# HTTPS

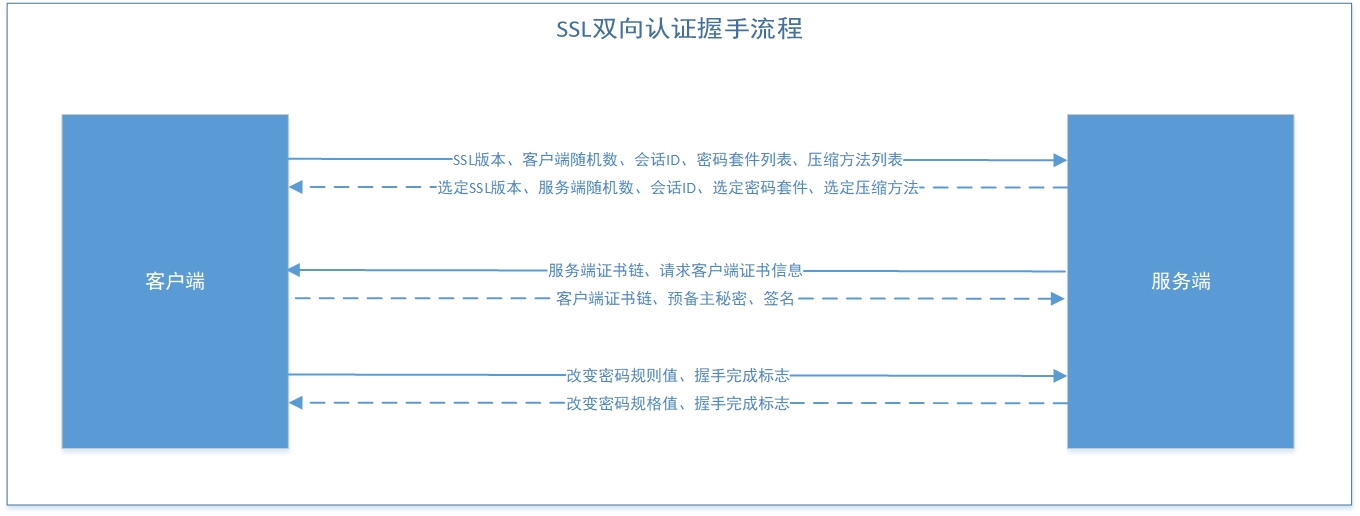
## 综述

HTTPS是应用层的安全协议，主要用于在网络上传输一些机密性的信息，其中使用了证书、加密、签名操作，可以保证防伪装、防窃听、防窜改。而HTTPS实质就是SSL/TLS+HTTP，所有有关的安全交互都在SSL/TLS协议中有具体的规定，而HTTP是不具备安全通信的功能，所以HTTPS的安全特性都是依赖与SSL/TLS协议的。

## SSL

SSL协议介于传输层和应用层间的协议，其中使用了证书验证防伪装、非对称加密传递主秘密（包括分组初始化向量、对称加密密钥、HMAC密钥）、对称加密防窃听、HMAC消息验证码防窜改；而TLS协议则是SSL协议的继承者，其交互流程大致相同，但是TLS增加了若干安全交互中的字段，安全性能比SSL更好。SSL协议中主要包括三个协议：握手协议、通信协议、警报协议，这里主要介绍SSL双向认证的握手协议，记录协议和警报协议做简单介绍。

### 握手协议



1、协商阶段：该阶段由客户端发出请求，服务端做出响应。主要是协商后续通信过程中需要的SSL版本、密码套件、压缩方法和生成主密码所需的2个随机数。

**请求：**

SSL版本：客户端可以支持的最高SSL版本。

客户端随机数：一个用于生成主秘密的32字节随机数。

会话ID：用于确定会话的会话ID。

压缩方法列表：客户端所支持的压缩方法列表。

密码套件列表：客户端支持的密码套件列表，其中主要包括客户端所支持的非对称加密算法、

对称加密算法和散列算法。

**响应：**

选定SSL版本：选定双方所支持的最高SSL版本中的较低版本做为后续通信的SSL版本。

服务端随机数：一个用于生成主秘密的32字节随机数。

会话ID：用于确定会话的会话ID。

选定压缩方法：根据双方所支持的压缩方法，选定一个压缩算法。

选定密码套件：根据双方所支持的密码套件列表，选定一个密码套件（包含了非对称加密算

法、对称加密算法、散列算法）。

2、证书、秘密交换阶段：该阶段由服务端请求（该请求可能分为多次请求发送），客户端响应（该响应可能分为多次响应发送）。主要是交换双方的证书和共享预备主秘密。

**请求：**

服务端证书链：服务端把自身证书到根证书整个证书链发送给客户端。

请求客户端证书信息：其中包括可接受证书列表（可接受的证书类型）和可接受验证列表（可

验证的证书类型）。

**响应：**客户端在接受到服务端证书后，就会对证书进行校验（校验过程后续ca部分重点介绍），当校验通过后才会做出响应。

客户端证书链：客户端把自身证书到根证书的整个证书链响应给服务端。

签名：用自身私钥对预备主秘密和服务端随机数进行签名，保证预备主秘密不被窜改。

预备主秘密：客户端把生成的预备主秘密传递到服务端（用服务端公钥加密），之后双方就可以

根据预备主秘密和双方在协商阶段协商的随机数生成相同主秘密（原理参照

Diffie-Hellman密钥交换算法），然后由主秘密作为伪随机数生成器的种子，可以

得到分组初始化向量、HMAC消息认证码密钥和对称加密密钥（并且双方各自有

各自的三个密钥，都是用对方的密钥解密，自身的密钥加密，保证了通信的单向

性）。

3、完成阶段：该阶段主要是通知对方后续的通信都使用以上两阶段协商好的密钥和算法进行通信，并且说明握手阶段完成。在该阶段进行之前，服务端也会对客户端的证书进行校验，成功后才会有后续通信。

改密密码规格值：通知对方后续的通信都使用以上两阶段协商好的密钥和算法进行通信。

完成标志：说明握手阶段完成。

### 记录协议

记录协议在客户端和服务端握手成功后使用，即客户机和服务器鉴别对方和确定安全信息交换使用的算法后，进入SSL记录协议，记录协议向SSL连接提供两个服务：

**保密性：**使用握手协议定义的对称加密密钥和分组初始化向量加密。

**完整性：**握手协议定义了HMAC密钥，用于保证消息完整性。

### 警报协议

客户端和服务端发现错误时，向对方发送一个警报消息。如果是致命错误，则算法立即关闭SSL连接，双方还会先删除相关的会话号，秘密和密钥。每个警报消息共2个字节，第1个字节表示错误类型，如果是警报，则值为1，如果是致命错误，则值为2；第2个字节制定实际错误类型。

## 数字证书

### 名词解析

数字证书认证在网络通信当中起到了确认通信双方真实身份并防止伪装的作用。在以下的介绍中涉及到很多名词，这里先对这些名词做一些介绍。

#### CA机构

ca机构就是具有签发ca证书权力的机构，其中又分为CA认证机构和中间认证机构。

**CA认证机构：**CA认证机构是受行业承认的认证机构，该机构可以安全的保管本身的密钥，并且

可以用密钥签发数字证书，另外还会向外发布根证书。

**中间认证机构：**中间认证机构是受CA认证机构承认的认证机构，同样可以签发CA证书，但中间机

构的证书不是根证书，而是中间证书（具有签发权的CA证书）。

#### 证书

证书中主要包括了发布者及其机构信息、所有者及其机构信息、有效期、公钥主体、证书指纹等信息。发布者是签发该证书的机构、所有者是使用该证书的机构、有效期记录了证书的发布时间和失效时间、公钥主体就是非对称加密中需要使用的公钥、证书指纹就是用散列函数处理证书信息得到的散列值。另外证书还分为根证书、中间证书和数字证书。

**根证书：**根证书是自己给自己签发的证书，其发布者和使用者都是自己，所以根证书的拥有机构必

须是受行业承认的机构。

**中间证书：**中间证书是根证书签发的CA证书。

**CA证书：**普通的CA证书，其对应的私钥不具备签发权，即不可以用普通CA证书签发CA证书，并

且只有普通CA证书才参与通信（加密、解密、签名、验签）。

**证书链：**由于证书中都有使用者和发布者，因此每个证书都可以找到自己的发布者，从而可以形成一条从数字证书到根证书的链接，这就是证书链。

#### JKS

JKS是一种java中存储密钥的文件格式，根据文件中存储的条目类型可以分为keyStore和trustStore（从本质上来说他们都是一样的，因为是同一种文件）

**keyStore：**当JKS文件中条目类型为PrivatetKeyEntry时，这个JKS就是keyStore，其中除了私钥外还

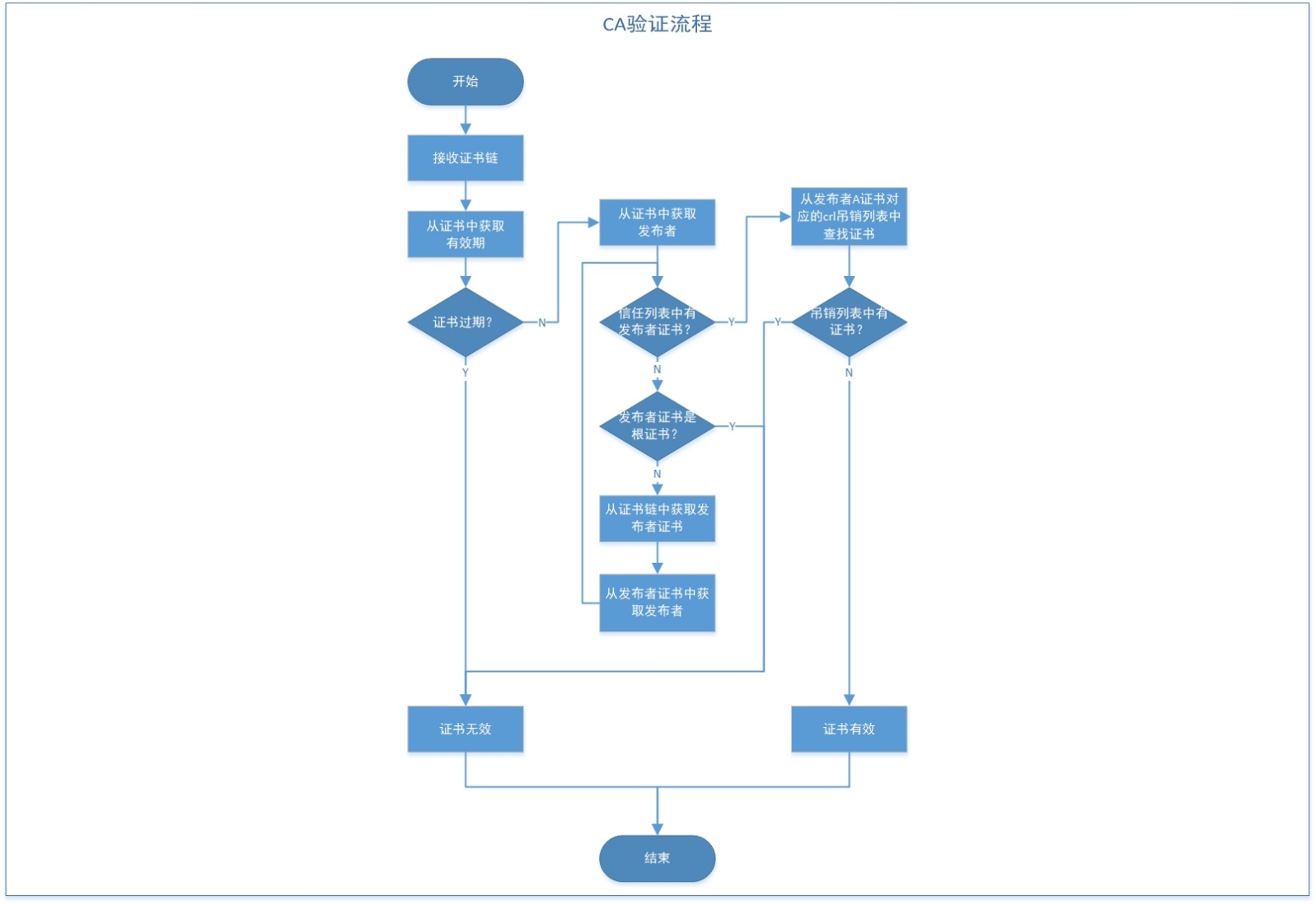
有私钥对应的证书。

**trustStore：**当JKS文件中条目类型为trustedCertEntry时，这个JKS就是trustStore，其中一般存储根

证书或中间证书，类似与浏览器中的证书信任列表。

### 证书校验

证书校验过程中主要校验有效期、是否受信任、是否被吊销三项，浏览器中还会校验机构名称和域名是否对应，还有其他更复杂的校验。这里主要介绍前三项校验过程。如图：



有效期校验：主要通过证书中的失效时间和当前时间对比，判断是否过期。

是否受信任：主要通过校验证书的发布者对应的证书是否是在本机的信任列表（在Java中即

trustStroe）中；若不在还可以通过证书链找到发布者证书的发布者，并再次校验是否

信任该发布者对应的证书，直到追寻到根证书依然不受信任，那么该证书无效。

是否被吊销：证书发布者发布的crl吊销列表，验证该证书是否已经被吊销。在浏览器中内置信任的

证书机构一般会定时同步crl吊销列表给浏览器厂商，然后等到浏览器升级时厂商就

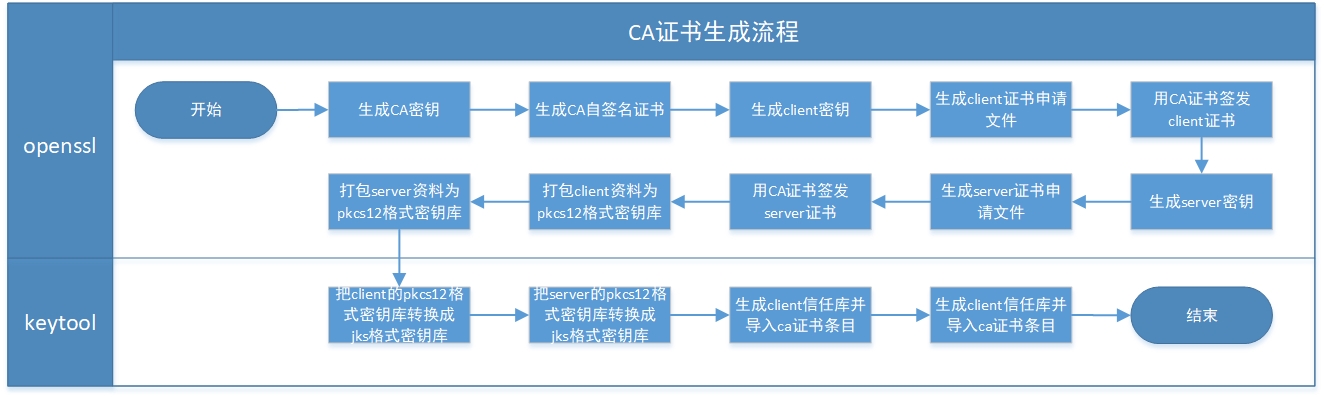
会把同步的crl吊销列表更新到新版本的浏览器中，因此该验证是有一定的时延的。

### 证书生成

#### 工具

本次例子中使用了OpenSSL和keytool两个工具。OpenSSl主要用于生成CA根证书（用于给客户端、服务端证书签名）、客户端证书、服务端证书；而keytool主要用于把OpenSSL生成的pkcs12类型的密钥库导入到jks类型的密钥库，并且生成保存数字证书的信任库。

#### 流程图



#### 流程文件映射表

|  |  |
| --- | --- |
| 流程 | 生成文件 |
| 生成CA密钥 | cakey.pem |
| 生成CA自签名证书 | ca.cer |
| 生成client密钥 | clientkey.pem |
| 生成client证书申请文件 | client.csr |
| 用CA证书签发client证书 | clinet.crt |
| 生成server密钥 | serverkey.pem |
| 生成server证书申请文件 | server.csr |
| 用CA证书签发server证书 | server.crt |
| 打包client资料为pkcs12格式密钥库 | client.pkcs12 |
| 打包server资料为pkcs12格式密钥库 | server.pkcs12 |
| 把client的pkcs12格式密钥库转换成jks格式密钥库 | client.jks |
| 把server的pkcs12格式密钥库转换成jks格式密钥库 | server.jks |
| 生成client信任库并导入ca证书条目 | clienttrust.jks |
| 生成server信任库并导入ca证书条目 | servertrust.jks |

#### OpenSSL

以下代码中统一用caPath代替存放生成文件的文件夹路径。使用时需要替换成自己实际的文件夹路径

1、生成CA密钥

**openssl genrsa –out caPath\ca.pem 2048**

2、生成CA自签名证书，注意此步骤和4、7步不一样 ,并且需要输入机构信息

**openssl req -x509 -new -key caPath\ca.pem -out ca.crt**

3、生成client密钥

**openssl genrsa -out caPath\clientkey.pem 2048**

4、生成client证书申请文件，需要定义机构信息 、证书请求密码

**openssl req -new -key caPath\clientkey.pem -out caPath\client.csr**

5、用CA证书签发client证书

**openssl x509 -req -in caPath\client.csr -CA caPath\ca.crt -CAkey caPath\cakey.pem -CAcreateserial -days 3650 -out caPath\client.crt**

6、生成server密钥

**openssl genrsa -out caPath\serverkey.pem 2048**

7、生成server证书申请文件，需要定义机构信息 、证书请求密码

**openssl req -new -key caPath\serverkey.pem -out caPath\server.csr**

8、用CA证书签发server证书

**openssl x509 -req -in caPath\server.csr -CA caPath\ca.crt -CAkey caPath\cakey.pem -CAcreateserial -days 3650 -out caPath\server.crt**

9、打包client资料为pkcs12格式密钥库，需要定义pkcs12密钥库密码

**openssl pkcs12 -export -in caPath\client.crt -inkey caPath\clientkey.pem -out caPath\client.pkcs12**

10、打包server资料为pkcs12格式密钥库，需要定义pkcs12密钥库密码

**openssl pkcs12 -export -in caPath\server.crt -inkey caPath\serverkey.pem -out caPath\server.pkcs12**

#### KeyTool

11、把client的pkcs12格式密钥库转换成jks格式密钥库，需要输入pkcs12密钥库密码，定义jks密钥库密码

**keytool -importkeystore -srckeystore caPath\client.pkcs12 -destkeystore caPath\client.jks -srcstoretype pkcs12**

12、把server的pkcs12格式密钥库转换成jks格式密钥库，需要输入pkcs12密钥库密码，定义jks密钥库密码

**keytool -importkeystore -srckeystore caPath\server.pkcs12 -destkeystore caPath\server.jks -srcstoretype pkcs12**

13、生成client信任库并导入ca证书条目

**keytool -importcert -alias ca -file caPath\ca.crt -keystore caPath\clienttrust.jks**

14、生成server信任库并导入ca证书条目

**keytool -importcert -alias ca -file caPath\ca.crt -keystore caPath\servertrust.jks**

PS：最终代码中需要使用的是client.jks、clienttrust.jks、server.jks、servertrust.jks四个密钥库，查看密钥库中信息可以用以下代码密钥库信息，其中可以查看到密钥库的类型和条目的的类型(PrivateKeyEntry为私钥即密钥库中条目，trustedCertEntry为证书即信任库中条目)

**keytool -list -v -keysrore caPath\client.jks**