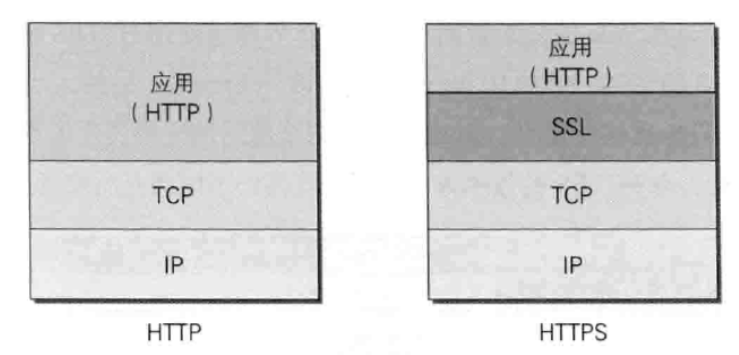
**为何要用https?**

**http协议的缺点**

* 通信使用明文，内容可能被窃听(重要密码泄露)
* 不验证通信方身份，有可能遭遇伪装(跨站点请求伪造)
* 无法证明报文的完整性，有可能已遭篡改(运营商劫持)

**用https能解决这些问题么？**

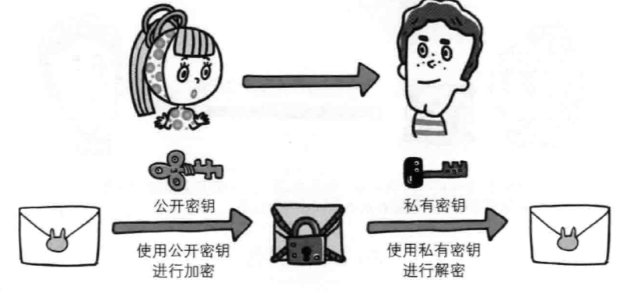
https是在http协议基础上加入加密处理和认证机制以及完整性保护，即http+加密+认证+完整性保护=https  
https并非应用层的一种新协议，只是http通信接口部分用ssl/tls协议代替而已。通常http直接和tcp通信，当使用ssl时则演变成先和ssl通信，再由ssl和tcp通信。  
所谓https，其实就是身披ssl协议这层外壳的http  


**SSL/TLS是什么？**

SSL 是“Secure Sockets Layer”的缩写，中文叫做“安全套接层”。它是在上世纪90年代中期，由网景公司设计的。  
为啥要发明 SSL 这个协议？因为原先互联网上使用的 HTTP 协议是明文的，存在很多缺点——比如传输内容会被偷窥（嗅探）和篡改。发明 SSL 协议，就是为了解决这些问题。  
到了1999年，SSL 因为应用广泛，已经成为互联网上的事实标准。IETF 就在那年把 SSL 标准化。标准化之后的名称改为 TLS（是“Transport Layer Security”的缩写），中文叫做“传输层安全协议”。  
所以这两者其实就是同一种协议，只不过是在不同阶段的不同称呼。

SSL协议位于TCP/IP协议与各种应用层协议之间，为数据通讯提供安全支持。SSL协议可分为两层：  
SSL记录协议（SSL Record Protocol）：它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。  
SSL握手协议（SSL Handshake Protocol）：它建立在SSL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。

**对称秘钥加密和非对称秘钥加密**

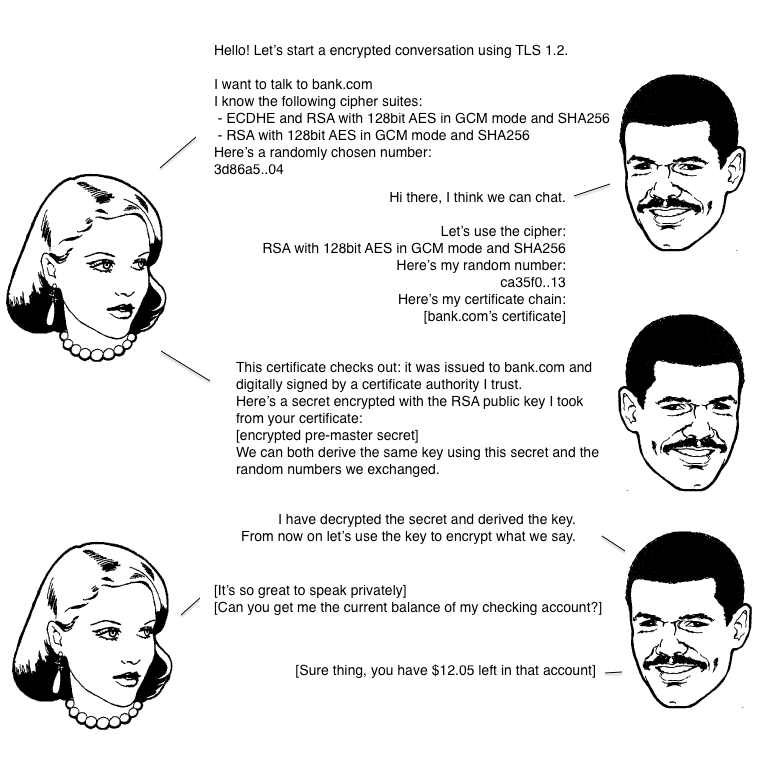
对称密钥加密，又称私钥加密，即信息的发送方和接收方用同一个密钥去加密和解密数据。它的最大优势是加/解密速度快，适合于对大数据量进行加密，但密钥管理困难。  
非对称密钥加密，又称公钥加密，它需要使用一对密钥来分别完成加密和解密操作，一个公开发布，即公开密钥，另一个由用户自己秘密保存，即私用密钥。信息发送者用公开密钥去加密，而信息接收者则用私用密钥去解密。  
从功能角度而言非对称加密比对称加密功能强大，但加密和解密速度却比对称密钥加密慢得多。  
非对称密钥通信过程  


**SSL/TLS协议基本原理**

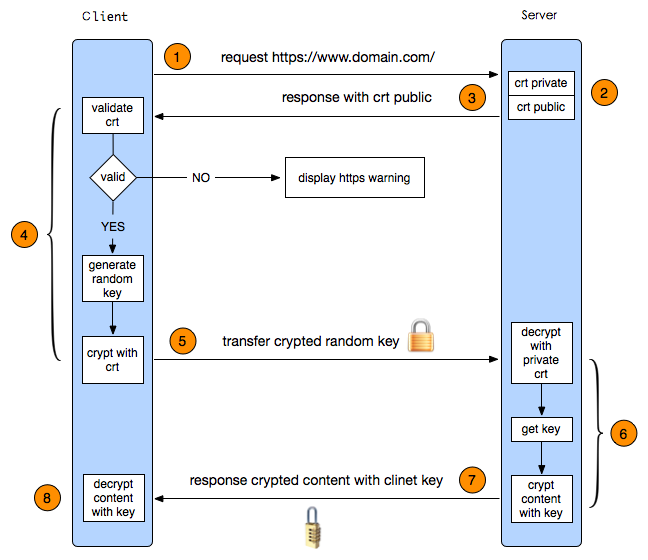
SSL/TLS协议的基本思路是采用公钥加密法，也就是说，客户端先向服务器端索要公钥，然后用公钥加密信息，服务器收到密文后，用自己的私钥解密，但是这里有两个问题：  
（1）、如何保证公钥不被篡改？  
解决方法：将公钥放在数字证书中，只要证书是可信的，公钥就是可信的。  
（2）、公钥加密计算量太大，如何减少耗用的时间？  
解决方法：每一次对话（session），客户端和服务器端都生成一个"对话密钥"（session key），用它来加密信息。由于"对话密钥"是对称加密，所以运算速度非常快，而服务器公钥只用于加密"对话密钥"本身，这样就减少了加密运算的消耗时间。

因此，SSL/TLS协议的基本过程是这样的：

1. 客户端向服务器端索要并验证公钥。
2. 双方协商生成“对话密钥”。
3. 双方采用“对话密钥”进行加密通信。

具体过程可参考下面的栗子  
假定客户端叫做爱丽丝，服务器叫做鲍勃，整个握手过程可以用下图说明  
  
第一步，爱丽丝给出协议版本号、一个客户端生成的随机数（Client random），以及客户端支持的加密方法，具体的加密方法可参考[SSL证书背后的加密算法](https://www.cnblogs.com/cioliuguilan/p/5518798.html)。  
第二步，鲍勃确认双方使用的加密方法，并给出数字证书、以及一个服务器生成的随机数（Server random）。  
第三步，爱丽丝确认数字证书有效，然后生成一个新的随机数（Premaster secret），并使用数字证书中的公钥，加密这个随机数，发给鲍勃。  
第四步，鲍勃使用自己的私钥，获取爱丽丝发来的随机数（即Premaster secret）。  
第五步，爱丽丝和鲍勃根据约定的加密方法，使用前面的三个随机数，生成"对话密钥"（session key），用来加密接下来的整个对话过程。

**HTTPS工作原理**

1、客户端发起HTTPS请求  
用户在浏览器里输入一个https网址，然后连接到server的443端口。  
2、服务端的配置  
采用HTTPS协议的服务器必须要有一套数字证书，可以自己制作，也可以向组织申请，区别就是自己颁发的证书需要客户端验证通过，才可以继续访问，而使用受信任的公司申请的证书则不会弹出提示页面。  
3、传送证书  
这个证书其实就是公钥，只是包含了很多信息，如证书的颁发机构，过期时间等等。  
4、客户端解析证书  
这部分工作是由客户端的TLS来完成的，首先会验证公钥是否有效，比如颁发机构，过期时间等等，如果发现异常，则会弹出一个警告框，提示证书存在问题。  
（1）首先浏览器读取证书中的证书所有者、有效期等信息进行一一校验  
（2）浏览器开始查找操作系统中已内置的受信任的证书发布机构CA，与服务器发来的证书中的颁发者CA比对，用于校验证书是否为合法机构颁发  
（3）如果找不到，浏览器就会报错，说明服务器发来的证书是不可信任的。  
（4）如果找到，那么浏览器就会从操作系统中取出颁发者CA 的公钥(多数浏览器开发商发布  
版本时，会事先在内部植入常用认证机关的公开密钥)，然后对服务器发来的证书里面的签名进行解密  
（5）浏览器使用相同的hash算法计算出服务器发来的证书的hash值，将这个计算的hash值与证书中签名做对比  
（6）对比结果一致，则证明服务器发来的证书合法，没有被冒充  
（7）此时浏览器就可以读取证书中的公钥，用于后续加密了  
5、传送加密信息  
这部分传送的是用证书加密后的随机值(私钥)，目的就是让服务端得到这个随机值，以后客户端和服务端的通信就可以通过这个随机值来进行加密解密了。  
6、服务端解密信息  
服务端用私钥解密后，得到了客户端传过来的随机值(私钥)，然后把内容通过该值进行对称加密  
7、传输加密后的信息  
这部分信息是服务端用私钥加密后的信息，可以在客户端被还原。  
8、客户端解密信息  
客户端用之前生成的私钥解密服务端传过来的信息，于是获取了解密后的内容，整个过程第三方即使监听到了数据，也束手无策。  
**原理如下图**  


**总结HTTP和HTTPS区别如下**

1. https协议需要到ca申请证书，一般免费证书较少，因而需要一定费用。
2. http是超文本传输协议，信息是明文传输，https则是具有安全性的ssl加密传输协议。
3. 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。
4. 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，比http协议安全。

**如何配置https**

配置https最重要的是配置ssl证书，配置SSL证书可以参考[SSL证书部署指南](https://freessl.wosign.com/guide)  
这里我们以自签证书来演示

**生成自签证书**

生成私钥文件  
sudo openssl genrsa -out server.key 2048  
生成自签证书文件  
sudo openssl req -new -x509 -days 1826 -key server.key -out server.crt

**修改apache配置**

apache2.4  
需开启的模块  
LoadModule ssl\_module libexec/apache2/mod\_ssl.so  
LoadModule socache\_shmcb\_module libexec/apache2/mod\_socache\_shmcb.so

sudo vim httpd-ssl.conf

<VirtualHost \*:443>

DocumentRoot "/var/www/html"

ServerName www.domain.com:443

SSLEngine on

SSLCertificateFile /usr/local/apache/conf/server.crt #添加证书文件

SSLCertificateKeyFile /usr/local/apache/conf/server.key #添加私钥文件

</VirtualHost>

检测配置文件是否有错误  
sudo apachectl configtest  
重启apache  
sudo apachectl restart

**Fiddler/Charles是如何监听HTTPS?**

第一步，Fiddler截获客户端发送给服务器的HTTPS请求，Fiddler伪装成客户端向服务器发送请求进行握手 。  
第二步，服务器发回相应，Fiddler获取到服务器的CA证书， 用根证书（这里的根证书是CA认证中心给自己颁发的证书）公钥进行解密， 验证服务器数据签名， 获取到服务器CA证书公钥。然后Fiddler伪造自己的CA证书（这里的CA证书，也是根证书，只不过是Fiddler伪造的根证书）， 冒充服务器证书传递给客户端浏览器。  
第三步，与普通过程中客户端的操作相同，客户端根据返回的数据进行证书校验、生成密码Pre\_master、用Fiddler伪造的证书公钥加密，并生成HTTPS通信用的对称密钥enc\_key。  
第四步，客户端将重要信息传递给服务器， 又被Fiddler截获。Fiddler将截获的密文用自己伪造证书的私钥解开， 获得并计算得到HTTPS通信用的对称密钥enc\_key。Fiddler将对称密钥用服务器证书公钥加密传递给服务器。  
第五步，与普通过程中服务器端的操作相同，服务器用私钥解开后建立信任，然后再发送加密的握手消息给客户端。  
第六步，Fiddler截获服务器发送的密文， 用对称密钥解开， 再用自己伪造证书的私钥加密传给客户端。  
第七步，客户端拿到加密信息后，用公钥解开，验证HASH。握手过程正式完成，客户端与服务器端就这样建立了”信任“。

在之后的正常加密通信过程中，Fiddler如何在服务器与客户端之间充当第三者呢？  
服务器—>客户端：Fiddler接收到服务器发送的密文， 用对称密钥解开， 获得服务器发送的明文。再次加密， 发送给客户端。  
客户端—>服务端：客户端用对称密钥加密，被Fiddler截获后，解密获得明文。再次加密，发送给服务器端。由于Fiddler一直拥有通信用对称密钥enc\_key， 所以在整个HTTPS通信过程中信息对其透明。  
