HTTP & TLS

2022年3月4日　　10:55

==========

==========

==========

Etag是 Entity tag的缩写，可以理解为“被请求变量的实体值”，Etag是服务端的一个资源的标识，在 HTTP 响应头中将其传送到客户端。所谓的服务端资源可以是一个Web页面，也可以是JSON或XML等。服务器单独负责判断记号是什么及其含义，并在HTTP响应头中将其传送到客户端。比如，浏览器第一次请求一个资源的时候，服务端给予返回，并且返回了ETag: "50b1c1d4f775c61:df3" 这样的字样给浏览器，当浏览器再次请求这个资源的时候，浏览器会将If-None-Match: W/"50b1c1d4f775c61:df3" 传输给服务端，服务端拿到该ETAG，对比资源是否发生变化，如果资源未发生改变，则返回304HTTP状态码，不返回具体的资源。

==========

==========

==========

==========

单向SSL 与 双向SSL

SSL 用于 在客户端 和服务器间 实现安全通信 以确保 数据安全性 和 完整性 的标准技术。
1995年，SSL v2 是 SSL 的第一个公共版本
1996，SSL v3
1999， TLS v1.0
2006, TLS v1.1
2008, TLS v1.2

SSL 可以采用 单向 或 双向 实现。 one-way SSL, two-way SSL(也称为 Mutual SSL)

单向SSL，只有客户端验证服务器 以确保 它从语气的服务器接收数据。 为了实现单向SSL，服务器与客户端共享其公共证书。
单向SSL的情况下，客户端和服务器 之间 建立连接 和 传输数据 所涉及的步骤 的描述：

1. 客户端通过 HTTPS 协议 向服务器 请求一些受保护的数据，这将启动 SSL/TLS 握手过程。
2. 服务器将其公共证书 连同 服务器 问候消息 一起返回给 客户端
3. 客户端验证 收到的证书。客户端 通过 CA 签名证书 的证书颁发机构(CA) 验证证书。
4. SSL/TLS 客户端发送 随机字节串，使客户端 和服务器 都能 计算出 用于 加密后续消息数据的 密钥。随机字符串本身 是用 服务器的 公钥加密的。
5. 在同意此 密钥后，客户端和服务器 通过 使用 此密钥 加解密 数据 进一步 通信以进行实际数据传输。

双向SSL， 客户端和 服务器 都相互验证，以确保 参与通信的双方 都是可信的。双方共享它们的公共证书，然后在此基础上 执行 验证。
双向SSL 的情况下，客户端和 服务器之间建立连接和传输数据 所涉及的描述

1. 客户端通过HTTPS 协议 请求一个 受保护的 资源， SSL/TLS 握手过程开始。
2. 服务器将其公共证书 与服务器问候 一起返回客户端
3. 客户端验证 收到的证书。客户端 通过 CA签名证书 的 证书颁发机构(CA) 验证证书
4. 如果服务器证书验证成功，客户端将向 服务器提供 公共证书。
5. 服务器 验证 收到的证书。服务器通过 CA签名证书 的 证书颁发机构(CA) 验证证书
6. 握手完成后，客户端 和服务器 相互通信 并传输数据，并在 握手过程中 使用 两者共享的 密钥加密。

==========

http://t.zoukankan.com/beiyan-p-6248187.html

单向https配置

生成https证书命令：

```
sudo keytool -genkey -keyalg RSA -dname
"cn=localhost,ou=none,o=none,l=shanghai,st=shanghai,c=cn" -alias server -
keypass 123456 -keystore server.keystore -storepass 123456 -validity 3650
```

生成CSR (Certificate Signing Request) 文件

本文只是生成 自签名 的https 证书，如果需要申请CA证书，就需要生成 CSR文件，并将此文件 提交给 相应 CA机构申请 CA证书

```
sudo keytool -certReq -alias server -keystore server.keystore -file ca.csr -
storepass 123456
```

生成CER文件

因为我们生成的 证书 是 keytool 生成的，没有经过 操作系统 可信任的 CA机构颁发，所以当用 浏览器访问时， 会出现 不信任证书 警告，我们 手工将 cer文件 (服务端公钥) 导入浏览器的 证书列表，让其信任

```
sudo keytool -export -alias server -keystore server.keystore -file ca.cer -
storepass 123456
```

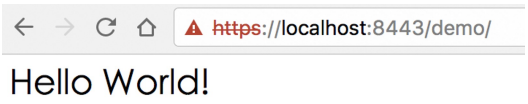
在Tomcat 中配置 https证书

在server.xml 中添加如下配置，即可访问 https 的站点了

```
<Connector SSLEnabled="true" clientAuth="false"
keystoreFile="/Users/beiyan/Documents/test/server.keystore"
keystorePass="123456" maxThreads="150" port="8443"
protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="https"
secure="true" sslProtocol="TLS" />
```

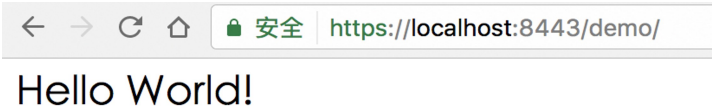
keystoreFile 为证书的地址，keystorePass 是证书的密码。

浏览器访问界面如下：



导出CER文件，让浏览器信任此证书

双击 ca.cer 即可导入，再设置为 始终信任，再次访问：



应用程序访问

如果使用 HttpClient 等工具访问该https链接时，需要 将 ca.cer 导入 jre中：

```
keytool -import -alias tomcatsso -file "ca.cer" -keystore
"/Library/Java/JavaVirtualMachines/jdk1.8.0_
111.jdk/Contents/Home/jre/lib/security/cacerts" -storepass 123456
```

一个是 jre 目录，123456 是 jre默认的密码。

双向https配置

生成Server端证书

```
sudo keytool -genkey -keyalg RSA -dname
"cn=localhost,ou=none,o=none,l=shanghai,st=shanghai,c=cn" -alias server -
keypass 123456 -keystore server.keystore -storepass 123456 -validity 3650
```

生成客户端证书

这里的客户端为浏览器，浏览器支持的证书格式为 PKCS12，这里生成 PKCS12格式的证书

```
sudo keytool -genkey -v -alias client -keyalg RSA -storetype PKCS12 -dname
"cn=localhost,ou=none,o=none,l=shanghai,st=shanghai,c=cn"  -keypass 123456 -
storepass 123456 -keystore client.p12 -validity 3650
```

让服务器端信任客户端的证书

由于是双向认证，服务器端必须验证 客户端的身份，所以需要将客户端的 公钥 导入到 服务端的 信任列表，但是这里生成的 PKCS12 文件不能直接导入，所以 先导出成 CER文件，再将 CER文件导入 服务端的证书库。

1. 将客户端证书导出为 一个单独的 CER文件

```
sudo keytool -export -alias client -keystore client.p12 -storetype
PKCS12 -storepass 123456 -rfc -file client.cer
```

2. 将CER文件导入到 服务端的 证书库。

```
sudo keytool -import -v -file client.cer -keystore server.keystore
```

3. 查看 server.keystore 里面的证书列表

```
sudo keytool -list -keystore server.keystore
```

让客户端信任服务端证书

客户端也需要验证服务端的证书是否可靠，所以也需要将 服务端证书的 公钥导入客户端的信任列表。通常的做法是 将服务器证书 导出为 一个单独的 CER文件，然后 双击安装到 浏览器的 证书列表 即可。

```
sudo keytool -keystore server.keystore -export -alias server -file
server.cer -validity 36500
```

所有生成的证书：

client.cer：客户端证书的公钥
client.p12：客户端证书的私钥
server.cer：服务端证书的公钥
server.keystore：服务端证书库，既包含服务端私钥，又包含客户端公钥

修改Tomcat配置

在server.xml 中添加 如下配置

```
<Connector SSLEnabled="true" clientAuth="true"
            keystoreFile="/Users/beiyan/Documents/test/keytool/server.keystore"
            truststoreFile="/Users/beiyan/Documents/test/keytool/server.keystore"
            truststorePass="123456" keystorePass="123456" maxThreads="150"
port="8443"
```

```
protocol="org.apache.coyote.http11.Http11NioProtocol" scheme="https"
secure="true" sslProtocol="TLS" />
```

clientAuth="true"表示双向认证

客户端安装私钥
双击 client.p12 即可安装

客户端安装服务端的公钥
双击 server.cer 文件。

成功访问
弹出客户端证书选择界面，选择客户端证书，即可正常访问。

公钥(证书) 和 私钥 是成对存在的。 通信双方 各自 持有 自己的私钥 和 对方的 公钥。
自己的 私钥需要密切保护。windows下，单独存在的公钥一般是 后缀为 .cer 的文件。

A用自己的私钥 对数据加密，发给B，B使用 A提供的公钥 解密。

公钥的 2个用途
1. 验证对方身份，防止其他人 假冒发送数据给你
2. 解密

私钥的 2个用途
1. 表明自己的身份，除非第三方 有 你的私钥， 否则 无法冒充你 发送数据 给对方。
2. 加密

jks(java key store)
Java使用的 存储密钥的 容器。后缀一般是 .jks, .keystore, .truststore 等。
用 jdk bin 目录下的 keytool.exe 对其进行 查看，导入，导出，删除，修改密码 等操作。可以对 jks 容器 加密码，输入正确 才能看到 容器中的 密钥。
还有一个 密码的概念 和 上者不同，是 jks 中存储着的 私钥的密码，通常是 绝密的。

pfx：
和 jks 功能相同，但 文件格式不同，pfx 是浏览器用的。
可以用一些工具程序 将 pfx 转化为 jks 供java使用。

---

https://zhuanlan.zhihu.com/p/528683483

基本概念
Hash： 为数据生成唯一签名，比如md5， sha1 之类的算法。
非对称加密： 比如 ECC，RSA 算法。 私钥加密 公钥解密 或 公钥加密 私钥解密
CA： certification authority， SSL/HTTPS 证书的 认证颁发机构。
CSR： certificate signing request， 包含 用户公钥 和 个人信息的 一个数据文件。 用户生成 这个CSR 文件，再把这个 CSR 文件发送给 CA，CA 就会根据 CSR中的 内容 来 签发数字证书。
digital certificate (数字证书)： 就是一张 附带了 数字签名的 信息表。经常使用 X.509 标准 (包含： 算法，信息，公钥，私钥加密后的签名)。类似 JWT

钥匙库
相当于 一个 管理公钥私钥的 数据库。钥匙太多，需要管理，包括：钥匙的 增删改查，不同库 之间的 转换。
最简单的就是 .ssh/authorized_keys 。

PKCS12 (public key cryptography standards)： 定义了一种 存档文件格式，用于实现存储 许多 加密对象在 一个 单独的文件中。通常用它来打包 一个私钥以及有关的 X.509 证书，或者 打包信任链 的全部项目，后缀为 .p12, .pfx

JKS: java key store, java用来管理私钥。

证书链
一般用于 CA 父机构 把 证书颁发权力 交给 子机构。 子机构必须经过 父机构 同意并签名，子证书 必须 包含 父机构的 信息和签名。

https基本流程：
流程分为3部分：
1. CA数字证书认证流程
2. 客户端CA数字证书验证流程
3. 服务端和客户端通信加密解密流程

CA数字证书认证流程
输入：公司生产的CSR (包括：公司信息 和公钥)
输出：CA签发的数字证书(X509格式)

把公司信息 和 CA 机构信息生成 前面，并且用 私钥 加密签名，最后生成 数字证书。
1. 服务器S 生成私钥 s_pri_key 和 公钥 s_csr: s_pub_key + s_info
2. CA 生成 ca_hash: ca_hash = hash(s_csr + ca_info)
3. CA 生成 enc_ca_hash: enc_ca_hash = enc(ca_hash, ca_pri_key)
4. CA 生成数字证书 ca_cert: ca_cert = s_csr + ca_info + ca_pub_key + enc_ca_hash, 下发CA证书 给服务器S

客户端ca_cert 认证流程
用数据证书中的 CA公钥 对加密的 前面 进行解密，并且 做一遍服务器信息 和 CA机构信息

生成 前面，比较 两签名 是否相等
输入：CA签发的数字证书
输出：数字证书验证 是否成功

1. 客户端 从服务器S 获得 ca_cert，并解析，验证CA
2. dec_ca_hash = dec(enc_ca_hash, ca_pub_key);
3. ca_hash = hash(s_csr + ca_info)
4. if dec_ca_hash == ca_hash, 则证明 ca_cert 是从 CA 签署的。

自己签发数字证书
创建认证中心(CA) 测试版
出于测试目的，该CA 代替了 因特网上公认的 CA (如 VeriSign)。 你使用 该 CA 以数字方式 签署 你计划用于测试的 每个证书

1. 创建密钥和CSR 文件
   ```
   openssl req -new -sha256 -newkey rsa:2048 -nodes \
   -keyout ca.com.key -out ca.com.csr \
   -subj "/C=CN/ST=hubei/L=jingzhou/O=CA/OU=RD/CN=ca.com"
   ```

2. 从证书请求 创建 X.509 数字证书。以下命令行创建 通过CA 专用密钥签署的 证书。证书有效期 3650 天。
   ```
   openssl x509 -sha256 -extensions v3_ca -in ca.com.csr -out ca.com.cert -
   req -signkey ca.com.key -days 3650
   ```

3. 可选：创建包含证书 和 专用密钥的 PKCS12 编码文件。以下命令将 P12 文件上的 密码设置为default
   ```
   openssl pkcs12 -passout pass:default -export -nokeys -cacerts \
   -in ca.com.cert -out ca.com.cert.p12 -inkey ca.com.key
   ```
   你现在拥有一个可安装到 受测试Web 服务器的 CA证书 (ca.com.cert) 以及一个可用于签署用户证书的 专用密钥文件 (ca.com.key)

4. 可选：添加域名限制
   ```
   openssl version -d #获取openssl配置目录 一般是 /usr/lib/ssl， mac:
   /private/etc/ssl
   cd /usr/lib/ssl
   vi openssl.cnf
   [req]
   req_extensions = v3_req

   [v3_req]
   basicConstraints = CA:FALSE
   keyUsage = nonRepudiation, digitalSignature, keyEncipherment
   subjectAltName = @SubjectAlternativeName

   [SubjectAlternativeName]
   DNS.1 = domain.com
   DNS.2 = *.domain.com
   ```

用CA数字证书签发 客户数字证书

1. 为用户创建 CSR 文件。初始密码为 abc
   ```
   openssl req -new -sha256 -newkey rsa:2048 -nodes \
   -keyout domain.com.key -out domain.com.csr \
   -subj "/C=CN/ST=hubei/L=jingzhou/O=domain/OU=RD/CN=domain.com"
   ```

2. 为新用户创建新的 X.509 证书，使用该用户 的专用密钥 以数字方式 对其 进行签署，并使用 CA 专用密钥 对其 进行认证。以下命令 创建 有效期为 3650 天的证书
   ```
   openssl x509 -sha256 -extensions v3_ca -req -in domain.com.csr -out
   domain.com.cert \
   -signkey domain.com.key -CA ca.com.cert -CAkey ca.com.key -CAcreateserial -
   days 3650
   ```

3. 可选：创建公用密钥的 DER 编码版本。该文件仅包含公用密钥，而不包含专用密钥。由于它不包含专用密钥，因此它可以共享，而且不需要受密码保护
   ```
   openssl x509 -in domain.cert -out domain.cert.der -outform DER
   ```

4. 可选：创建 PKCS#12 编码的文件。以下命令行将 P12 文件上的密码设置为 default。
   ```
   openssl pkcs12 -passout pass:default -export -in domain.cert -out
   domain.cert.p12 -inkey domain.key
   ```
   重复该步骤 以创建 测试所需的数量的 数字证书。 确保密钥文件安全，当不在需要密钥文件的时候 将其删除， 不要删除 CA专用密钥文件。 你需要 CA专用密钥文件 来签署证书。

将签发后的 数字证书 domain.com.cert 和 密钥 domain.com.key 放入 服务器中(nginx)

```
server {
    listen       443 ssl;
    server_name  locahost domain.com www.domain.com;

    # ssl证书地址
    ssl_certificate      certs/domain.com.cert;  # 数字证书路径
    ssl_certificate_key  certs/domain.com.keyy;  # 密钥路径

    root   html;
    index  index.html index.htm;
    location / {

    }
}
```

快速生成自测证书

```
# RSA算法证书
openssl genrsa -out server.key 2048
# 生成数字证书（包含公钥和组织信息）
openssl req -new -x509 -days 3650 -key domain.key -out server.csr -subj
"/C=CN/ST=Guandong/L=Shenzhen/O=MyCompany/OU=RD/CN=localhost/CN=127.0.0.1"

# ECC算法证书
```

```
openssl ecparam -genkey -name prime256v1 -out server.key
openssl req -new -sha384 -key server.key -out server.csr -subj
"/C=CN/ST=Guandong/L=Shenzhen/O=MyCompany/OU=RD/CN=localhost/CN=127.0.0.1"

# Key considerations for algorithm "ECDSA" ≥ secp384r1
# List ECDSA the supported curves (openssl ecparam -list_curves)
openssl ecparam -genkey -name secp384r1 -out server.key
openssl req -new -x509 -sha256 -key server.key -out server.pem -days 3650 -
subj "/C=CN/ST=Guandong/L=Shenzhen/O=MyCompany/OU=RD/CN=localhost/CN=
127.0.0.1"
```

字段	字段含义	示例
/C=	Country 国家	CN
/ST=	State or Province 省	Guangzhou
/L=	Location or City 城市	Shenzhen
/O=	Organization 组织或企业	IBM
/OU=	Organization Unit 部门	RD
/CN=	Common Name域名或IP，可以多个	hello.com

Keytools (Java)

```
keytool -genkey -alias domain.com -sigalg SHA256withRSA -keyalg RSA -keysize
2048 \
-keystore domain.com.jks -dname
"C=CN,ST=hubei,L=jingzhou,O=company_name,OU=my_apartment,CN=domain.com" && \
 keytool -certreq -alias domain.com -file domain.com.csr -keystore
domain.com.jks

#migrate to PKCS12 which is an industry standard
keytool -importkeystore -srckeystore domain.com.jks -destkeystore
domain.com.jks \
-deststoretype pkcs12
```

数字格式 (X509) 格式

PEM格式

```
——BEGIN CERTIFICATE——
base64编码内容
——END CERTIFICATE——
```

包含 数字证书，私钥

DER格式

二进制格式，Java 用的多

包含 数字证书，私钥

P7B/PKCS7

——BEGIN PKCS——

base64编码内容

——END PKCS7——

包含 数字证书 和 数字证书链，不能包含 私钥。

PFX/PKCS12

二进制格式

包含 数字证书，私钥

证书转换命令

```
#PEM to DER, P7B, PFX
openssl x509 -outform der -in certificate.pem -out certificate.der
openssl crl2pkcs7 -nocrl -certfile certificate.cer -out certificate.p7b -
certfile CAcert.cer
openssl pkcs12 -export -out certificate.pfx -inkey privateKey.key -in
certificate.crt -certfile CAcert.crt
# DER,P7B,PFX to PEM
openssl x509 -inform der -in certificate.cer -out certificate.pem
openssl pkcs7 -print_certs -in certificate.p7b -out certificate.cer
openssl pkcs12 -in certificate.pfx -out certificate.cer -nodes
```

ECC证书相关命令

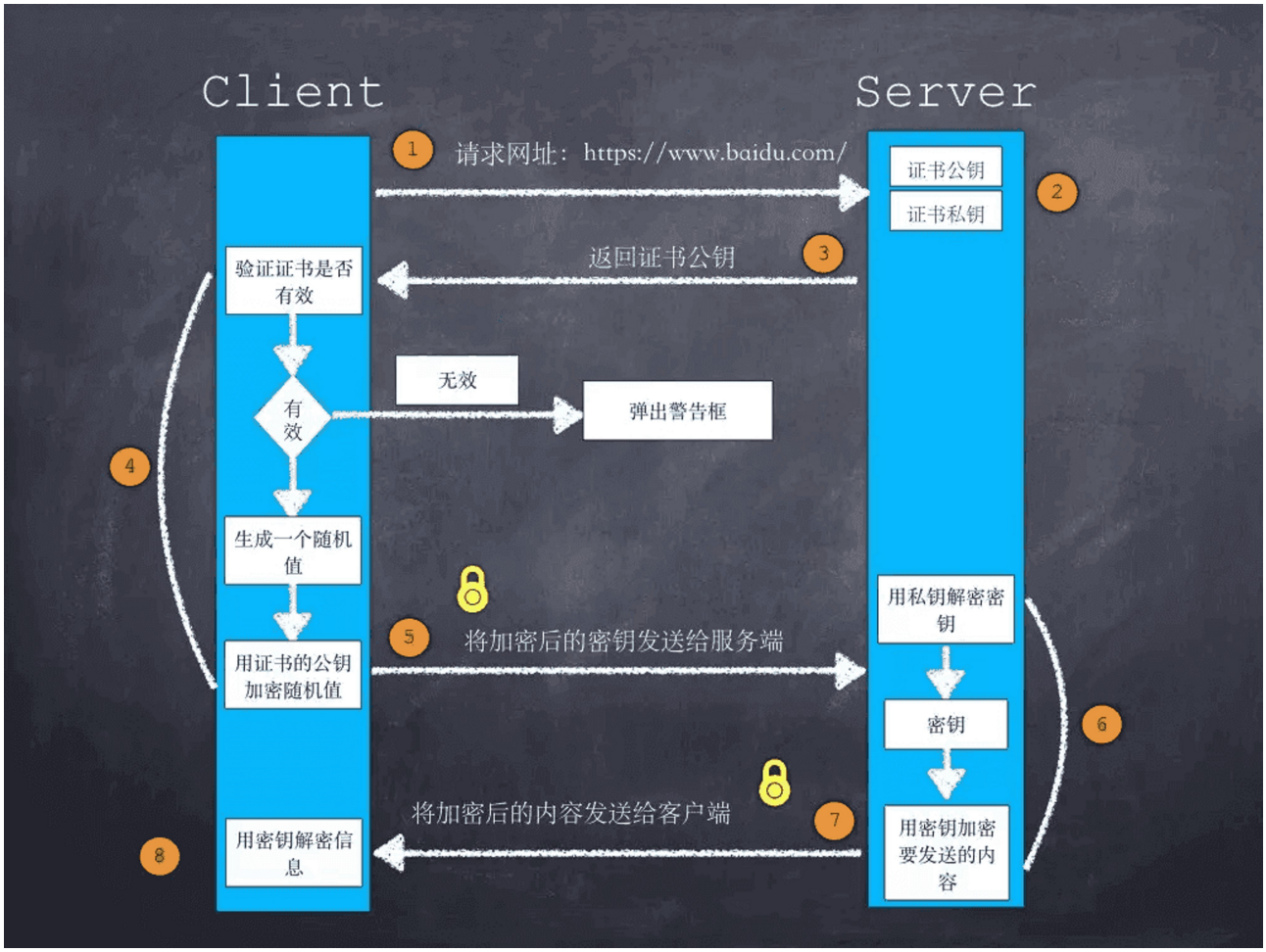
```
# 生成ECC私钥
openssl ecparam -genkey -name prime256v1 -out key.pem
# 生成CSR
openssl req -new -sha256 -key key.pem -out csr.csr -subj
"/C=CN/ST=hubei/L=jingzhou/O=CA/OU=RD/CN=ca.com"
# 生成数字证书
openssl req -x509 -sha256 -days 365 -key key.pem -in csr.csr -out
certificate.pem
# 查看数字证书
test_ecc % openssl req -in csr.csr -text -noout
# 从数字证书提取certificate.pem
openssl x509 -noout -pubkey -in certificate.pem > public_key.pem
# 私钥签名
openssl dgst -sha1 -sign key.pem < randfile > signatrue.bin
# 公钥验证签名
openssl dgst -sha1 -verify public_key.pem -signature signatrue.bin
< randfile
```

==========

==========

https://blog.csdn.net/alan_liuyue/article/details/126384286
关于HTTPS的原理 及 证书，验证，数据加密，解密

HTTPS介绍
HTTPS其实由 两部分组成： HTTP + SSL/TLS， 也就是在 HTTP上又 加了一层处理 加密信息的 模块。
服务端和客户端的信息传输 都通过 TLS加密，所以 传输的数据 是加密后的数据。



HTTPS请求过程：
1. 客户端向服务端发起 HTTPS请求，连接到服务器的 443 端口
2. 服务器 将 非对称加密的 公钥 传给客户端，以证书的形式 传给客户端
3. 客户端收到 公钥，进行验证，即验证上一步的证书，如果有问题，HTTPS请求无法继续；如果没有问题，则上述公钥是合格的。客户端这个时候 随机 生成一个 私钥，称

为 client key, 客户端私钥， 用于 对称加密数据；使用 前面的 公钥 对 client ket 进行 非对称加密
4. 进行第二次HTTPS请求，将 加密后的 client key 传给服务端
5. 服务端 使用 私钥进行解密，获得 client key， 使用 client key 对数据进行 对称加密
6. 将 对称加密 后的 数据 发送给客户端，客户端使用 对称解密，获得 服务端发送的数据。 完成第二次HTTPS请求

SSL 位于 应用层 和 TCP层之间。 应用层数据 不再直接 传递给 传输层，而是 传递给SSL层，SSL 层 对 从 应用层 收到的 数据 进行加密，并增加 自己的 SSL 头。

RSA 性能非常低，因为需要寻找 大素数，大数计算，数据分割 需要消耗很多 CPU周期，所以一般的 HTTPS 连接 只在 第一次 握手的 时候 使用 非对称加密， 通过 握手 交换 对称加密密钥，之后 的通信 走 对称加密。

HTTPS协议 和 HTTP协议的区别
1. https 协议需要 向 CA 申请证书，一般免费证书很少，需要交费
2. http 是超文本传输协议，信息是 明文传输，https是具有安全性的 SSL 加密传输协议
3. http 和 https 使用 完全不同的连接方式，使用的端口也不同。
4. http 的连接 是 无状态的。

HTTPS协议是 由 SSL + HTTP 协议构建的，可进行加密传输，身份认证的 网络协议，比 http 协议安全。

Nginx 实现 HTTPS 网站设置

证书和私钥的生成
1. 创建服务器证书 密钥文件 server.key

```
[root@C7--01 ~]# openssl genrsa -des3 -out server.key 1024
Generating RSA private key, 1024 bit long modulus
.................................++++++
...++++++
e is 65537 (0x10001)
Enter pass phrase for server.key:                    #输入密码 密码直接输入这里输入123.com
Verifying - Enter pass phrase for server.key:        #再次输入密码 密码直接输入这里输入123.com
```

2. 创建服务器证书的 申请文件 server.csr

```
[root@C7--01 ~]# openssl req -new -key server.key -out server.csr

Enter pass phrase for server.key:
#输入serve.key生成的密码
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
```

```
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]:CN            #国家代号，中国输入CN
State or Province Name (full name) []:BeiJing      #省的全名，拼音
Locality Name (eg, city) [Default City]:BeiJng       #市的全名，拼音
Organization Name (eg, company) [Default Company Ltd]:MyCompany Corp.
#公司英文名
Organizational Unit Name (eg, section) []:       #可以不输入;组织单位
Common Name (eg, your name or your server's hostname) []:www.bene.com
#输入域名，如：www.bene.com
Email Address []:QQ@123456.com                #电子邮箱，可随意填

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:                    #可以不输入
An optional company name []:                  #可以不输入
```

3. 备份一份服务器密钥文件

```
[root@C7--01 ~]# cp server.key server.key.org
```

4. 去除文件口令

```
[root@C7--01 ~]# openssl rsa -in server.key.org -out server.key
Enter pass phrase for server.key.org:       #输入server.key
writing RSA key
```

5. 生成证书文件 server.crt

```
[root@C7--01 ~]# openssl x509 -req -days 365 -in server.csr -signkey
server.key -out server.crt
Signature ok
subject=/C=CN/ST=BeiJing/L=BeiJng/O=MyCompany
Corp./CN=www.bene.com/emailAddress=QQ@123456.com
Getting Private key
[root@C7--01 ~]# ls

server.crt  server.csr  server.key  server.key.org
```

6. 配置 nginx.com 文件

```
[root@C7--01 ~]# vim /usr/local/nginx/conf/nginx.conf

worker_processes  1;
events {
    worker_connections  1024;
}
http {
    include       mime.types;
    default_type  application/octet-stream;
    sendfile        on;
    keepalive_timeout  65;
```

```
      log_format  main  '$http_user_agent' '$request_uri' '$remote_addr -
$remote_user [$time_local] "$request" '
                        '$status $body_bytes_sent "$http_referer" '
                        '"$http_user_agent" "$http_x_forwarded_for"'
'"$upstream_cache_status"';


    server {
        listen       443 ssl;        #比起默认的80使用了443默认是ssl方式
        ssl_certificate      ssl/server.crt;    #证书（公钥，发送到客户端）
        ssl_certificate_key  ssl/server.key;    #私钥
        server_name  www.benet.com;

        charset UTF-8;
        location / {
        root   html;
        index  index.html index.htm;
        }

        error_page   500 502 503 504  /50x.html;
        location = /50x.html {
            root   html;
        }
    }
}
[root@C7--01 ~]# nginx -s reload
[root@C7--01 ~]# mkdir -p /usr/local/nginx/conf/ssl
[root@C7--01 ~]# cp server.crt server.key /usr/local/nginx/conf/ssl/
```

注意：ssl on； nginx1.15版本之前需要加，之后的不用加

```
[root@C7--01 ~]# nginx -s reload

nginx: [warn] the "ssl" directive is deprecated, use the "listen ... ssl"
directive instead in /usr/local/nginx/conf/nginx.conf:19
```

注意：如果保存文件时报错ssl那么进行下面的操作

```
    [root@C7--01 ~]# nginx -s stop        #先停止nginx服务
    [root@C7--01 ~]# cd /usr/src/nginx-1.18.0/

    ----------#####---------进行重新编译安装----配置模块ssl------#####

    [root@C7--01 nginx-1.18.0]# ./configure --prefix=/usr/local/nginx --
    user=nginx --group=nginx --with-file-aio --with-
    http_stub_status_module --with-http_gzip_static_module --with-
    http_flv_module --with-http_ssl_module --with-pcre &&make install
```

```
[root@C7--01 ~]# nginx
```

==========

https://www.xinnet.com/knowledge/1622188676.html

SSL协议，在建立传输链路时，首先对 对称加密的密钥 进行 非对称加密，链路建立好后，SSL 对传输内容 使用 对称加密。
一般web 应用都采用 单向认证，因为 用户数目广泛，且无需在 通讯层 进行 用户身份验证，一般都在应用逻辑层 来保护用户的 合法登入。
如果是 企业应用对接，情况就不一样，可能要求 对客户端 做身份验证。这时 就需要 双向认证。

SSL单向认证过程

1. 客户端向 服务器发送 SSL 协议版本号，加密算法种类，随机数 等信息
2. 服务端 给客户端 返回 SSL 协议版本号， 加密算法种类，随机数 等信息， 同时也返回 服务端 的证书， 即 公钥证书
3. 客户端使用 服务端 返回的 信息 验证 服务器的 合法性，验证通过后，继续进行 通信；验证不通过 则终止通信，验证内容包括：
    a. 证书是否过期
    b. 发行 服务器证书的 CA 是否可靠
    c. 返回的 公钥 是否能正确解开 返回证书中的 数字签名
    d. 服务器证书上的 域名 是否和 服务器的 实际域名 相匹配
4. 客户端 向 服务器 发送 自己 所能支持的 对称加密方案，供服务器选择。
5. 服务器在 客户端 提供的 加密方案中 选择 加密程度最高的 加密方式
6. 服务器将 选择好的 加密方式 通过 明文 方式 返回给 客户端
7. 客户端接收到 服务器返回的 加密方案后，使用 该加密方案 产生随机码，用作 通信过程中 对称加密的 密钥，使用 服务器的 公钥 进行加密，将加密后的 随机码 发送到 服务器。
8. 服务器收到 客户端 返回的 加密信息后，使用自己的 私钥 进行解密，获取 对称 加密密钥。在接下来的 会话中，服务器和 客户端 将会使用 该密码 进行 对称加密，保证 通信过程中的 信息安全。

SSL双向认证过程

1. 客户端向 服务器发送 连接请求 (SSL 协议版本号，加密算法种类，随机数 等信息)
2. 服务器给客户端 返回 服务器端的 证书，即 公钥证书，同时也返回 证书相关信息 (SSL 协议版本号，加密算法种类，随机数 等信息)
3. 客户端使用 服务端 返回的 信息 验证 服务器的 合法性 (首先检查 服务器发送过来

的 证书 是否是由 自己信赖 的 CA 中心 所签发的， 再比较 证书里的 信息，例如 域名 和 公钥，与服务器 刚刚发送的 相关信息 是否一致，如果是 一致的，客户端认可这个 服务端的 合法身份)，验证 通过后，可以继续通信，否则 终止通信，验证的内容包括：

a. 证书是否过期
b. 发行服务器证书 的CA 是否可靠
c. 返回的 公钥 是否能正确 解开 返回证书 中的 数字签名
d. 服务器证书上的 域名 是否和 服务器的 实际域名 相匹配

4. 服务端 要求客户端 发送 客户端的证书，客户端会将 自己的证书 发送到 服务端。
5. 服务器 验证 客户端的证书，通过验证后，会得到 客户端的 公钥。
6. 客户端 向服务器 发送自己所能支持的 对称加密方案，供服务器端 进行选择
7. 服务器 选择 客户端提供的 加密方案中 选择 加密等级 最高的 加密方式
8. 将 加密方式 通过 之前 获得 的公钥（客户端的 公钥）进行加密，返回给 客户端
9. 客户端 收到 服务器端 返回的 加密方案密文后，使用 自己的 私钥 进行解密，获取具体的加密方式，而后 获取该加密方式的 随机码，用作 加密过程中的 密钥，使用之前 从服务端 证书中 获取 的 公钥进行 加密后，发送给 服务端。
10. 服务器收到 客户端 的消息后，使用自己的 私钥 进行解密，获取 对称加密的 密钥，在接下来的 会话中，服务器和 客户端 将会 使用 该密码 进行 对称加密，保证 通信过程中的 信息的安全。

单向 和 双向 的区别

单向认证 只要求 站点部署了 SSL 证书 即可，任何用户都可以去访问 (IP 被限制除外等)， 只是 服务器提供了 身份认证。

双向认证 要求 服务器 和 客户端 提供 身份认证，只能是 服务器允许的 客户端 去访问，安全性相对高一点。

双向认证 要求 服务器和客户端 双方 都有 证书。 单向认证 不需要 客户端 拥有 CA证书，只需要 将服务器 验证 客户端证书的 过程去掉， 以及在 协商 对称密码方案，对称密钥 时， 服务器发送给 客户端的 是 没有 加过密 的 (这不影响 SSL 过程的 安全性) 密码方案。 这样， 双方具体的 通讯内容，就是 加过密的 数据，如果有 第三方 攻击，获得的也只是 加密的数据，第三方 要获得 有用的信息，就需要 对 加密的 数据 进行解密，这时的安全 就依赖于 密码方案的 安全。 幸运的是，目前所用的 密码方案，只要 通讯密钥 足够长，就足够安全，这也是 强调使用 128位加密通讯的 原因。

==========

==========

https://www.jianshu.com/p/487e52059ead
ELK之logstash和filebeat的证书验证

生成新文件夹，进入新文件夹

生成ca私钥	openssl genrsa 2048 > ca.key
使用ca私钥建立ca证书	openssl req -new -x509 -nodes -days 1000 -key ca.key -subj /CN=elkCA\ CA/OU=Development\ group/O=HomeIT\ SIA/DC=elk/DC=com > ca.crt 。。但是这里 \ 分不清，不知道是换行还是什么。而且内容也得自己换。所以 实际用了下面的，应该不影响，后面的相同，-subj 都被删除了。感觉应该要全部都一样的内容 openssl req -new -x509 -nodes -days 1000 -key ca.key > ca.crt
生成服务器csr证书请求文件	openssl req -newkey rsa:2048 -days 1000 -nodes -keyout server.key -subj /CN=server.t.com/OU=Development\ group/O=Home\ SIA/DC=elk/DC=com > server.csr
使用ca证书与私钥签发服务器证书	openssl x509 -req -in server.csr -days 1000 -CA ca.crt -CAkey ca.key -set_serial 01 > server.crt
生成客户端csr证书请求文件	openssl req -newkey rsa:2048 -days 1000 -nodes -keyout client.key -subj /CN=client.t.com/OU=Development\ group/O=Home\ SIA/DC=elk/DC=com > client.csr
使用ca证书和私钥签发客户端证书	openssl x509 -req -in client.csr -days 1000 -CA ca.crt -CAkey ca.key -set_serial 01 > client.crt 请将命令中的两个域名按实际情况进行修改，这里需要使用域名，不然会报错 it doesn't contain any IP SANs，如果没有域名，可以在/etc/hosts中配置一下： server.t.com 服务器域名，配置在logstash的input字段中； client.t.com 客户端域名，配置在filebeat.yml文件中。 。。不知道，我没有遇到问题，没有要求写域名啊。不知道自签的证书能不能使用。。

完成后，目录下有8个文件

```
[root@web filebeat_crt]# ll
总用量 32
-rw-r--r-- 1 root root 1350 1月  8 21:20 ca.crt
-rw-r--r-- 1 root root 1679 1月  8 21:20 ca.key
-rw-r--r-- 1 root root 1216 1月  8 21:20 client.crt
-rw-r--r-- 1 root root 1013 1月  8 21:20 client.csr
-rw-r--r-- 1 root root 1704 1月  8 21:20 client.key
```

```
-rw-r--r-- 1 root root 1216 1月  8 21:20 server.crt
-rw-r--r-- 1 root root 1013 1月  8 21:20 server.csr
-rw-r--r-- 1 root root 1704 1月  8 21:20 server.key
```

把文件复制到 filebeat 和 logstash 目录中

```
[root@web filebeat_ssl]# cp -r /root/filebeat_ssl/ /etc/filebeat/

[root@web filebeat_ssl]# cp -r /root/filebeat_ssl/ /etc/logstash/
```

配置logstash的input配置

```
[root@web ~]# vim /etc/logstash/conf.d/01-logstash-listen-5045.conf
input {
beats {
port => 5045
ssl => true
ssl_certificate_authorities => ["/etc/logstash/conf.d/filebeat_ssl/ca.crt"]
ssl_certificate => "/etc/logstash/conf.d/filebeat_ssl/server.crt"
ssl_key => "/etc/logstash/conf.d/filebeat_ssl/server.key"
ssl_verify_mode => "force_peer"
}
}
```

重启logstash

```
[root@web ~]# systemctl restart logstash
```

修改filebeat配置文件

```
[root@web ~]# vim /etc/filebeat/filebeat.yml
output.logstash:
hosts: ["server.t.com:5045"]
ssl.certificate_authorities: ["/etc/filebeat/filebeat_ssl/ca.crt"]
ssl.certificate: "/etc/filebeat/filebeat_ssl/client.crt"
ssl.key: "/etc/filebeat/filebeat_ssl/client.key"
```

重启filebeat

```
[root@web ~]# systemctl restart filebeat
```

==========

ssl_protocols
设置 TLS 版本
似乎是nginx 的

需要在Web服务器的证书配置文件中找到ssl_protocols

==========

nginx
ssl_verify_client on; #双向认证

==========

==========

==========

https://www.cnblogs.com/6b7b5fc3/p/12716291.html
OkHttp配置HTTPS访问+服务器部署

==========

==========

==========

==========

==========

==========

==========

==========

==========

==========

==========