A Programmer’s Guide to Secure Connections

Liz Rice
Cyber and online shopping security tips

How a green padlock can help

Cybercrime is a constant threat, but you can help make your site safer for customers by providing a secure online connection.
Provisioning a CA and Generating TLS Certificates

In this lab you will provision a PKI Infrastructure using CloudFlare's PKI toolkit, cfssl, then use it to bootstrap a Certificate Authority, and generate TLS certificates for the following components: etcd, kube-apiserver, kube-controller-manager, kube-scheduler, kubelet, and kube-proxy.

Certificate Authority

In this section you will provision a Certificate Authority that can be used to generate additional TLS certificates.

Generate the CA configuration file, certificate, and private key:

```
{
  
  cat > ca-config.json <<EOF
```
A guide to TLS connections

- As a Go programmer, how do I secure my connections?
- What do these error messages mean?
- What the hell are all these .crt, .key, .csr and .pem files?
Hello, I'm Liz

Hi, I'm your bank

Great! Here's $500

Establishing identity is critical
Hello, I'm Liz

Hi! I'm your bank

Great! Here’s $500

Encrypted traffic prevents interception
HTTPS

From Wikipedia, the free encyclopedia

HTTP Secure (HTTPS) is an extension of the Hypertext Transfer Protocol (HTTP) for secure communication over a computer network, and is widely used on the Internet.[1][2] In HTTPS, the communication protocol is encrypted using Transport Layer Security (TLS), or formerly, its predecessor, Secure Sockets Layer (SSL). The protocol is therefore also often referred to as HTTP over TLS,[3] or HTTP over SSL.
HTTP(S) runs over TCP

- Create TCP connection
- TLS - encrypt TCP connection
  - Skip if regular HTTP
- Send HTTP packets on connection
TCP connection

59.97.3.25:60401  27.193.43.2:8080

“Connection refused” = wrong port
(nearly always)
Establishing TCP

Verify certificate, then call
VerifyPeerCertificate

GetClientCertificate
(or Certificate)
Generate Pre-Master Secret
Generate session key
from Pre-Master Secret
Symmetric encryption with session key

<Pre-Master Secret>
(encrypted with server key)
Change cipher <session key>
FINISHED
FINISHED
Symmetric encryption with session key

<Server certificate>
HELLO
HELLO DONE

GetCertificate
(or Certificate)
Verify certificate, then call
VerifyPeerCertificate
Generate session key
from Pre-Master Secret

<Client certificate>
Verify certificate, then call
VerifyPeerCertificate
GetClientCertificate
(or Certificate)
Generate Pre-Master Secret
Generate session key
from Pre-Master Secret
Symmetric encryption with session key
Keys & certificates
Public / private key encryption

- **Public key** can be freely distributed and is used to encrypt
- **Private key** must be kept private and is used to decrypt

```
"hello"  <encrypted>  "hello"
```
Public / private key signatures

- **Private key** must be kept private and is used to sign message
- **Public key** is used to verify signature

“hello” + signature = signature

“hello” + signature

“hello” + signature
Sharing a public key

Hi, I’m Liz. Here’s my public key.

Why should I believe you?

Need a trusted authority in common “Certificate Authority”
X.509 certificate

- Subject name
- Subject’s public key
- Issuer (CA) name
- Validity

Certificate **signed** by issuer (CA)

This is to certify that

liz-server

has public key

abcdef

CA
Subject Name

- Your certs should use Subject alternative names (SAN)
- Common Name deprecated in 2000
  - Chrome browser stopped supporting CN in April 2017
  - SAN supports multiple DNS names in one certificate
Creating keys & certificates
Trusted Certificate Authorities

- Like Let’s Encrypt
- Known in system certificate pools
- Create a Certificate Signing Request
  - `openssl req -key private-key -new -out csr`
- For public-facing domains
- Not for internal components in a distributed system
CLI tools

- **openssl**
  - See contents of certificate: `openssl x509 -text`
  - Doesn’t easily support SANs (Subject Alternative Names)

- **cfssl**
  - Comprehensive toolkit

- **mkcert**
  - Local development
  - Installs CA into your system & browsers

- **minica**
  - Easy generation of key & certs
There are several commonly used filename extensions for X.509 certificates. Unfortunately, some of these extensions are also used for other data such as private keys.

- `.pem` – *(Privacy-enhanced Electronic Mail)* Base64 encoded DER certificate, enclosed between "-----BEGIN CERTIFICATE-----" and "-----END CERTIFICATE-----"
- `.cer`, `.crt`, `.der` – usually in binary DER form, but Base64-encoded certificates are common too (see `.pem` above)
Mutually-authenticated TLS (mTLS)
Takeaways
To establish your identity

You will need:

- A private key
- A certificate for your identity

The other end needs to trust the Certificate Authority that signed your certificate. This may require appending the CA’s certificate.
File extensions

Inconsistently used

- Information type: .crt for certificate, .key for private key...
- Or file format: .pem

PEM files are base64-encoded and tell you what they contain

- openssl can tell you about the contents
Common error messages

- Connection refused
  - Check you’re connecting to the right port

- Certificate signed by unknown authority
  - Received a certificate, but it’s not trusted
  - Examine CA in certificate to see if it should be known to receiver

- Remote error
  - It’s the other end that’s complaining
github.com/lizrice/secure-connections

@lizrice