Http/Https是什么？

Http (HTTP-Hypertext transfer protocol) 是一个简单的请求-响应协议，它通常运行在TCP之上。它指定了客户端可能发送给服务器什么样的消息以及得到什么样的响应。请求和响应消息的头以ASCII码形式给出；而消息内容则具有一个类似MIME的格式。

Https（全称：Hyper Text Transfer Protocol over SecureSocket Layer），是以安全为目标的 HTTP 通道，在HTTP的基础上通过传输加密和身份认证保证了传输过程的安全性 。HTTPS 在HTTP 的基础下加入SSL 层，HTTPS 的安全基础是 SSL，因此加密的详细内容就需要 SSL。 HTTPS 存在不同于 HTTP 的默认端口及一个加密/身份验证层（在 HTTP与 TCP 之间）。这个系统提供了身份验证与加密通讯方法。

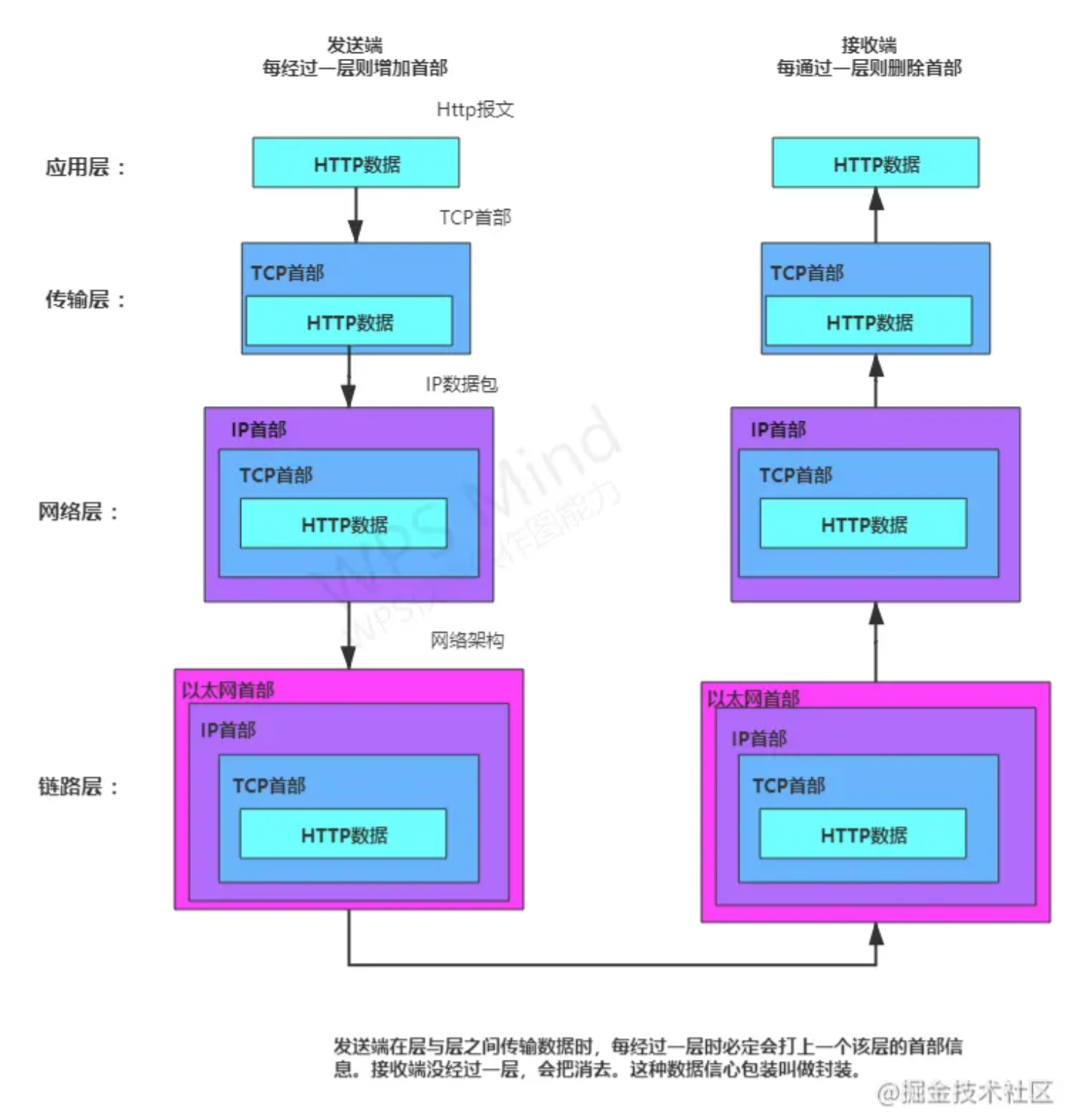
Http/Https请求流程

Http是超文本传输协议，数据明文传输，所以会被抓包导致信息泄露，有安全风险问题！Https 则是具有安全性的SSL加密传输协议。Http和Https使用的是完全不同的连接方式用的端口也不一样,前者是80,后者是443。Http的连接很简单,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。Https协议需要到CA付费申请证书,

HTTP是应用层协议，同其他应用层协议一样，是为了实现某一类具体应用的协议，并由某一运行在用户空间的应用程序来实现其功能。HTTP是一种协议规范，这种规范记录在文档上，为真正通过HTTP协议进行通信的HTTP的实现程序。

HTTP是一种无状态协议，即服务器不保留与客户交易时的任何状态。这就大大减轻了服务器记忆负担，从而保持较快的响应速度。

HTTP是一种面向对象的协议。允许传送任意类型的数据对象。它通过数据类型和长度来标识所传送的数据内容和大小，并允许对数据进行压缩传送。当用户在一个HTML文档中定义了一个超文本链后，浏览器将通过TCP/IP协议与指定的服务器建立连接。



上图是Http的工作流程图，Https会在Tcp首部这一层前加上安全层（SSL/TSL）加解密Http报文

Https解决什么问题？

HTTP也有以下明显缺点：

通信使用明文，内容可能被窃听

不验证通信方的身份，因此有可能遭遇伪装

无法证明报文的完整性，所以有可能遭到篡改

这样，HTTPS就登场了。HTTPS中的S表示SSL或者TLS，就是在原HTTP的基础上加上一层用于数据加密、解密、身份认证的安全层，即

HTTP + 加密 + 认证 + 完整性保护 = HTTPS

加密相关的预备知识：对称加密和非对称加密。

对称加密 ： 加密和解密数据使用同一个密钥。这种加密方式的特点是速度很快，常见对称加密的算法有 AES；

非对称加密： 加密和解密使用不同的密钥，这两个密钥形成有且仅有唯一的配对，叫公钥和私钥。数据用公钥加密后必须用私钥解密，数据用私钥加密后必须用公钥解密。一般来说私钥自己保留好，把公钥公开给别人（一般公钥不会单独出现，而是会写进证书中），让别人拿自己的公钥加密数据后发给自己，这样只有自己才能解密。 这种加密方式的特点是速度慢，CPU 开销大，常见非对称加密算法有 RSA。

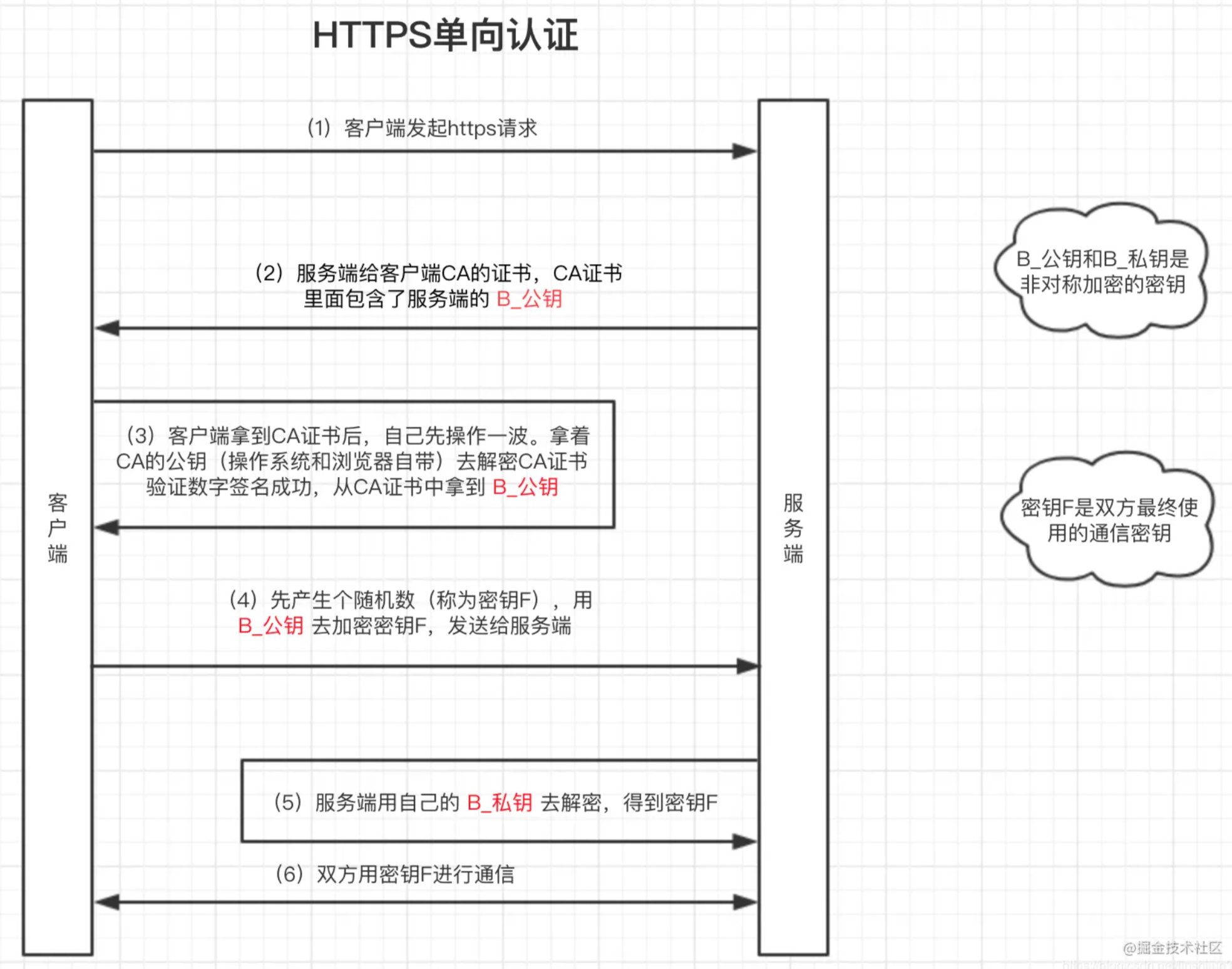
CA证书的相关知识：

CA证书是由CA（Certification Authority）机构发布的数字证书。其内容包含：电子签证机关的信息、公钥用户信息、公钥、签名和有效期。这里的公钥服务端的公钥，这里的签名是指：用hash散列函数计算公开的明文信息的信息摘要，然后采用CA的私钥对信息摘要进行加密，加密完的密文就是签名。

即：证书 = 公钥 + 签名 +申请者和颁发者的信息。

客户端中因为在操作系统中就预置了CA的公钥，所以支持解密签名（因为签名使用CA的私钥加密的）

Https单向验证



从上图可以看出，服务端拥有一对非对称密钥：B\_公钥和B\_私钥。详细过程如下：

（1）客户端发起HTTPS请求，将SSL协议版本的信息发送给服务端。

（2）服务端去CA机构申请来一份CA证书，在前面提过，证书里面有服务端公钥和签名。将CA证书发送给客户端

（3）客户端读取CA证书的明文信息，采用相同的hash散列函数计算得到信息摘要（hash目的：验证防止内容被修改），然后用操作系统带的CA的公钥去解密签名（因为签名是用CA的私钥加密的），对比证书中的信息摘要。如果一致，则证明证书是可信的，然后取出了服务端公钥

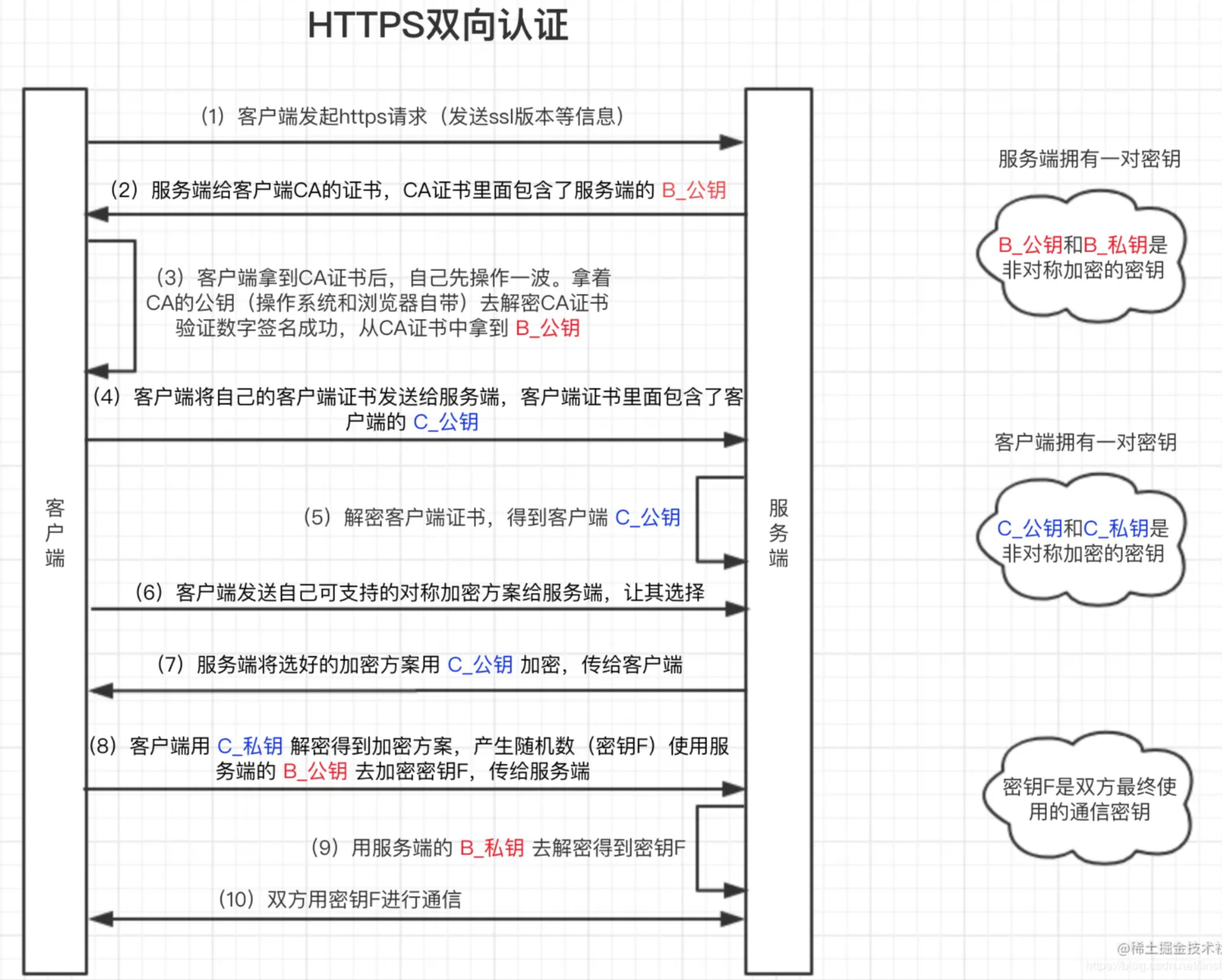
（4）客户端生成一个随机数（密钥F），用刚才等到的服务端B\_公钥去加密这个随机数形成密文，发送给服务端。

（5）服务端用自己的B\_私钥去解密这个密文，得到了密钥F

（6）服务端和客户端在后续通讯过程中就使用这个密钥F进行通信了。和之前的非对称加密不同，这里开始就是一种对称加密的方式

Https双向验证

双向认证和单向认证原理基本差不多，单向认证客户端需要认证服务端，而在双向认证中增加了服务端对客户端的认证



双向认证详细过程如下：

（1）客户端发起HTTPS请求，将SSL协议版本的信息发送给服务端。

（2）服务端去CA机构申请来一份CA证书，在前面提过，证书里面有服务端公钥和签名。将CA证书发送给客户端

（3）客户端读取CA证书的明文信息，采用相同的hash散列函数计算得到信息摘要（hash目的：验证防止内容被修改），然后用操作系统带的CA的公钥去解密签名（因为签名是用CA的私钥加密的），对比证书中的信息摘要。如果一致，则证明证书是可信的，然后取出了服务端公钥

（4）客户端发送自己的客户端证书给服务端，证书里面有客户端的公钥：C\_公钥

（5）客户端发送支持的对称加密方案给服务端，供其选择

（6）服务端选择完加密方案后，用刚才得到的C\_公钥去加密选好的加密方案

（7）客户端用自己的C\_私钥去解密选好的加密方案，客户端生成一个随机数（密钥F），用刚才等到的服务端B\_公钥去加密这个随机数形成密文，发送给服务端。

（8）服务端和客户端在后续通讯过程中就使用这个密钥F进行通信了。和之前的非对称加密不同，这里开始就是一种对称加密的方式

ps：双向认证的客户端证书一般都可以是如openssl生成的自签名证书，包括 client.crt 和 client.key，这两部分内容可以集成在 p12 证书中, p12 证书可以设置打开密码。

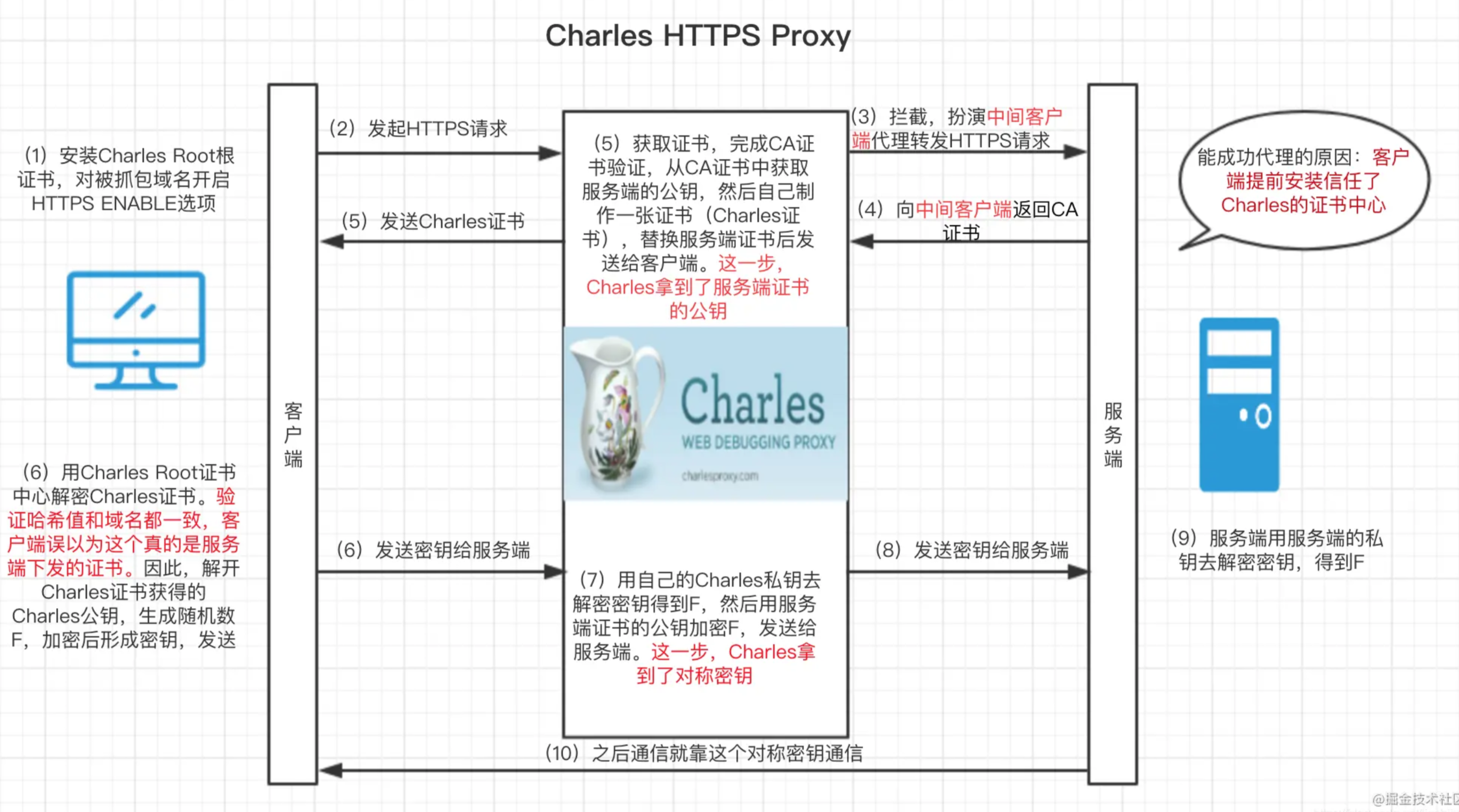
抓包原理

常用的HTTPS抓包方式是作为中间人，对客户端伪装成服务端，对服务端伪装成客户端。简单来说：

截获客户端的HTTPS请求，伪装成中间人客户端去向服务端发送HTTPS请求

接受服务端返回，用自己的证书伪装成中间人服务端向客户端发送数据内容。

具体过程如下图所示：



反抓包策略

为了防止中间人攻击，可以使用SSL-Pinning的技术来反抓包。

可以发现中间人攻击的要点的伪造了一个假的服务端证书给了客户端，客户端误以为真。解决思路就是，客户端也预置一份服务端的证书，比较一下就知道真假了。

SSL-pinning有两种方式： 证书锁定（Certificate Pinning） 和公钥锁定（ Public Key Pinning）。

证书锁定

需要在客户端代码内置仅接受指定域名的证书，而不接受操作系统或浏览器内置的CA根证书对应的任何证书，通过这种授权方式，保障了APP与服务端通信的唯一性和安全性，因此客户端与服务端（例如API网关）之间的通信是可以保证绝对安全。但是CA签发证书都存在有效期问题，缺点是在 证书续期后需要将证书重新内置到APP中。

公钥锁定

提取证书中的公钥并内置到客户端中，通过与服务器对比公钥值来验证连接的正确性。制作证书密钥时，公钥在证书的续期前后都可以保持不变（即密钥对不变），所以可以避免证书有效期问题，一般推荐这种做法。

突破反抓包策略

在逆向界，一山更比一山高。

思路是这样的：内置证书或者公钥的时候，常常会有对比验证的函数，直接控制这个函数的返回结果让验证通过不就好了吗。

于是就有了一个突破SLL-Pinning的经典操作：采用Xposed+justTrustme模块。

这个方案使用的是JustTrustMe这个Xposed模块，它所做的事情就是将各种已知的的HTTP请求库中用于校验证书的API都进行Hook，使无论是否是可信证书的情况，校验结果返回都为正常状态，从而实现绕过证书检查的效果。