# **用Go语言写HTTPS程序**

这篇文字基本是Tony Bai的这篇博客[tony](https://github.com/k8sp/tls/blob/master/golang.md" \l "tony)的翻版；只是使 内容和前两篇介绍TLS原理的OpenSSL操作的文字衔接。

## **单向验证身份**

一般的HTTPS服务都是只需要客户端验证服务器的身份就好了。比如我们想访问 银行的网站，我们得确认那个网站真是我们要访问的银行的网站，而不是一个界 面类似的用来诱骗我们输入银行账号和密码的钓鱼网站。而银行网站并不需要通 过TLS验证我们的身份，因为我们会通过在网页里输入账号的密码向服务器展示 我们的用户身份。

### **HTTPS服务器程序**

[上文](https://github.com/k8sp/tls/blob/master/openssl.md" \l "https-server)中我们贴了一个用Go语言写的HTTPS server程序[./server.go](./server.go)：

package main

import (

"io"

"log"

"net/http"

)

func main() {

http.HandleFunc("/", func(w http.ResponseWriter, req \*http.Request) {

io.WriteString(w, "hello, world!\n")

})

if e := http.ListenAndServeTLS(":443", "server.crt", "server.key", nil); e != nil {

log.Fatal("ListenAndServe: ", e)

}

}

我们可以用[create\_tls\_asserts.bash](./create\_tls\_asserts.bash)创建私 钥 server.key和身份证server.crt，

openssl genrsa -out server.key 2048

openssl req -nodes -new -key server.key -subj "/CN=localhost" -out server.csr

openssl x509 -req -sha256 -days 365 -in server.csr -signkey server.key -out server.crt

并且启动服务程序：

sudo go run server.go &

### **不验证服务器身份的客户端程序**

[上文](https://github.com/k8sp/tls/blob/master/openssl.md" \l "https-server)中我们展示了可以给curl一个-k参数， 让它不验证服务器身份即访问。我们自己也写一个类似curl -k的client程序 [unsecure-client.go](./unsecure-client.go)来坚持访问一个不一定安全的 HTTPS server：

package main

import (

"crypto/tls"

"io"

"log"

"net/http"

"os"

)

func main() {

c := &http.Client{

Transport: &http.Transport{

TLSClientConfig: &tls.Config{InsecureSkipVerify: true},

}}

if resp, e := c.Get("https://localhost"); e != nil {

log.Fatal("http.Client.Get: ", e)

} else {

defer resp.Body.Close()

io.Copy(os.Stdout, resp.Body)

}

}

用以下命令编译和启动这个客户端程序：

$ go run unsecure-client.go

hello, world!

### **用自签署的身份证验证服务器身份**

[上文](https://github.com/k8sp/tls/blob/master/openssl.md" \l "https-server)中我们还展示了可以把服务器的身份证 server.crt通过--cacert参数传给curl，让curl用服务器自己的身份证验证 它自己。类似的，我们也可以写一个类似curl --cacert server.crt的Go程序 [secure-client.go](./secure-client.go)来访问HTTPS server。这个程序和 上一个的区别仅仅在于 TLSClientConfig 的配置方式：

c := &http.Client{

Transport: &http.Transport{

TLSClientConfig: &tls.Config{RootCAs: loadCA("server.crt")},

}}

其中 loadCA 的实现很简单：

func loadCA(caFile string) \*x509.CertPool {

pool := x509.NewCertPool()

if ca, e := ioutil.ReadFile(caFile); e != nil {

log.Fatal("ReadFile: ", e)

} else {

pool.AppendCertsFromPEM(ca)

}

return pool

}

## **双方认证对方身份**

有的时候，客户端通过输入账号和密码向服务器端展示自己的身份的方式太过繁 琐。尤其是在如果客户端并不是一个人，而只是一个程序的时候。这时，我们希 望双方都利用一个身份证（certificate）通过TLS协议向对方展示自己的身份。 比如这个关于Kubernetes的[例子](https://github.com/k8sp/tls/blob/master/tls.md" \l "%E5%8F%8C%E6%96%B9%E8%AE%A4%E8%AF%81)。

### **创建CA并签署server以及client的身份证**

我们可以按照[上文](https://github.com/k8sp/tls/blob/master/golang.md" \l "%E7%94%A8%E8%87%AA%E7%AD%BE%E7%BD%B2%E7%9A%84%E8%BA%AB%E4%BB%BD%E8%AF%81%E9%AA%8C%E8%AF%81%E6%9C%8D%E5%8A%A1%E5%99%A8%E8%BA%AB%E4%BB%BD)中例子展示的：让通 信双方互相交换身份证，这样既可互相验证。但是如果一个分布式系统里有多方， 任意两房都要交换身份证太麻烦了。我们通常创建一个 [自签署的根身份证](https://github.com/k8sp/tls/blob/master/tls.md" \l "%E6%A0%B9%E8%BA%AB%E4%BB%BD%E8%AF%81%E5%92%8C%E8%87%AA%E7%AD%BE%E5%90%8D)，然后用它来签署分布式系 统中各方的身份证。这样每一方都只要有这个根身份证即可验证所有其他通信方。 [这里](https://github.com/k8sp/tls/blob/master/openssl.md" \l "%E7%AD%BE%E7%BD%B2%E8%BA%AB%E4%BB%BD%E8%AF%81)解释了用OpenSSL生成根身份证和签署其他身 份证的过程。针对我们的例子，具体过程如下：

1.创建我们自己CA的私钥：

openssl genrsa -out ca.key 2048

2.创建我们自己CA的CSR，并且用自己的私钥自签署之，得到CA的身份证：

openssl req -x509 -new -nodes -key ca.key -days 10000 -out ca.crt -subj "/CN=we-as-ca"

3.创建server的私钥，CSR，并且用CA的私钥自签署server的身份证：

openssl genrsa -out server.key 2048

openssl req -new -key server.key -out server.csr -subj "/CN=localhost"

openssl x509 -req -in server.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out server.crt -days 365

4.创建client的私钥，CSR，以及用ca.key签署client的身份证：

openssl genrsa -out client.key 2048

openssl req -new -key client.key -out client.csr -subj "/CN=localhost"

openssl x509 -req -in client.csr -CA ca.crt -CAkey ca.key -CAcreateserial -out client.crt -days 365

### **Server**

相对于上面的例子，server的源码 [./bidirectional/server.go](./bidirectional/server.go)稍作了一些修改： 增加了一个 http.Server 变量s，并且调用s.ListenAndServeTLS，而不 是像之前那样直接调用http.ListenAndServeTLS了：

func main() {

s := &http.Server{

Addr: ":443",

Handler: http.HandlerFunc(func(w http.ResponseWriter, r \*http.Request) {

fmt.Fprintf(w, "Hello World!\n")

}),

TLSConfig: &tls.Config{

ClientCAs: loadCA("ca.crt"),

ClientAuth: tls.RequireAndVerifyClientCert,

},

}

e := s.ListenAndServeTLS("server.crt", "server.key")

if e != nil {

log.Fatal("ListenAndServeTLS: ", e)

}

}

### **Client**

客户端程序[./bidirectional/client.go](./bidirectional/client.go)相对 于上面提到的unsecure-client.go和secure-client.go的变化主要在于

1. 调用tls.LoadX509KeyPair读取client.key和client.crt，并返回一个 tls.Certificate变量，
2. 把这个变量传递给http.Client变量，然后调用其Get函数。

func main() {

pair, e := tls.LoadX509KeyPair("client.crt", "client.key")

if e != nil {

log.Fatal("LoadX509KeyPair:", e)

}

client := &http.Client{

Transport: &http.Transport{

TLSClientConfig: &tls.Config{

RootCAs: loadCA("ca.crt"),

Certificates: []tls.Certificate{pair},

},

}}

resp, e := client.Get("https://localhost")

if e != nil {

log.Fatal("http.Client.Get: ", e)

}

defer resp.Body.Close()

io.Copy(os.Stdout, resp.Body)

}

### **运行和测试**

cd bidirectional

./create\_tls\_asserts.bash # 创建各种TLS资源

sudo go run ./server.go & # 启动服务器

go run ./client.go # 尝试连接服务器

应当看到屏幕上打印出来 Hello World!。

在这篇博客[tony](https://github.com/k8sp/tls/blob/master/golang.md" \l "tony)中提到，需要创建一个client.ext文件， 使得client的身份证里包含ExtKeyUsage字段。但是我并没有这么做，得到的 程序也可以运行。

文章地址：https://github.com/k8sp/tls/blob/master/golang.md