

# 用 HTTPS 加密 API 请求

## 本节核心内容

- 介绍 HTTPS 基本原理
- 介绍如何用 HTTPS 加密 API 请求，并测试加密效果

本小节源码下载路径：[demo10](#)

([https://github.com/lexkong/apiserver\\_demos/tree/master](https://github.com/lexkong/apiserver_demos/tree/master))

可先下载源码到本地，结合源码理解后续内容，边学边练。

本小节的代码是基于 [demo09](#)

([https://github.com/lexkong/apiserver\\_demos/tree/master](https://github.com/lexkong/apiserver_demos/tree/master)) 来开发的。

## 什么是 HTTPS

在前面的小节中，客户端与 API 服务器请求响应的是 HTTP，不过 HTTP 是明文的，在网络上进行传输可能会被窃听、篡改甚至冒充，因此对于一个企业级的 API 服务器来说，通常需要采用更安全的 HTTPS 协议。

HTTPS（全称 Hyper Text Transfer Protocol over Secure Socket Layer），是以安全为目标的 HTTP 通道，简单讲是 HTTP 的安全版。即 HTTP 下加入 SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。

SSL：安全套接层，是 Netscape 公司设计的主要用于 Web 的安全传输协议。这种协议在 Web 上获得了广泛的应用。通过证书认证来确保客户端和网站服务器之间的通信数据是加密安全的。

TLS 是 SSL 的升级版，使用层面，读者可以理解为二者无区别。

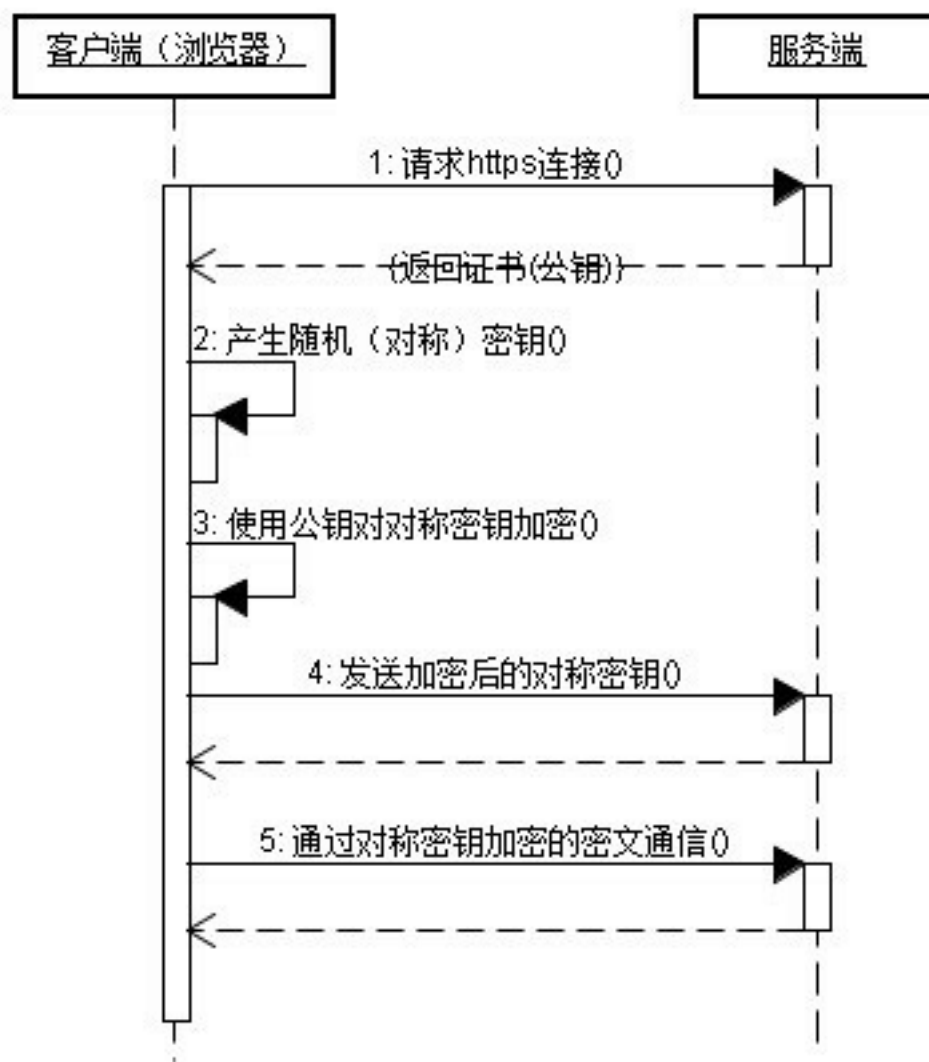
## HTTPS 的实现原理

HTTPS 在传输数据之前需要客户端（浏览器）与服务端（网站）之间进行一次握手，在握手过程中将确立双方加密传输数据的密码信息。

有两种基本的加解密算法类型：

1. 对称加密：密钥只有一个，加密解密为同一个密码，且加解密速度快，典型的对称加密算法有 DES、AES 等。
2. 非对称加密：密钥成对出现（且根据公钥无法推知私钥，根据私钥也无法推知公钥），加密解密使用不同密钥（公钥加密需要私钥解密，私钥加密需要公钥解密），相对对称加密速度较慢，典型的非对称加密算法有 RSA、DSA 等。

下图是 HTTPS 的通信过程：



流程大概是这样的：

1. SSL 客户端通过 TCP 和服务器建立连接之后（443 端口），并且在一般的 TCP 连接协商（握手）过程中请求证书。

即客户端发出一个消息给服务器，这个消息里面包含了自己可实现的算法列表和其它一些需要的消息，SSL 的服务器端会回应一个数据包，这里面确定了这次通信所需要的算法，然后服务器向客户端返回证书。（证书里面包含了服务器信息——域名、申请证书的公司、公共密钥。）

2. 客户端在收到服务器返回的证书后，判断签发这个证书的公共签发机构，并使用这个机构的公共密钥确认签名是否有效，客户端还会确保证书中列出的域名就是它正在连接的域名。

3. 如果确认证书有效，那么生成对称密钥并使用服务器的公共密钥进行加密。然后把它发送给服务器，服务器使用它的私钥解密出会话密钥，然后把内容通过会话密钥对称加密，这样两台计算机可以开始进行对称加密进行通信。

HTTPS 通信的优点：

1. 客户端产生的密钥只有客户端和服务端能得到；
2. 加密的数据只有客户端和服务端才能得到明文；
3. 客户端到服务端的通信是安全的。

## Go 语言 HTTPS 支持

Go 语言的 `net/http` 包中的 `ListenAndServeTLS()` 函数提供了对 HTTPS 的支持。`ListenAndServeTLS()` 函数的原型为：

```
func ListenAndServeTLS(addr string, certFile  
string, keyFile string, handler Handler) error
```

可以看出，这个函数原型其实和 HTTP 方式的差别就在于，需要提供数字证书 `certFile` 和私钥文件 `keyFile`。在测试环境，我们没有必要花钱去购买什么证书，利用 OpenSSL 工具，我们可以自己生成私钥文件和自签发的数字证书。

## API Server 添加 HTTPS 支持

在 `apiserver` 中添加 HTTPS 功能，步骤很简单，大概分为以下三步。

1. 生成私钥文件 (`server.key`) 和自签发的数字证书 (`server.crt`)：

```
$ openssl req -new -nodes -x509 -out
conf/server.crt -keyout conf/server.key -days
3650 -subj "/C=DE/ST=NRW/L=Earth/O=Random
Company/OU=IT/CN=127.0.0.1/emailAddress=xxxxx@qq.
com"
```

```
[api@centos apiserver]$ ls conf
config.yaml  server.crt  server.key
[api@centos apiserver]$
```

2. 在配置文件中配置私钥文件、数字证书文件的路径和 HTTPS 端口，供 ListenAndServeTLS() 函数调用：

```
runmode: debug          # 开发模式, debug, release, test
addr: :8080              # HTTP绑定端口
name: apiserver          # API Server的名字
url: http://127.0.0.1:8080 # pingServer函数请求的API服务器的ip:port
max_ping_count: 10       # pingServer函数try的次数
jwt_secret: Rtg8BPKNEf2mB4mgvKONGPZZQSaJWNLijxR42qRgq0iBb5
tls:
  addr: :8081
  cert: conf/server.crt
  key: conf/server.key
log:
```

3. 在 main 函数中增加 ListenAndServeTLS() 调用，启动 HTTPS 端口：

```
// Ping the server to make sure the router is working.
go func() {
    if err := pingServer(); err != nil {
        log.Fatal("The router has no response, or it might took too long to start up.", err)
    }
    log.Info("The router has been deployed successfully.")
}()

// Start to listening the incoming requests.
cert := viper.GetString("tls.cert")
key := viper.GetString("tls.key")
if cert != "" && key != "" {
    go func() {
        log.Infof("Start to listening the incoming requests on https address: %s", viper.GetString("tls.addr"))
        log.Info(http.ListenAndServeTLS(viper.GetString("tls.addr"), cert, key, g).Error())
    }()
}

log.Infof("Start to listening the incoming requests on http address: %s", viper.GetString("addr"))
log.Info(http.ListenAndServe(viper.GetString("addr"), g).Error())
}
```

main 函数的逻辑是：如果提供了 TLS 证书和私钥则启动 HTTPS 端口。

创建证书和密钥需要 Linux 安装 openssl 工具，大部分 Linux 发行版已经默认安装，如果没有安装请安装。

## 编译并测试

1. 下载 apiserver\_demos 源码包（如前面已经下载过，请忽略此步骤）

```
$ git clone  
https://github.com/lexkong/apiserver\_demos
```

2. 将 apiserver\_demos/demo10 复制为  
\$GOPATH/src/apiserver

```
$ cp -a apiserver_demos/demo10/  
$GOPATH/src/apiserver
```

3. 在 apiserver 目录下编译源码

```
$ cd $GOPATH/src/apiserver  
$ gofmt -w .  
$ go tool vet .  
$ go build -v .
```

请求时不携带 CA 证书和私钥

```
$ curl -XGET -H "Authorization: Bearer  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOiE1  
MjgwMTY5MjIsImklIjowLCJuYmYiOiE1MjgwMTY5MjIsInVzZ  
XJuYW1lIjoiYWRtaW4ifQ.LjxrK9DuAwAzUD8-  
9v43NzWBN7HXsSLfebW92DKd1JQ" -H "Content-Type:  
application/json" https://127.0.0.1:8081/v1/user
```

```
curl: (60) Peer certificate cannot be  
authenticated with known CA certificates  
More details here:  
http://curl.haxx.se/docs/sslcerts.html
```

curl performs SSL certificate verification by default, using a "bundle" of Certificate Authority (CA) public keys (CA certs). If the default bundle file isn't adequate, you can specify an alternate file using the `--cacert` option.

If this HTTPS server uses a certificate signed by a CA represented in the bundle, the certificate verification probably failed due to a problem with the certificate (it might be expired, or the name might not match the domain name in the URL).

If you'd like to turn off curl's verification of the certificate, use the `-k` (or `--insecure`) option.

可以看到请求认证失败。

请求协议需要是 HTTPS，请求的端口需要是 HTTPS 的 8081 端口。

**请求时携带 CA 证书和私钥**



```
$ curl -XGET -H "Authorization: Bearer  
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpYXQiOi0jE1  
MjgwMTY5MjIsImIjoiYWRtaW4ifQ.LjxrK9DuAwAzUD8-  
9v43NzWBN7HXsSLfebW92DKd1JQ" -H "Content-Type:  
application/json" https://127.0.0.1:8081/v1/user  
--cacert conf/server.crt --cert conf/server.crt -  
-key conf/server.key
```

```
{  
  "code": 0,  
  "message": "OK",  
  "data": {  
    "totalCount": 1,  
    "userList": [  
      {  
        "id": 0,  
        "username": "admin",  
        "sayHello": "Hello qybyTdSmg",  
        "password":  
"$2a$10$veGcArz47VGj7l9xN7g2iuT9TF21jLI1YGXarGzvA  
RNdnt4inC9PG",  
        "createdAt": "2018-05-28 00:25:33",  
        "updatedAt": "2018-05-28 00:25:33"  
      }  
    ]  
  }  
}
```

成功请求。

## 小结

本小节是 API 开发的进阶内容，讲解了如何给 HTTP 请求进行数据加密。在企业级的 API 服务器中，通常会进行 HTTPS 加密。