drewfarris / **sample-cfssl-ca**

Sample Cloudflare SSL (CFSSL) Certificate Authority

#pki #tls #ssl #ssh-keys

5 commits **1** branch **0** releases **2** contributors BSD-2-Clause

Branch: master New pull request Create new file Upload files Find file Clone or download

wsargent and **drewfarris** Flesh out client keystore and truststores (#1) … Latest commit `fff812f` on Aug 27

.gitignore	Initial commit	a year ago
LICENSE	Added BSD 2-clause "simplified" license	2 months ago
README.md	Slight mod to pkcs8 generation instructions	a year ago
ca-config.json	Initial commit	a year ago
ca-csr.json	Initial commit	a year ago
client-csr.json	Initial commit	a year ago
doit.sh	Flesh out client keystore and truststores (#1)	2 months ago
server-csr.json	Initial commit	a year ago

README.md

Sample Cloudflare SSL (CFSSL) Certificate Authority

This repository contains configuration files and command-line examples for generating a Certificate Authority and Certificates using Cloudflare's PKI and TLS toolkit, otherwise know as CFSSL, available at https://github.com/cloudflare/cfssl

DISCLAIMER:

1. **Use at your own risk**
2. **Do not use this for production systems**

I created this repository for personal use and for sharing with folks that need to generate Certificate Authorities and Certificates for **testing purposes only**. Much of this is scattered across the web and I got tired of Googling all the time and needed to automate portions of this for certain processes.

There is much that goes into running a production grade certificate authority, this guide is far from comprehensive. Hopefully it's helpful in some way.

And sure, you can do much of this without CFSSL, but I appreciate the flexibility provided by their use of json configuration files.

See also: Jason Riddle's Guide

Creating the Certificate Authority

Create ca-csr.json:

```
$> cfssl gencert -initca ca-csr.json | cfssljson -bare ca
2017/06/09 18:40:36 [INFO] generating a new CA key and certificate from CSR
2017/06/09 18:40:36 [INFO] generate received request
2017/06/09 18:40:36 [INFO] received CSR
2017/06/09 18:40:36 [INFO] generating key: rsa-2048
2017/06/09 18:40:37 [INFO] encoded CSR
2017/06/09 18:40:37 [INFO] signed certificate with serial number 584075898042636948700563739914716129807961430233
```

Create cs-config.json

Keep in mind that none of the `.pem` files generated in this process are encrypted - and should be protected as such

Creating the Server Keypair

Create prototype-server-csr.json:

```
cfssl gencert \
  -ca=ca.pem \
  -ca-key=ca-key.pem \
  -config=ca-config.json \
  -profile=server \
  prototype-server-csr.json | cfssljson -bare prototype-server
```

The output is something like:

```
2017/06/09 18:48:31 [INFO] generate received request
2017/06/09 18:48:31 [INFO] received CSR
2017/06/09 18:48:31 [INFO] generating key: rsa-2048
2017/06/09 18:48:31 [INFO] encoded CSR
2017/06/09 18:48:31 [INFO] signed certificate with serial number 615205400119918557792655758449511430876160885834
```

Creating the Client Keypair

Create prototype-client-csr.json:

```
cfssl gencert \
    -ca=ca.pem \
    -ca-key=ca-key.pem  \
    -config=ca-config.json \
    -profile=client \
    prototype-client-csr.json | cfssljson -bare prototype-client
```

The output is something like:

```
2017/06/09 18:52:29 [INFO] generate received request
2017/06/09 18:52:29 [INFO] received CSR
2017/06/09 18:52:29 [INFO] generating key: rsa-2048
2017/06/09 18:52:30 [INFO] encoded CSR
2017/06/09 18:52:30 [INFO] signed certificate with serial number 474147588856460409123627575091895953881007131809
```

Creating PCKS12 Keystores

Creating the p12 from the key and cert:

```
openssl pkcs12 -export -out prototype-client.p12 \
  -inkey prototype-client-key.pem -in prototype-client.pem -certfile ca.pem
openssl pkcs12 -export -out prototype-server.p12 \
  -inkey prototype-server-key.pem -in prototype-server.pem -certfile ca.pem
```

Creating JKS Keystore and Truststore

Java applications commonly need these:

Creating the jks from the p12:

```
keytool -importkeystore -srckeystore prototype-client.p12 \
  -storetype pkcs12 -destkeystore prototype-client.jks \
  -deststoretype jks

keytool -importkeystore -srckeystore prototype-server.p12 \
  -storetype pkcs12 -destkeystore prototype-server.jks \
  -deststoretype jks
```

Creating a truststore

```
keytool -import -file ca.pem -alias 'PrototypeCA' -keystore truststore.jks
```

Creating a PCKS8 Keystore

This will generate an encrypted pkcs8 keystore, use `-nocrypt` if you don't want that.

```
openssl pkcs8 -topk8 -in prototype-server-key.pem -out prototype-server-key.p8
```

Encrypting Private Keys

```
openssl rsa -aes256 -in prototype-server-key.pem -out prototype-server-key-enc.pem
```

If you need to remove encryption for some reason, e.g: to change the encryption key or re-encrypt with a different cipher, use:

```
openssl rsa -in prototype-server-key-enc.pem -out prototype-server-key.pem
```

Creating SSH Keypairs from PKCS12 Keystores

SSH private keys are simply encrypted forms of the keys generated by CFSSL, but you can generate the public version of this key using:

```
ssh-keygen -f prototype-client-key.pem -y > prototype-client.pub
```