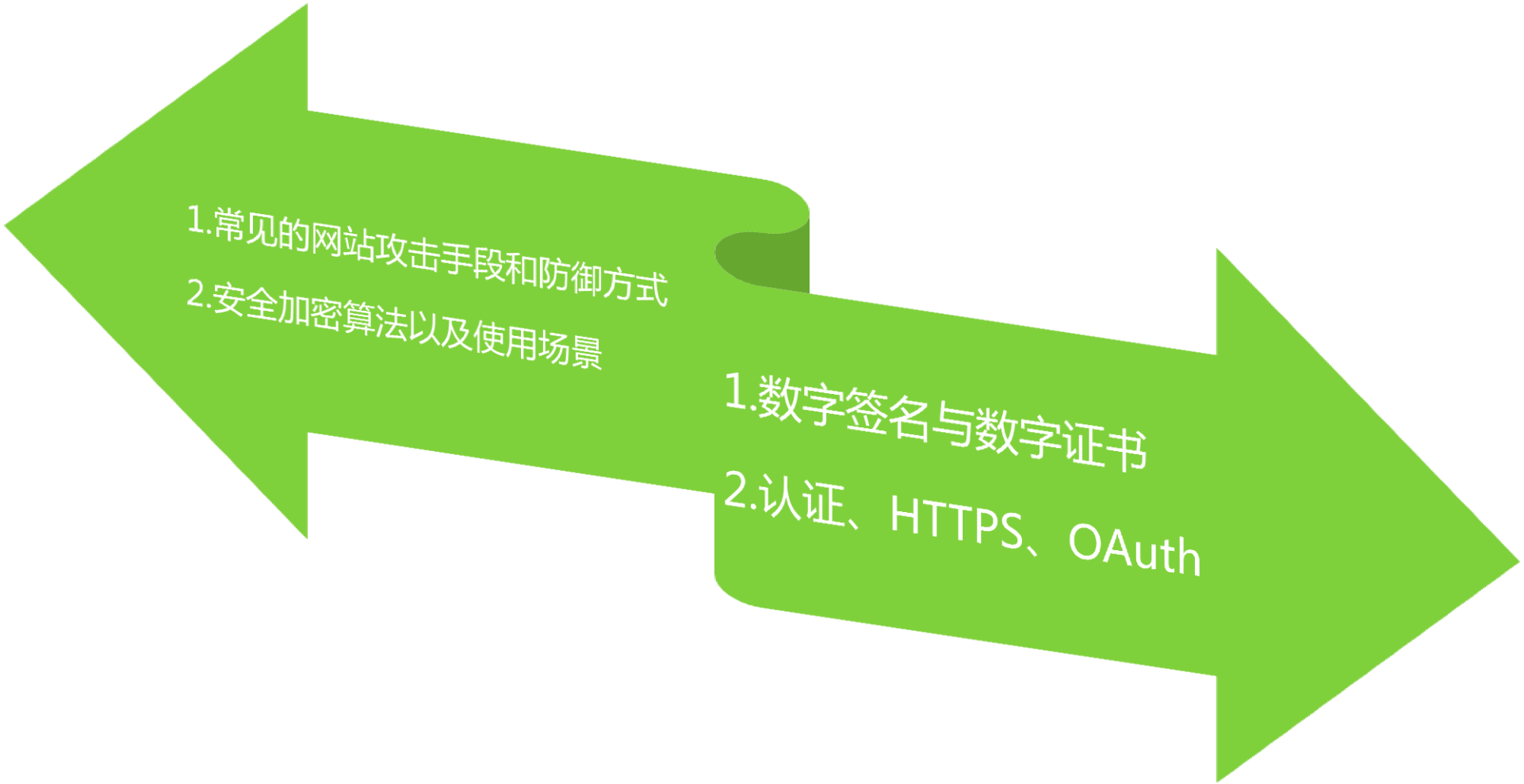
大型电商分布式系统实践 第4周

DATAGURU专业数据分析社区

上节内容回顾


1.常见的网站攻击手段和防御方式
2.安全加密算法以及使用场景
1.数字签名与数字证书
2.认证、HTTPS、OAuth

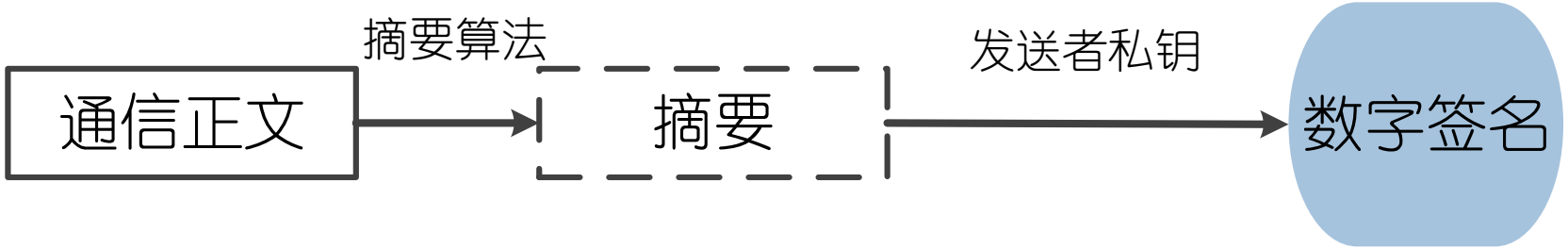
常见安全算法—数字签名

签名认证是对非对称加密技术与数字摘要技术的综合运用，指的是将通信内容的摘要信息使用发送者的私钥进行加密，然后将密文与原文一起传输给信息的接收者，接收者通过发送者的公钥解密被加密的摘要信息，然后使用与发送者相同的摘要算法，对接收到的内容采用相同的方式产生摘要串，与解密的摘要串进行对比，如果相同，则说明接收到的内容是完整的，在传输过程中没有受到第三方篡改，否则则说明通信内容已被第三方修改。

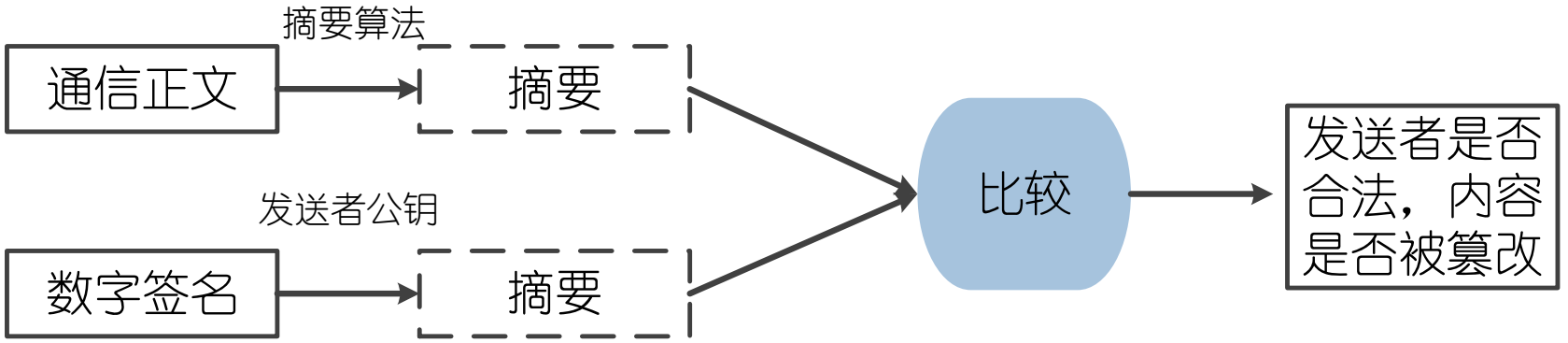
每个人都有其特有的私钥且都是对外界保密的，而通过私钥加密的的信息，只能通过其对应的公钥才能解密，因此，私钥可以代表私钥持有者的身份，可以通过私钥对应的公钥来对私钥拥有者的身份进行校验。通过数字签名，能够确认消息是由信息发送方签名并发送出来的，因为其他人根本假冒不了消息发送方的签名，他们没有消息发送者的私钥。而不同的内容，摘要信息千差万别，通过数字摘要算法，可以确保传输内容的完整性，如果传输内容中途被篡改，对应的数字签名的值也将发生改变。

常见安全算法—数字签名

只有信息的发送者才能产生别人无法伪造的数字签名串，这个串能对信息发送者所发送的内容完整性以及发送者的身份进行校验和鉴别。



通信正文经过相应的摘要算法生成摘要后，使用消息发送者的私钥进行加密，生成数字签名。



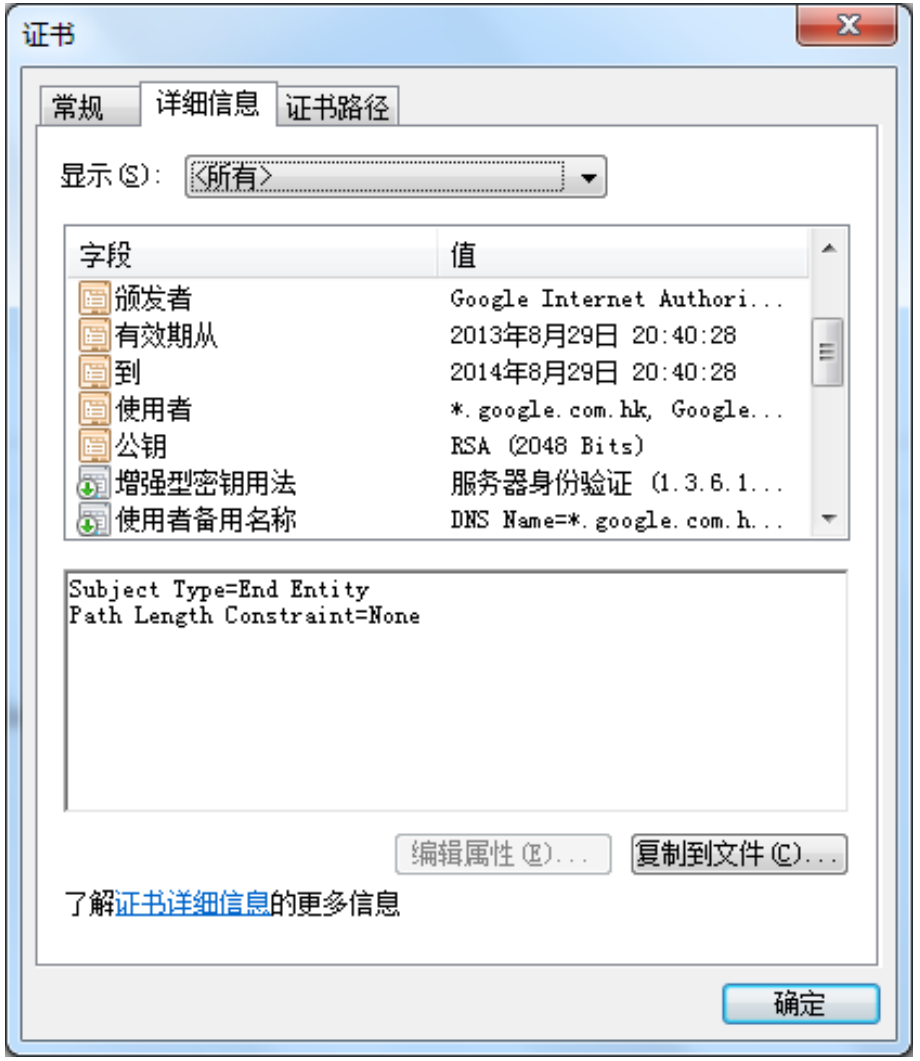
常见安全算法—数字证书

我们每个人都有很多形式的身份证明，如身份证、驾驶证、护照等等，这些证件都是由相应的签发机构盖章认证，可信程度较高，很难进行伪造，并且，随着科技的发展，还可以通过指纹、视网膜等生物特征进行身份的认证。

数字证书(Digital Certificate)，也称为电子证书，类似于日常生活中的身份证，也是一种形式的身份认证，用于标识网络中的用户身份。数字证书集合了多种密码学的加密算法，证书自身带有公钥信息，可以完成相应的加密、解密操作，同时，还拥有自身信息的数字签名，可以鉴别证书的颁发机构，以及证书内容的完整性。由于证书本身含有用户的认证信息，因此可以作为用户身份识别的依据。

常见安全算法—数字证书

通常数字证书会包含如下内容：
对象的名称(人，服务器，组织)
证书的过期时间
证书的颁发机构(谁为证书担保)
证书颁发机构对证书信息的数字签名
签名算法
对象的公钥



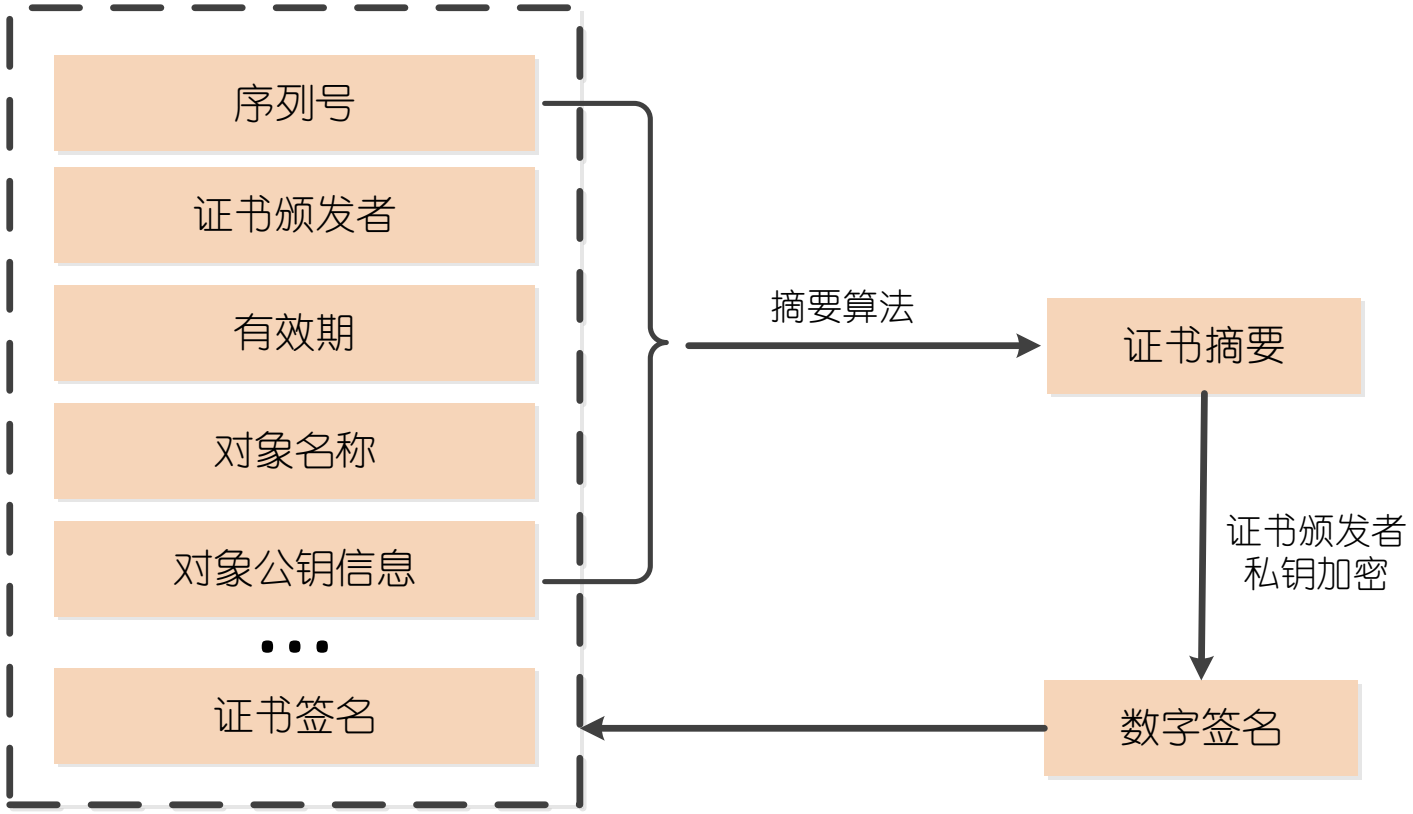
常见安全算法—X.509证书标准

不同的数字证书，所包含的内容信息以及格式可能不尽相同，因此，需要有一种格式标准，来规范数字证书的存储和校验。大多数的数字证书都以一种标准的格式，即X.509，来存储他们的信息。
X.509提供了一种标准的方式，将证书信息规范地存储到一系列可解析的字段当中，X.509 V3是X.509标准目前使用最为广泛的版本。

字段	描述
版本	当前证书的X.509标准的版本号
序列号	证书颁发机构(CA)生成的唯一序列号
签名算法	签名使用的签名算法，如SHA1withRSA
证书颁发者	发布并对该证书进行签名的组织名称
有效期	此证书有效的开始时间和结束时间
对象名称	证书所代表的实体，比如一个人或者是一个组织
对象的公钥信息	证书对象的公钥，生成公钥的算法，以及附加参数
发布者ID(可选)	证书颁发者的唯一标识符
对象ID(可选)	证书所代表对象的唯一标识符
扩展字段	可选的扩展字段集
证书颁发机构数字签名	证书的颁发机构使用指定的签名算法以及颁发机构的私钥对上述所有字段生成的数字签名

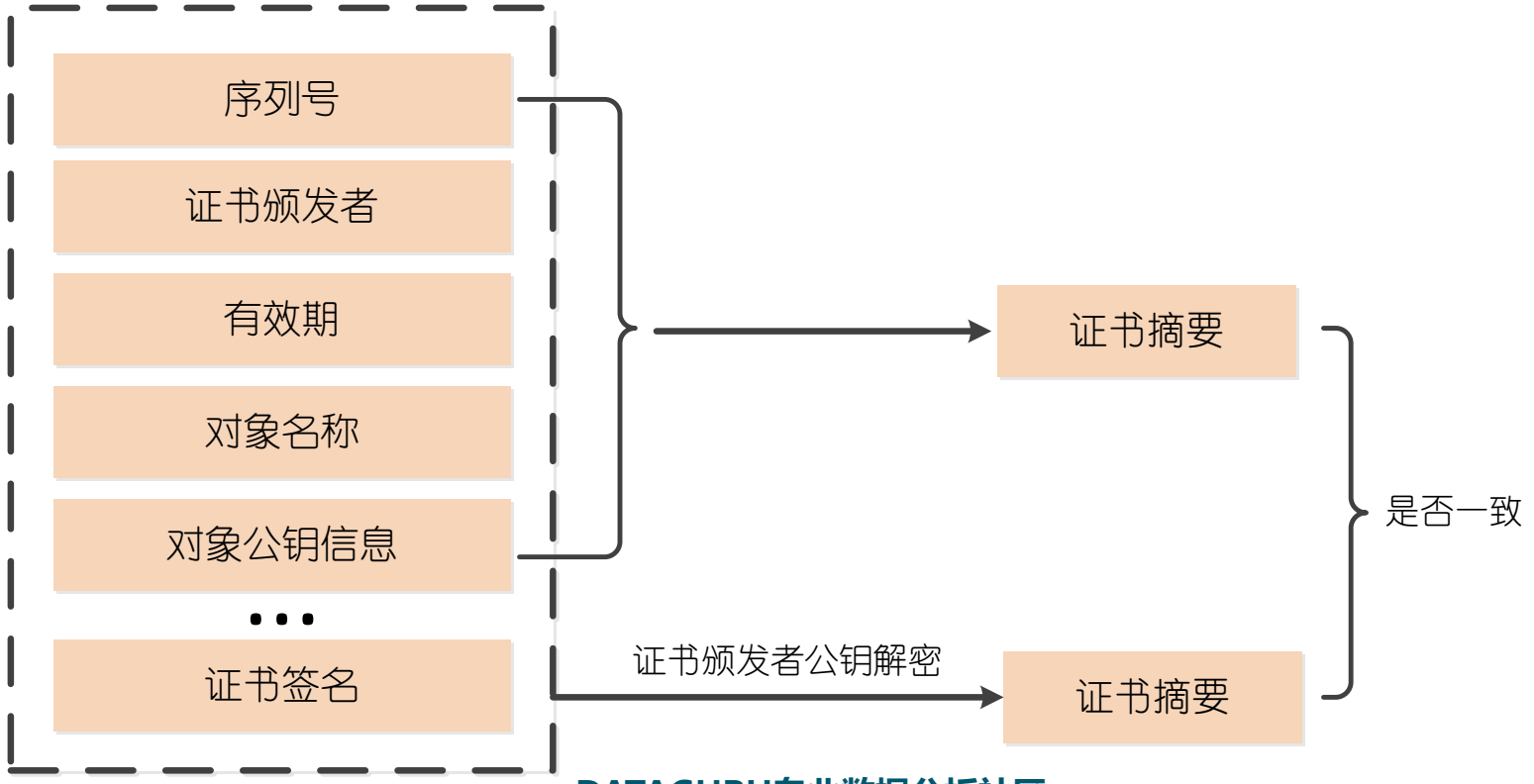
常见安全算法—证书签发

现实生活中，我们的身份证件需要由相应的政府机关进行签发，而网络用户的数字证书则需要由数字证书认证机构(Certificate Authority，CA)来进行颁发，只有经过CA颁发的数字证书在网络中才具备有可认证性。数字证书的签发过程实际上就是对数字证书的内容，包括证书所代表对象的公钥，进行数字签名，而验证证书的过程，实际上是校验证书的数字签名，包含对证书有效期的验证。



常见安全算法—证书校验

客户端接收到数字证书时，首先会对证书的认证机构进行检查，如果该机构是权威的证书认证机构，则通过该权威认证机构的根证书，获得证书颁发者的公钥，通过该公钥，对证书的数字签名进行校验，如图所示，并验证证书的有效时间是否过期。根证书是证书认证机构给自己颁发的数字证书，是证书信任链的起始点，安装根证书则意味着对这个证书认证机构的信任。



常见安全算法—证书管理

任何机构或者个人都可以申请数字证书，并使用数字证书对网络通信保驾护航，要获得数字证书，首先需要使用数字证书管理工具，如KeyTool、OpenSSL等等，构建CSR(Certificate Signing Request，数字证书签发申请)，提交给数字证书认证机构进行签名，最终形成数字证书。

keytool

KeyTool是java的数字证书管理工具，用于数字证书的生成、导入、导出以及撤销等操作。它与本地密钥库关联，并可以对本地密钥库进行管理，可以将私钥存放于密钥库，而公钥则使用数字证书进行输出。

OpenSSL

OpenSSL包含一个开源的SSL协议的实现，虽然OpenSSL使用SSL作为其名字的重要组成部分，但其实现的功能却远远超出了SSL协议本身。OpenSSL事实上包括了三个组成部分：SSL协议库、密码算法库以及各种与之相关的应用程序。关于SSL协议，后面介绍HTTPS协议的时候会介绍到。同时作为一个基于密码学的安全开发包，OpenSSL提供的功能相当强大和全面，囊括了主要的密码算法、常用的密钥和证书封装管理功能以及SSL协议，并提供了丰富的应用程序供测试或其它目的使用。

Keystore的使用

在构建CSR之前，需要先在密钥库中生成本地数字证书。生成本地数字证书需要提供用户的身份、加密算法、有效期等等一些数字证书的基本信息，这里使用www.codeaholic.net作为别名，使用RSA算法作为加密算法，并使用1024位的密钥，使用MD5withRSA作为数字签名算法，证书的有效期设置为365天

```
keytool -genkeypair -keyalg RSA -keysize 1024 -sigalg MD5withRSA -validity 365 -alias www.codeaholic.net -keystore /tmp/chenkangxian.keystore
```

参数介绍：

-genkeypair 产生密钥对
-keyalg 加密算法，这里用的是RSA算法
-keysize 密钥大小，这里设置为1024位
-sigalg 签名算法，这里用的是MD5withRSA
-validity 证书有效期，这里指定365天
-alias 别名，这里用的是www.codeaholic.net
-keystore 指定密钥库的位置，此处为/tmp/chenkangxian.keystore

openssl的安装

OpenSSL包含一个开源的SSL协议的实现，虽然OpenSSL使用SSL作为其名字的重要组成部分，但其实现的功能却远远超出了SSL协议本身。OpenSSL事实上包括了三个组成部分：SSL协议库、密码算法库以及各种与之相关的应用程序。关于SSL协议，后面介绍HTTPS协议的时候会涉及，此处暂且不表。作为一个基于密码学的安全开发包，OpenSSL提供的功能相当强大和全面，囊括了主要的密码算法、常用的密钥和证书封装管理功能以及SSL协议，并提供了丰富的应用程序供测试或其它目的使用。这里主要是使用OpenSSL来进行密钥和证书的管理。

安装命令集合：

```
sudo apt-get remove openssl
wget http://mirrors.ibiblio.org/openssl/source/openssl-1.0.1e.tar.gz
tar -xf openssl-1.0.1e.tar.gz
./config
make
make install
```

openssl生成私钥

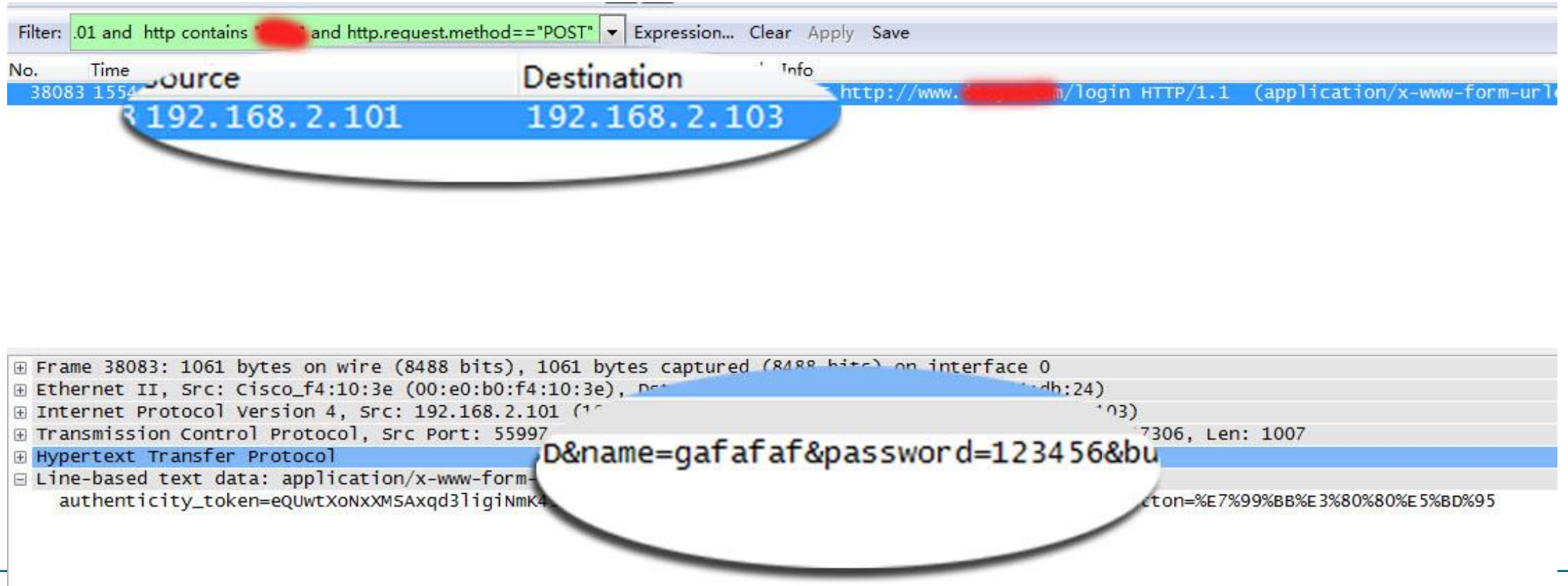
openssl genrsa -aes256 -out private/cakey.pem 1024

```
longlong@ubuntu:~/ca_dir$ openssl genrsa -aes256 -out private/cakey.pem 1024
Generating RSA private key, 1024 bit long modulus
............++++++
....++++++
e is 65537 (0x10001)
Enter pass phrase for private/cakey.pem:
Verifying - Enter pass phrase for private/cakey.pem:
longlong@ubuntu:~/ca_dir$
```

参数含义如下：
genrsa　表示使用RSA算法产生私钥
-aes256 表示使用256位密钥的AES算法对私钥进行加密
-out　　表示输出文件的路径
1024用来指定生成私钥的长度

为什么需要认证

经由HTTP协议进行通信的数据大都是未经加密的明文，包括请求参数、返回值、cookie、head等等数据，因此，外界通过对通信的监听，轻而易举便可根据请求和响应双方的格式，伪造请求与响应，修改和窃取各种信息。相对于基于TCP协议层面的通讯方式，针对HTTP协议的攻击门槛更低。因此，基于HTTP协议的WEB及SOA架构，在应用的安全性方面，需要更加的重视。

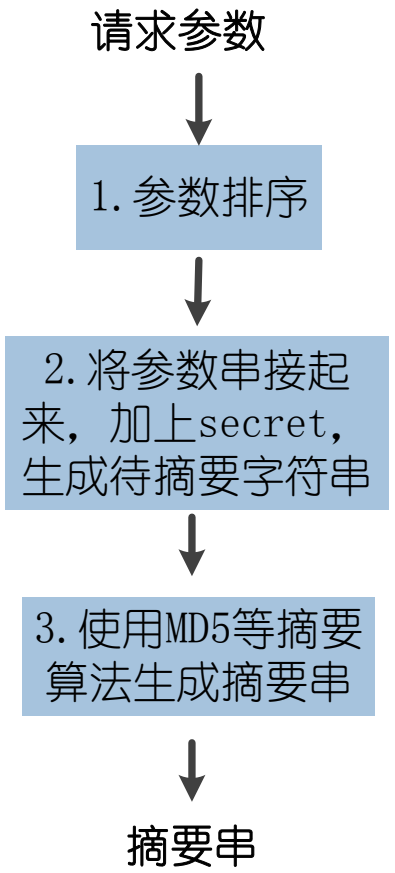


摘要认证

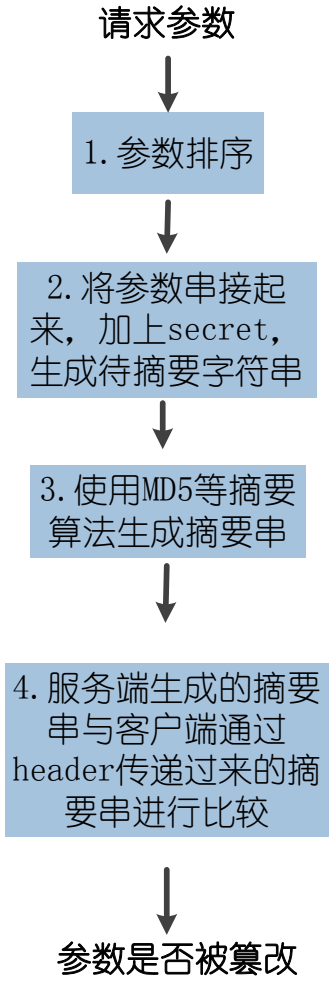
对于普通的非敏感数据，我们更多的关注其真实性和准确性，因此，如何在通信过程中保障数据不被篡改，便成为首当其冲需要考虑的问题。
鉴于使用HTTPS性能上的成本以及需要额外申请CA证书，这种情况下，一般采用对参数和响应进行摘要的方法，即能够满足需求 。

由于摘要算法的不可逆性，并且，大部分情况下不同的请求参数，会有不同的服务端响应，鉴于参数和响应的多变性，因此，摘要认证这种方式能够在一定程度上防止信息被篡改，保障通信的安全。但是，摘要认证的安全性取决于secret的安全性，由于服务端与客户端采用的是相同的secret，一旦secret泄露，通信的安全则无法保障。

服务端校验

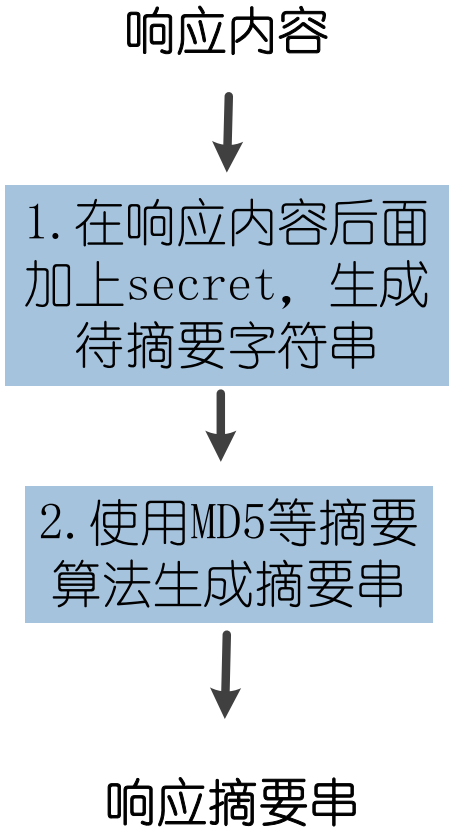


1-摘要生成

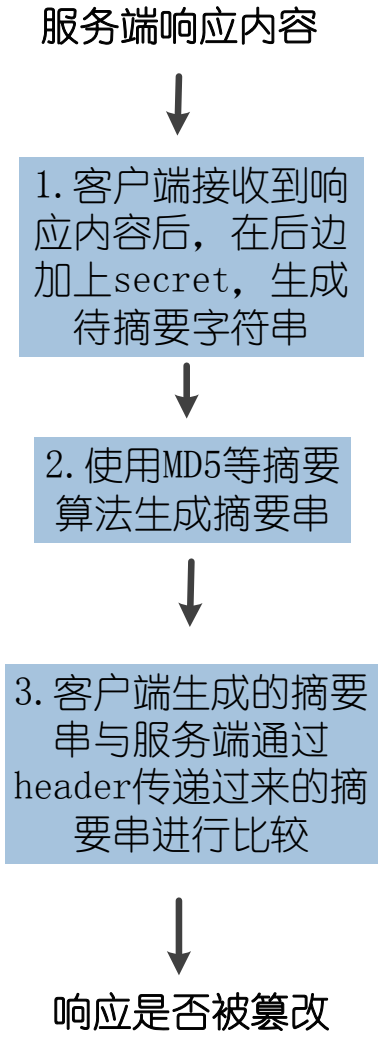


2-服务端验证

客户端校验



1-服务端摘要生成



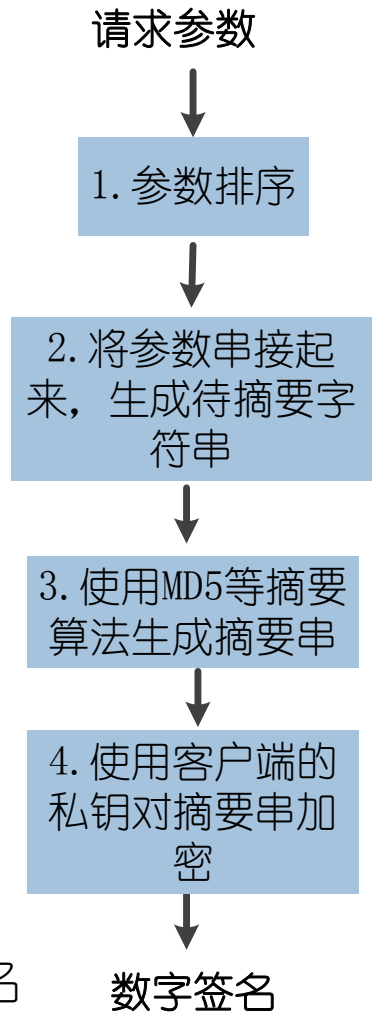
2-客户端验证

签名认证

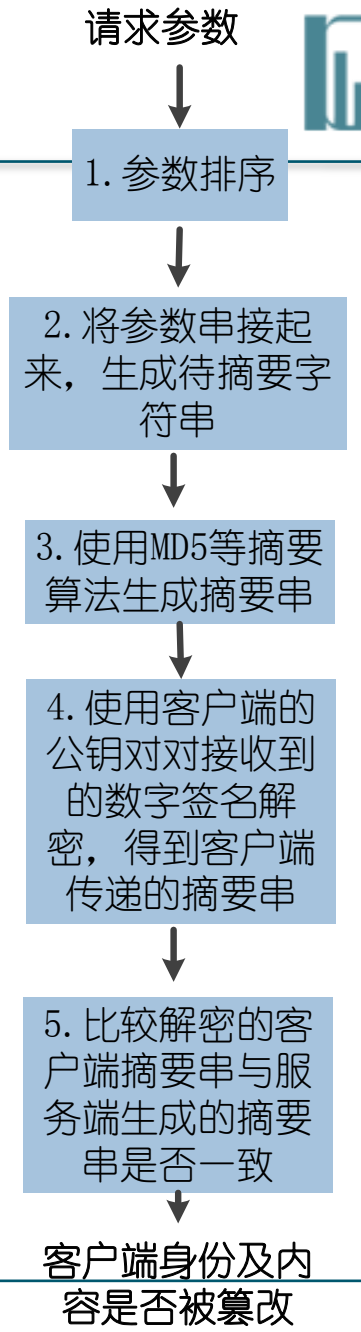
摘要认证的方式能够一定程度上防止通信的内容被篡改，但是，算法的安全性取决于secret的安全性，由于通信的客户端与服务端采用的是相同的secret，一旦secret泄露，恶意攻击者便可以根据相应的摘要算法，伪造出合法的请求或响应的摘要，达成攻击目的。

与摘要认证的方式类似，由于传递端和接收端都认为HTTP协议的请求参数是无序的，因此对于签名认证来说，客户端与服务端双方需要约定好参数的排序方式，请求的参数经过排序后，再将参数名称和值经过一定的策略组织起来，这时不再是加上secret，而是直接通过约定的摘要算法生成数字摘要，并且使用客户端私钥对数字摘要进行加密，将加密的密文传递给服务端。

签名认证

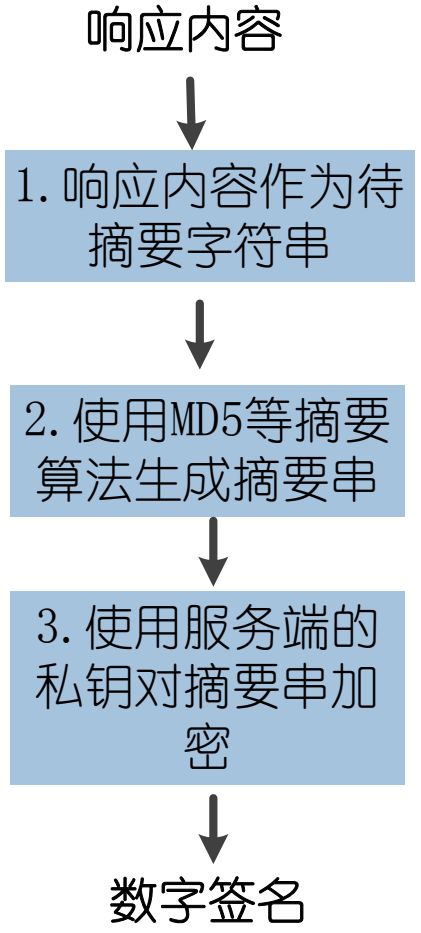


1- 客户端请求数字签名

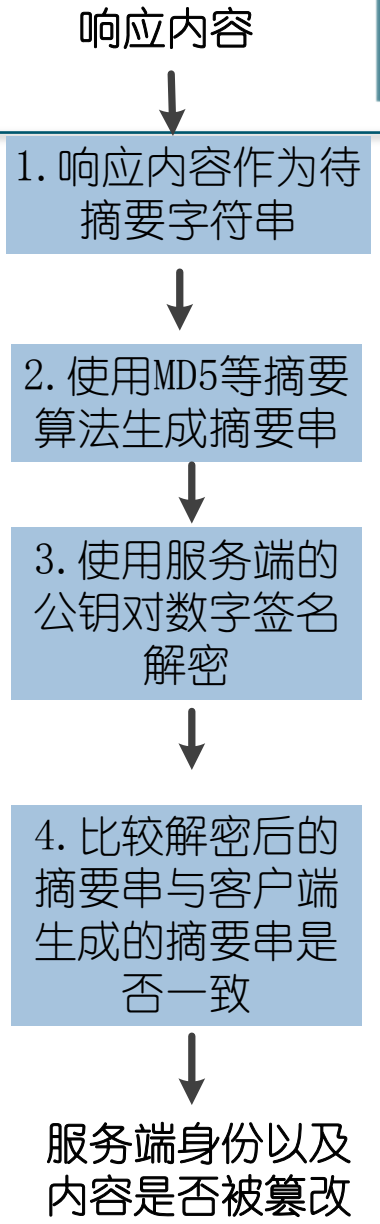


2 - 服务端校验数字签名

签名认证

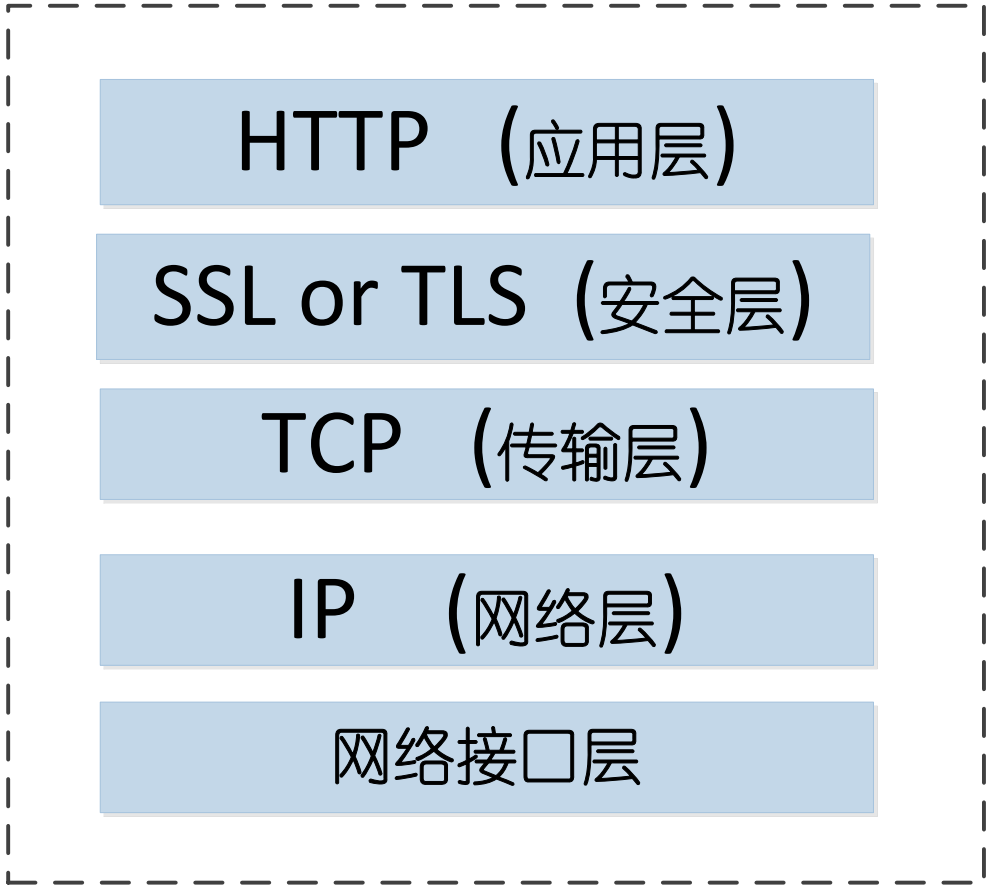


1- 服务端响应数字签名



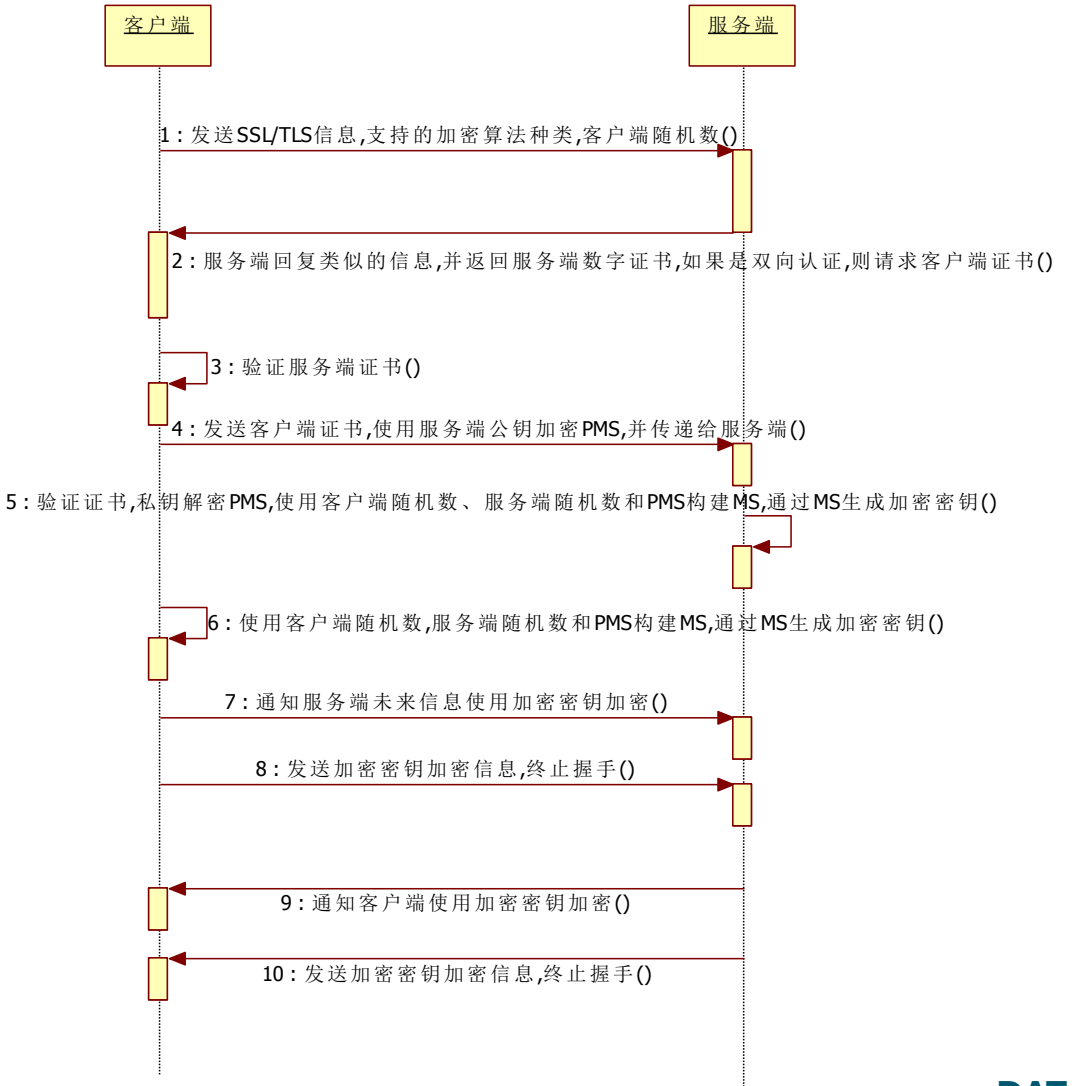
2 - 客户端校验数字签名

HTTPS协议



HTTPS协议在HTTP协议与TCP协议增加了一层安全层，所有请求和响应数据在经过网络传输之前，都会先进行加密， 然后再进行传输。SSL及其继任者TLS是为网络通信提供安全及数据完整性保障的一种安全协议，利用加密技术，以维护互联网数据传输的安全，验证通信双方的身份，防止数据在网络传输的过程中被拦截及窃听。

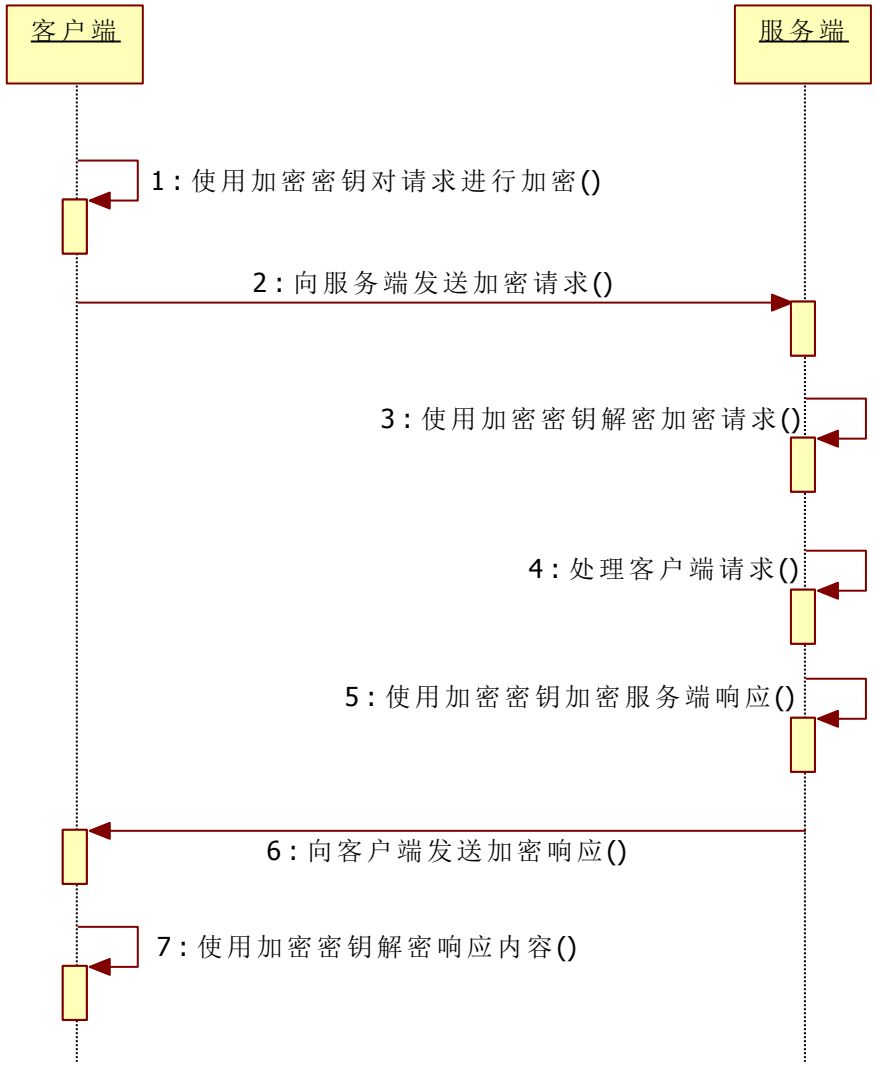
SSL协议—握手



SSL，全称为Secure Sockets Layer，即安全套接层，他是一种网络安全协议，是网景在其推出的Web浏览器中同时提出的，目的是为了保障网络通信的安全，校验通信双方的身份，加密传输的数据，SSL在传输层与应用层之间进行数据通信的加密。

SSL协议的优势在于它与应用层协议独立无关，高层的应用层协议如HTTP、SSH、FTP等等，能透明的建立于SSL协议之上，在应用层通信之前就已经完成加密算法、通信密钥的协商以及服务端及客户端的认证工作，在此之后所有应用层协议所传输的数据都会被加密，从而保证通信的私密性。

SSL协议—数据传输



服务端与客户端真正的数据交换阶段，实际上数据是通过对称加密算法来实现加密的，密钥为双方约定好的加密密钥。客户端首先使用加密密钥加密请求内容，发送给服务端，服务端使用加密密钥对请求进行解密，并处理相应的请求，生成响应内容，响应经过加密密钥加密过后，发送给客户端，客户端通过加密密钥对响应内容进行解密，以获得响应内容。

Tomcat部署HTTPS WEB

修改tomcat配置
cd tomcat/conf
vim server.xml

找到默认注释的HTTPS配置的这一行

```
<!--
  <Connector port="8443" protocol="HTTP/1.1" SSLEnabled="true"
        maxThreads="150" scheme="https" secure="true"
        clientAuth="false" sslProtocol="TLS" />
 -->
```

打开注释，加上keystore的地址以及keystore密码两项，keystoreFile="/home/longlong/temp/testssl/server.keystore"，keystorePass="123456"，并且，将port修改为HTTPS默认的443端口，由于此处使用的keystore为PKCS#12格式，因此需要加上参数keystoreType="pkcs12"，由于这里配置的是单向认证，只需要校验服务端证书的有效性，所以clientAuth一项设置为false。

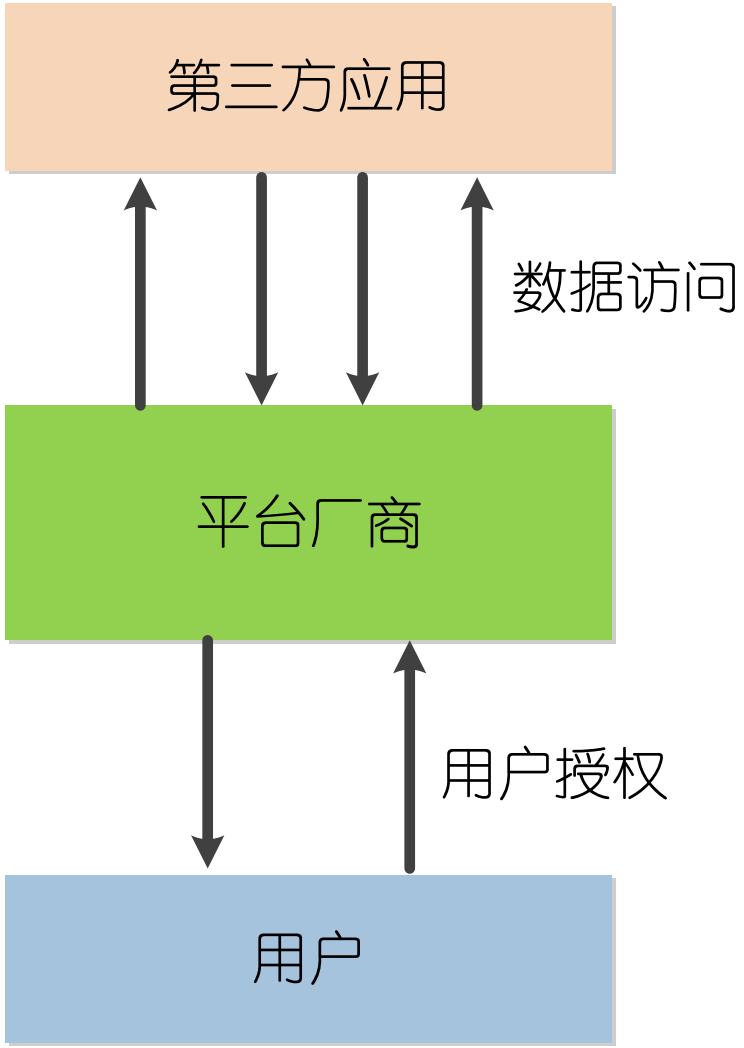
Tomcat部署HTTPS WEB

```
<Connector port="443" protocol="HTTP/1.1" SSLEnabled="true"
        maxThreads="150" scheme="https" secure="true"
        clientAuth="false" sslProtocol="TLS"
        keystoreFile="/home/longlong/temp/testssl/server.keystore"
        keystorePass="123456"
        keystoreType="pkcs12"/>
```

ubuntu从10.04起，默认关闭1024以下的端口，需要安装authbind，才能使用相应的端口。authbind是GNU下的一个小工具，用于帮助系统管理员来为程序指定端口。

```
sudo apt-get install authbind
sudo touch /etc/authbind/byport/443
sudo  ./shutdown.sh
sudo authbind --deep ./startup.sh
```

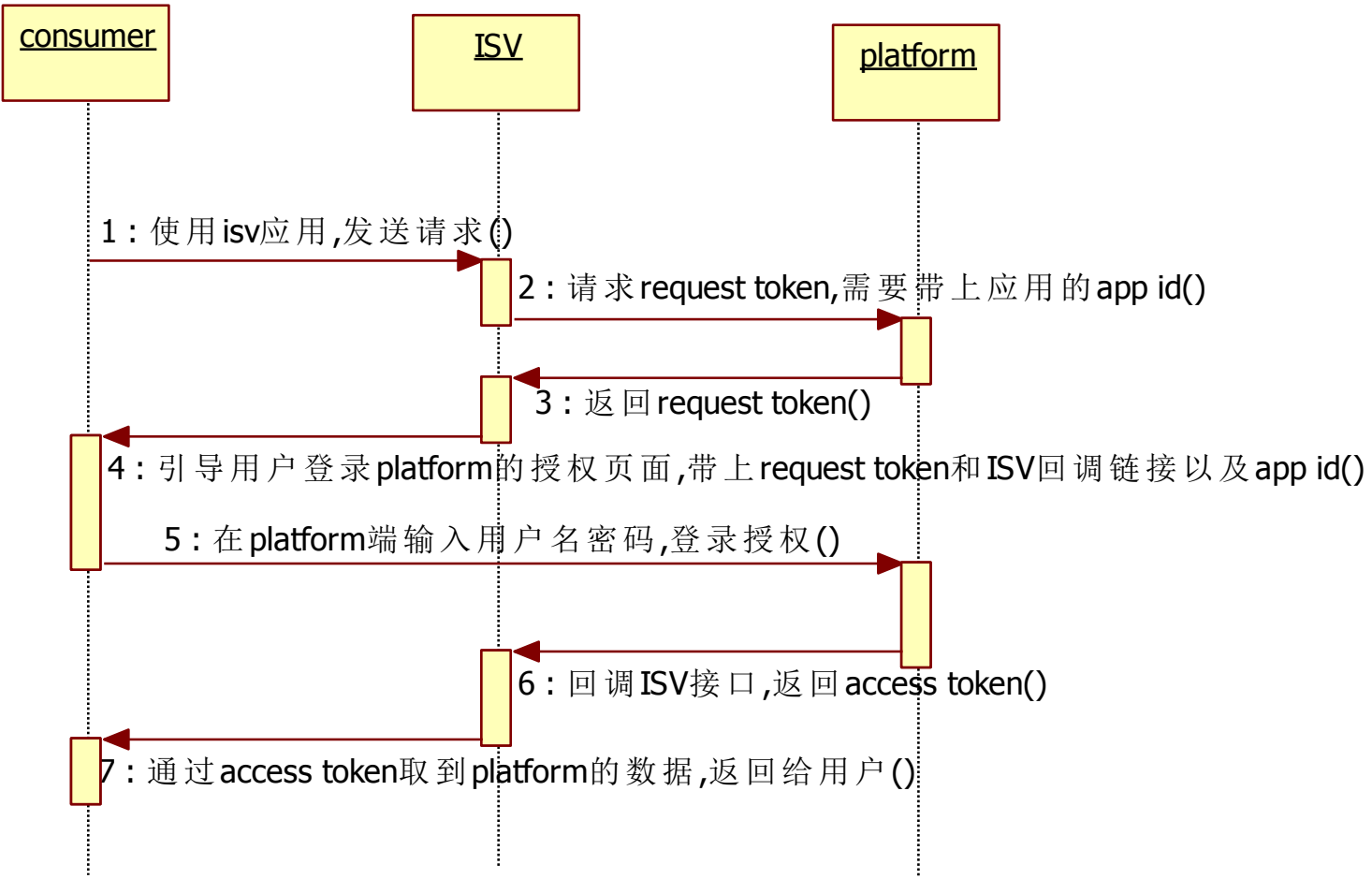
OAuth协议



OAuth协议旨在为用户资源的授权访问提供一个安全、开放的标准。平台商通过OAuth协议，提示用户对第三方软件厂商(ISV)进行授权，使得第三方软件厂商能够使用平台商的部分数据，对用户提供服务。与以往的授权方式不同，OAuth协议并不需要触及到用户的账户信息，即用户名密码，便可以完成第三方对用户信息访问的授权。

用户通过平台商，对第三方应用进行授权，而第三方应用得到授权后，便可以在一定的时间内，通过平台商提供的接口，访问到用户授权的信息，为用户提供服务。

OAuth协议



要获得授权，首先需要第三方开发者向平台商申请应用ID，对自己的APP进行注册。一次OAuth授权涵盖了三个角色：普通用户(consumer)、第三方应用(ISV)、平台商(platform)。OAuth对ISV授权数据访问过程包含了如下几个步骤：

1. 用户对ISV的应用进行访问，发起请求。
2. ISV收到请求后，向平台商请求request token，并带上其申请的appId。
3. 平台将返回给第三方应用request token。
4. ISV应用将用户引导到平台授权页面，带上自己的appId、request token以及回调地址。
5. 用户在平台的页面上进行登录，并且完成授权。
6. 平台通过ISV提供的回调链接，返回给ISV应用access token。
7. ISV应用通过access token取到用户授权的数据，进行加工后返回给用户，授权数据访问完成。

Thanks

FAQ时间