gpg --import Paul_pub.gpg
   ```

4. Mark both your own key and Paul's key with 'ultimate' trust in GPG.

   ```
   gpg --edit john@example.com  (or) gpg --edit-key john@example.com
   gpg> trust

           Please decide how far you trust this user to correctly verify other
   users' keys
   (by looking at passports, checking fingerprints from different sources, etc.)

     1 = I don't know or won't say
     2 = I do NOT trust
     3 = I trust marginally
     4 = I trust fully
     5 = I trust ultimately
     m = back to the main menu

   Your decision? 5
   Do you really want to set this key to ultimate trust? (y/N) y
   ```

   Do the same for `paul@example.com`

5. Add Paul's public GPG key as a trusted git-crypt user

   ```
   git-crypt add-gpg-user paul@example.com
   ```

6. Add the `config/production.json` file (or equivalent YAML etc) to the `.gitattributes file`, so git-crypt will manage the encryption / decryption where necessary. Add the following line to the `.gitattributes` file (substituting the appropriate extension):

   ```
   config/production.json filter=git-crypt diff=git-crypt
   ```

7. Now create the file and add it to your repo (substituting the appropriate syntax and extension):

   ```
   echo '{foo:"secret"}' > config/production.json
   git add config/production.json
   git commit -m "production settings"
   git push origin
   ```

On the git remote (server), `config/production.json` is now encrypted.

Git-crypt Workflow

Initially, when Paul clones or pulls the repo, `config/production.json` is encrypted for him too.

But his public key is listed as trusted, and he has the corresponding private key on his keyring. So he can unlock all encrypted files with a single command:

```
git-crypt unlock
```

Paul is prompted for his private key password.

From now on, Paul's and your own workflow is unchanged from normal.

On your machine and Paul's, `config/production.json` is just a normal JSON file.

From Ringo's point of view, `config/production.json` is a binary file.

Mimicking production mode

Ringo might have the need to run the application with `NODE_ENV=production` e.g. to invoke minification, or to debug some issue on the production server.

But Ringo has a problem: when he runs the node program with `NODE_ENV=production` , node-config tries to read `config/production.json` , which on his machine is not a valid JSON file since it's encrypted. The app crashes.

To cater for this, Ringo can specify that any git-crypt encrypted files are skipped, via a `CONFIG_SKIP_GITCRYPT` environment variable, for example:

```
NODE_ENV=production CONFIG_SKIP_GITCRYPT=1 node foo.js
```

This tells node-config to continue even if a git-crypt file is encountered.

Deployment Servers, Build Servers, CI Servers

Git Clone Once Only

Unless you're cloning the git repository every time, you need only unlock it once.

You can create a GPG key for a server (or set of servers) so a single `git-crypt unlock` command is all that is needed. Of course, the corresponding private key must be installed on the server's GPG keychain already.

Git Clone Every Time

In the case where your server *is* performing a git clone every time, you can instead export a symmetric (non-GPG) key from an unlocked git-crypt repository:

```
git-crypt export-key ~/Desktop/git-crypt-key
```

You can then use that symmetric key to unlock the repository from your server without a pass phrase.

```
git-crypt unlock /path/to/git-crypt-key
```