iOS 中 HTTPS 证书验证浅析

原创 2017-01-12 李晴 腾讯Bugly

导语

在 WWDC 16 中，Apple 表示, 从 2017年1月1日起（最新消息, 实施时间已延期），所有新提交的 App 使用系统组件进行的 HTTP 网络请求都需要是 HTTPS 加密的，否则会导致请求失败而无法通过审核。

精神哥对 HTTPS 的验证过程有一些了解，但对于在iOS中如何实现 HTTPS 验证却不是很清楚，在内网搜索到李晴同学写的这篇文章，阅读后收获不小，分享给大家。

正文

本文的目的：一是简要分析下对服务器身份验证的完整握手过程，二是证书链的验证，三是探索下iOS中原生库NSURLConnection或NSURLSession如何支持实现https。

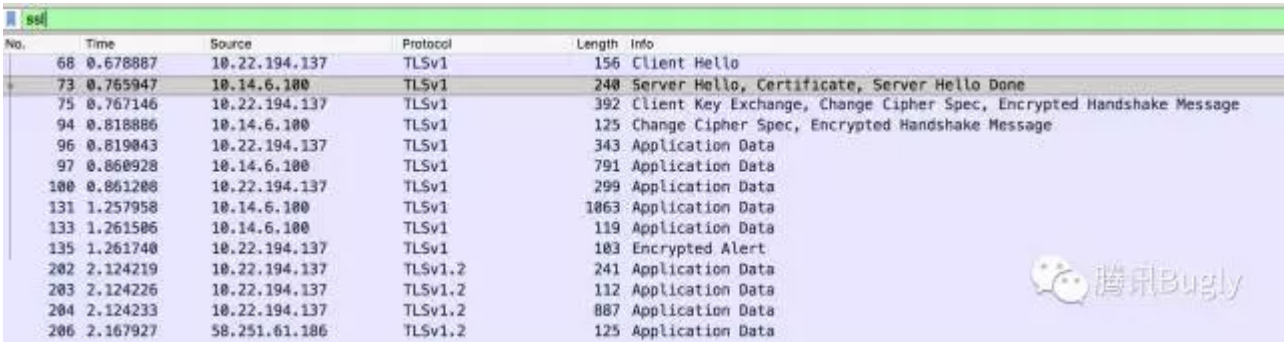
一、HTTPS

HTTPS是承载在TLS/SSL之上的HTTP，相较于HTTP明文数据传输方面所暴露出的缺点，HTTPS具有防止信息被窃听、篡改、劫持，提供信息加密，完整性校验及身份验证等优势。TLS/SSL是安全传输层协议，介于TCP和HTTP之间。TLS1.0是建立在SSL3.0规范之上的，可以理解为SSL3.0的升级版本。目前推荐使用的版本是TLS1.2。

TLS/SSL协议通常分为两层：TLS记录协议(TLS Record Protocol)和TLS握手协议(TLS Handshake Protocol)。 TLS记录协议建立在可靠的传输协议（如TCP）之上，为高层协议提供数据封装、压缩、加密等基本功能的支持。TLS握手协议建立在记录协议之上，用于在实际的数据传输开始前，通讯双方进行身份认证、协商加密算法、交换加密密钥等。除了这俩协议以外，还存在其它三种辅助协议： Changecipher spec 协议用来通知对端从handshake切换到record协议(有点冗余，在TLS1.3里面已经被删掉了)。alert协议，用来通知各种返回码。application data协议，就是把http，smtp等的数据流传入record层做处理并传输。

想象一种场景：通常我们会访问HTTPS://xxx的网站，当你在浏览器地址栏输入支持HTTPS协议的URL地址后，服务器返回的数据会显示在页面上。对于不了解HTTPS协议工作原理的小伙伴可能觉得这个过程很简单：发送请求－服务器响应请求－结果返回并显示。但对于HTTPS而言，在整个发送请求返回数据过程中还涉及到通讯双方证书验证、数据加密、数据完整性校验等。

下面以登录qq邮箱为例，通过Wireshark抓包可以看到如下图：

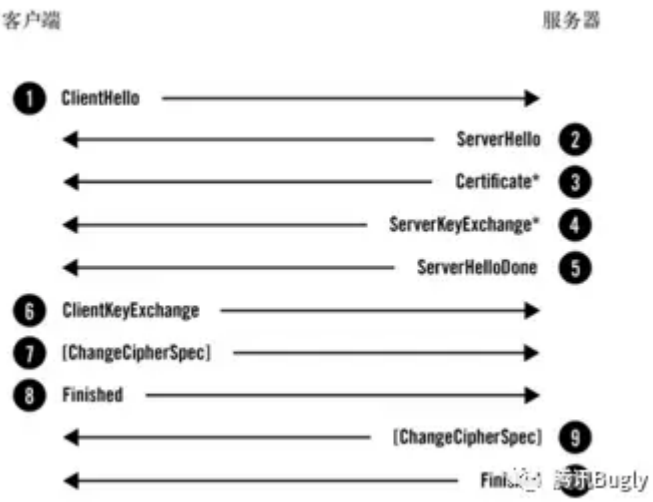


No.	Time	Source	Protocol	Length	Info
68	0.678887	10.22.194.137	TLSv1	156	Client Hello
73	0.765947	10.14.6.180	TLSv1	240	Server Hello, Certificate, Server Hello Done
75	0.767146	10.22.194.137	TLSv1	392	Client Key Exchange, Change Cipher Spec, Encrypted Handshake Message
94	0.818886	10.14.6.180	TLSv1	125	Change Cipher Spec, Encrypted Handshake Message
96	0.819043	10.22.194.137	TLSv1	343	Application Data
97	0.860928	10.14.6.180	TLSv1	791	Application Data
100	0.861208	10.22.194.137	TLSv1	299	Application Data
131	1.257958	10.14.6.180	TLSv1	1063	Application Data
133	1.261586	10.14.6.180	TLSv1	119	Application Data
135	1.261740	10.22.194.137	TLSv1	103	Encrypted Alert
202	2.124219	10.22.194.137	TLSv1.2	241	Application Data
203	2.124226	10.22.194.137	TLSv1.2	112	Application Data
204	2.124233	10.22.194.137	TLSv1.2	887	Application Data
206	2.167927	58.251.61.186	TLSv1.2	125	Application Data

在浏览器与服务器进行Application Data传输之前，还经历了Client Hello－Server Hello－Client Key Exchange－Change Cipher Spec等过程。而这些过程正是TLS/SSL提供的服务所决定的：

- 认证服务器身份，确保数据发送到正确的服务器；
- 加密数据以防止数据中途被窃取；
- 维护数据的完整性，确保数据在传输过程中不被改变。

上述单向验证的完整握手过程，总结如下：



第一阶段：ClientHello

客户端发起请求，以明文传输请求信息，包含版本信息，加密套件候选列表，压缩算法候选列表，随机数random_C，扩展字段等信息。

第二阶段：ServerHello－ServerHelloDone

如上图可以看出这个阶段包含4个过程(有的服务器是单条发送，有的是合并一起发送)。服务端返回协商的信息结果，包括选择使用的协议版本，选择的加密套件，选择的压缩算法、随机数random_S等，其中随机数用于后续的密钥协商。服务器也会配置并返回对应的证书链Certificate，用于身份验证与密钥交换。然后会发送ServerHelloDone信息用于通知服务器信息发送结束。

第三阶段：证书校验

在上图中的5-6之间，客户端这边还需要对服务器返回的证书进行校验。只有证书验证通过后，才能进行后续的通信。(具体分析可参看后续的证书验证过程)

第四阶段：ClientKeyExchange－Finished

服务器返回的证书验证合法后， 客户端计算产生随机数字Pre-master，并用server证书中公钥加密，发送给服务器。同时客户端会根据已有的三个随机数根据相应的生成协商密钥。客户端会通知服务器后续的通信都采用协商的通信密钥和加密算法进行加密通信。然后客户端发送Finished消息用于通知客户端信息发送结束。

第五阶段：服务器端生成协商密钥

服务器也会根据已有的三个随机数使用相应的算法生成协商密钥，会通知客户端后续的通信都采用协商的通信密钥和加密算法进行加密通信。然后发送Finished消息用于通知服务器信息发送结束。

第六阶段：握手结束

在握手阶段结束后，客户端和服务器数据传输开始使用协商密钥进行加密通信。

总结

简单来说，HTTPS请求整个过程主要分为两部分。一是握手过程：用于客户端和服务器验证双方身份，协商后续数据传输时使用到的密钥等。二是数据传输过程：身份验证通过并协商好密钥后，通信双方使用协商好的密钥加密数据并进行通信。在握手过程协商密钥时，使用的是非对称密钥交换算法， 密钥交换算法本身非常复杂，密钥交换过程涉及到随机数生成，模指数运算，空白补齐，加密，签名等操作。在数据传输过程中，客户端和服务器端使用协商好的密钥进行对称加密解密。

二、证书

PKI (Public Key Infrastructure)，公开密钥基础设施。它是一个标准,在这个标准之下发展出的为了实现安全基础服务目的的技术统称为PKI。 权威的第三方机构CA(认证中心)是PKI的核心， CA负责核实公钥的拥有者的信息，并颁发认证”证书”，同时能够为使用者提供证书验证服务。 x.509是PKI中最重要的标准，它定义了公钥证书的基本结构。

证书申请过程

- 证书申请者向颁发证书的可信第三方CA提交申请证书相关信息，包括：申请者域名、申请者生成的公钥(私钥自己保存)及证书请求文件.cer等

- CA通过线上、线下等多种手段验证证书申请者提供的信息合法和真实性。
- 当证书申请者提供的信息审核通过后，CA向证书申请者颁发证书，证书内容包括明文信息和签名信息。其中明文信息包括证书颁发机构、证书有效期、域名、申请者相关信息及申请者公钥等，签名信息是使用CA私钥进行加密的明文信息。当证书申请者获取到证书后，可以通过安装的CA证书中的公钥对签名信息进行解密并与明文信息进行对比来验证签名的完整性。

证书验证过程

- 验证证书本身的合法性（验证签名完整性，验证证书有效期等）
- 验证证书颁发者的合法性（查找颁发者的证书并检查其合法性，这个过程是递归的）

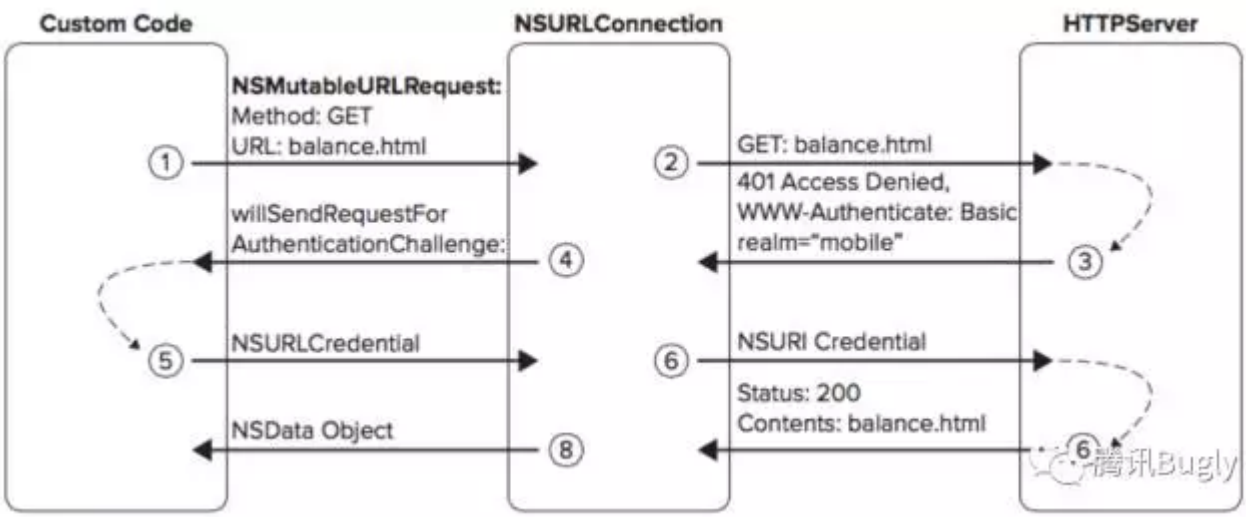
证书验证的递归过程最终会成功终止，而成功终止的条件是：证书验证过程中遇到了锚点证书，锚点证书通常指：嵌入到操作系统中的根证书(权威证书颁发机构颁发的自签名证书)。

证书验证失败的原因

- 无法找到证书的颁发者
- 证书过期
- 验证过程中遇到了自签名证书，但该证书不是锚点证书。
- 无法找到锚点证书(即在证书链的顶端没有找到合法的根证书)
- 访问的server的dns地址和证书中的地址不同

三、iOS实现支持HTTPS

在OC中当使用NSURLConnection或NSURLSession建立URL并向服务器发送https请求获取资源时，服务器会使用HTTP状态码401进行响应(即访问拒绝)。此时NSURLConnection或NSURLSession会接收到服务器需要授权的响应，当客户端授权通过后，才能继续从服务器获取数据。如下图所示：



非自签名证书验证实现

在接收到服务器返回的状态码为401的响应后，对于NSURLSession而言，需要代理对象实现URLSession:task:didReceiveChallenge:completionHandler:方法。对于NSURLConnection而言，需要代理对象实现connection:willSendRequestForAuthenticationChallenge: 方法(OS X v10.7和iOS5及以上)，对于早期的版本代理对象需要实现代理对象要实现connection:canAuthenticateAgainstProtectionSpace:和connection:didReceiveAuthenticationChallenge:方法。代码如下(参考文档)：

```
//建立连接
NSURL * httpsURL = [NSURL URLWithString:@"https://developer.apple.com"];
self.connection = [NSURLConnection connectionWithRequest:[NSURLRequest requestWithURL:httpsURL] delegate:self];


//回调
- (void)connection:(NSURLConnection *)connection willSendRequestForAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {
    //1)获取trust object
    SecTrustRef trust = challenge.protectionSpace.serverTrust;
    SecTrustResultType result;

    //2)SecTrustEvaluate对trust进行验证
    OSStatus status = SecTrustEvaluate(trust, &result);
    if (status == errSecSuccess &&
        (result == kSecTrustResultProceed ||
        result == kSecTrustResultUnspecified)) {

        //3)验证成功，生成NSURLCredential凭证cred，告知challenge的sender使用这个凭证来继续连接
        [challenge.sender useCredential:[NSURLCredential credentialForTrust:trust] forAuthenticationChallenge:challenge];

    } else {
        //5)验证失败，取消这次验证流程
        [challenge.sender cancelAuthenticationChallenge:challenge];
    }
}
```

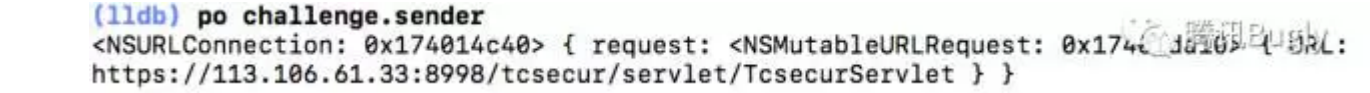

当客户端发送https请求后，服务器会返回需要授权的相关信息，然后connection:willSendRequestForAuthenticationChallenge:方法被调用。客户端根据返回的challenge信息，首先获取需要验证的信任对象trust,然后调用SecTrustEvaluate方法是用系统默认的验证方式对信任对象进行验证，当验证通过时，使用该信任对象trust生成证书凭证，然后self.connection使用该凭证继续连接。如下详解：

NSURLAuthenticationChallenge包含如下信息：

- error： 最后一次授权失败的错误信息
- failureResponse： 最后一次授权失败的错误信息
- previousFailureCount： 授权失败的次数
- proposedCredential： 建议使用的证书
- protectionSpace： NSURLProtectionSpace对象，代表了服务器上的一块需要授权信息的区域。包括了服务器地址、端口等信息。在此指的是challenge.protectionSpace。其中Auth-scheme指protectionSpace所支持的验证方法，NSURLAuthenticationMethodServerTrust指对protectionSpace执行证书验证。

```
(lldb) po challenge.protectionSpace
<NSURLProtectionSpace: 0x170203120>: Host:113.106.61.33, Server:https, Auth-
Scheme:NSURLAuthenticationMethodServerTrust, Realm:(null), Port:8998, Proxy:NO, Proxy-
Type:(null)
```

- sender:发送者，在此指的是self.connection



SecTrustRef

表示需要验证的信任对象(Trust Object)，在此指的是challenge.protectionSpace.serverTrust。包含待验证的证书和支持的验证方法等。

SecTrustResultType

表示验证结果。其中 kSecTrustResultProceed表示serverTrust验证成功，且该验证得到了用户认可(例如在弹出的是否信任的alert框中选择always trust)。 kSecTrustResultUnspecified表示serverTrust验证成功，此证书也被暗中信任了，但是用户并没有显示地决定信任该证书。 两者取其一就可以认为对serverTrust验证成功。

SecTrustEvaluate

函数内部递归地从叶节点证书到根证书验证。使用系统默认的验证方式验证Trust Object，根据上述证书链的验证可知，系统会根据Trust Object的验证策略，一级一级往上，验证证书链上每一级证书有效性。

NSURLCredential

表示身份验证证书。URL Lodaing支持3种类型证书：password-based user credentials, certificate-based user credentials, 和certificate-based server credentials(需要验证服务器身份时使用)。因此NSURLCredential可以表示由用户名/密码组合、客户端证书及服务器信任创建的认证信息，适合大部分的认证请求。对于NSURLCredential也存在三种持久化机制：

- NSURLCredentialPersistenceNone： 要求 URL 载入系统 “在用完相应的认证信息后立刻丢弃”。
- NSURLCredentialPersistenceForSession： 要求 URL 载入系统 “在应用终止时，丢弃相应的 credential ”。
- NSURLCredentialPersistencePermanent： 要求 URL 载入系统 “将相应的认证信息存入钥匙串（keychain），以便其他应用也能使用。

对于已经验证通过的信任对象，客户端也可以不提供证书凭证。

- 对于NSURLSession，传递如下之一的值给completion handler回调：
  - NSURLSessionAuthChallengePerformDefaultHandling处理请求，就好像代理没有提供一个代理方法来处理认证请求

    - NSURLSessionAuthChallengeRejectProtectionSpace拒接认证请求。基于服务器响应的认证类型，URL加载类可能会多次调用代理方法。
- 对于 NSURLConnection 和 NSURLDownload，在[challenge sender] 上调用continueWithoutCredentialsForAuthenticationChallenge:方法。不提供证书的话，可能会导致连接失败，调用connectionDidFailWithError:方法 ，或者会返回一个不需要验证身份的替代的URL。 如下代码：

```
//3)验证成功，生成NSURLCredential凭证cred，告知challenge的sender使用这个凭证来继续连接
//[challenge.sender useCredential:[NSURLCredential credentialForTrust:trust] forAuthenticationChallenge:challenge];
[challenge.sender continueWithoutCredentialForAuthenticationChallenge:challenge];
```


对于非自签名的证书，即使服务器返回的证书是信任的CA颁发的，而为了确定返回的证书正是客户端需要的证书，这需要本地导入证书，并将证书设置成需要参与验证的锚点证书，再调用SecTrustEvaluate通过本地导入的证书来验证服务器证书是否是可信的。如果服务器证书是这个锚点证书对应CA或者子CA颁发的，或服务器证书本身就是这个锚点证书，则证书信任通过。如下代码(参考文档)：

```
//导入证书
NSString * cerPath = ...; //证书的路径
NSData * cerData = [NSData dataWithContentsOfFile:cerPath];/* 在 App Bundle 中用来置锚点的证书数据，证书是CER编码的，常见扩展名有：cer, crt...*/
SecCertificateRef certificate = SecCertificateCreateWithData(NULL, (__bridge CFDataRef)(cerData));
self.trustedCertificates = @[CFBridgingRelease(certificate)];

//回调
- (void)connection:(NSURLConnection *)connection willSendRequestForAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {
    //1)获取trust object
    SecTrustRef trust = challenge.protectionSpace.serverTrust;
    SecTrustResultType result;

    //将之前导入的证书设置成下面验证的Trust Object的锚点证书
    SecTrustSetAnchorCertificates(trust, (__bridge CFArrayRef)self.trustedCertificates);

    //通过导入的证书来验证服务器证书是否可信
    OSStatus status = SecTrustEvaluate(trust, &result);
    if (status == errSecSuccess &&
        (result == kSecTrustResultProceed ||
        result == kSecTrustResultUnspecified)) {

        //3)验证成功，生成NSURLCredential凭证cred，告知challenge的sender使用这个凭证来继续连接
        [challenge.sender useCredential:[NSURLCredential credentialForTrust:trust] forAuthenticationChallenge:challenge];

    } else {
        //5)验证失败，取消这次验证流程
        [challenge.sender cancelAuthenticationChallenge:challenge];
    }
}
```


自签名证书验证实现

对于自签名证书，这样Trust Object中的服务器证书是不可信任的CA颁发的，直接使用SecTrustEvaluate验证是不会成功的。可以采取下述简单代码绕过HTTPS的验证：

```
//回调
- (void)connection:(NSURLConnection *)connection willSendRequestForAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {
    if ([challenge.protectionSpace.authenticationMethod isEqualToString:NSURLAuthenticationMethodServerTrust])
        {
            [challenge.sender useCredential:[NSURLCredential credentialForTrust:challenge.protectionSpace.serverTrust]
forAuthenticationChallenge:challenge];
            [challenge.sender continueWithoutCredentialForAuthenticationChallenge:challenge];
        }
}
```


上述代码一般用于当服务器使用自签名证书时，为了方便测试，客户端可以通过该方法信任所有自签名证书。

综上对非自建和自建证书验证过程的分析，可以总结如下：

- 获取需要验证的信任对象(Trust Object)。对于NSURLConnection来说，是从delegate方法-connection: willSendRequestForAuthenticationChallenge:回调回来的参数challenge中获取(challenge.protectionSpace.serverTrust) 。
- 使用系统默认验证方式验证Trust Object。SecTrustEvaluate会根据Trust Object的验证策略，一级一级往上，验证证书链上每一级数字签名的有效性，从而评估证书的有效性。
- 如第二步验证通过了，一般的安全要求下，就可以直接验证通过，进入到下一步：使用Trust Object生成一份凭证([NSURLCredential credentialForTrust:serverTrust])，传入challenge的sender中([challenge.sender useCredential:cred forAuthenticationChallenge:challenge])处理，建立连接。
- 假如有更强的安全要求，可以继续对Trust Object进行更严格的验证。常用的方式是在本地导入证书，验证Trust Object与导入的证书是否匹配。
- 假如验证失败，取消此次Challenge-Response Authentication验证流程，拒绝连接请求。
- 假如是自建证书的，则不使用第二步系统默认的验证方式，因为自建证书的根CA的数字签名未在操作系统的信任列表中。

> **参考文档：**
> Overriding TLS Chain Validation Correctly
> Making HTTP and HTTPS Request
> HTTPS Server Trust Evaluation
> NSURLConnection －HTTPS Server Trust Evaluation
> Glossary－HTTPS Server Trust Evaluation

如果您觉得我们的内容还不错，就请扫描转发到朋友圈，和小伙伴一起分享吧~