**HTTP VS HTTPS （为何使用HTTPS协议？）**

HTTP协议

使用明文传输【80端口】

导致\*中间人攻击[[1]](#footnote-1)（MITM）：{

1. 广告链接嵌入服务器发给用户的http报文
2. 修改用户的请求头URL

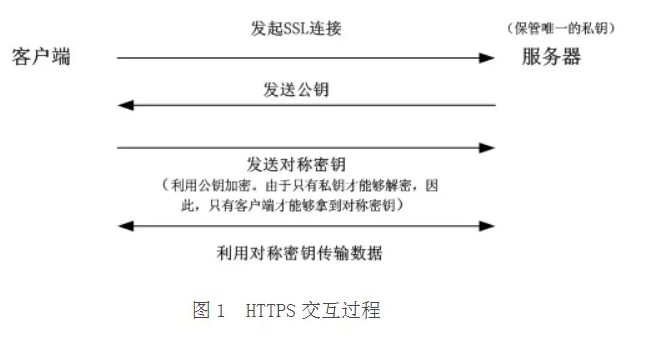
}

HTTPS协议

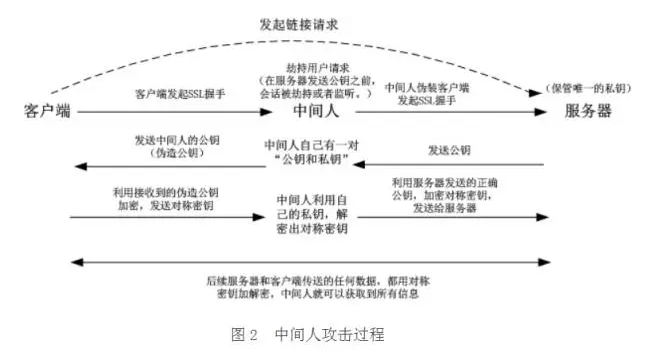
加入SSL[[2]](#footnote-2)（secure sockets layer）加密传输【443端口】

**SSL安全性**

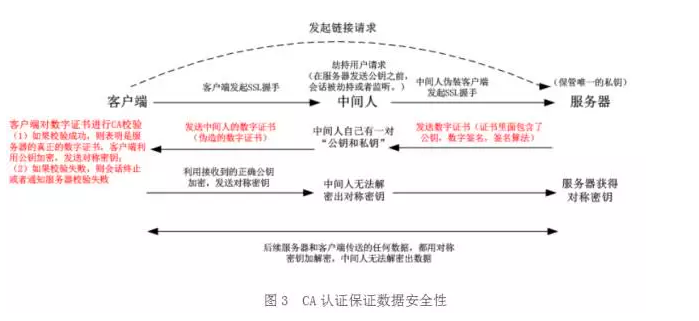
1.非对称加解密

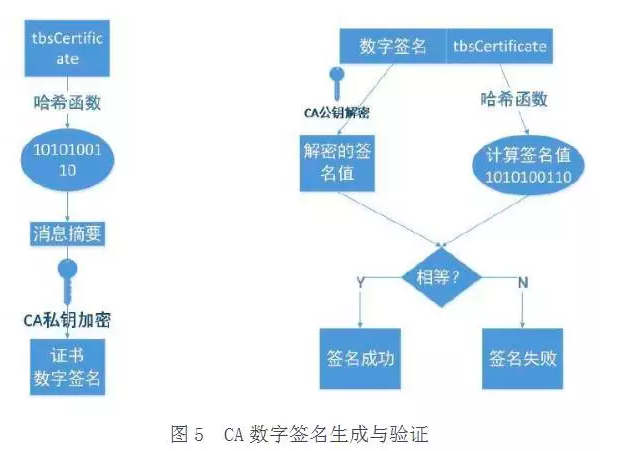


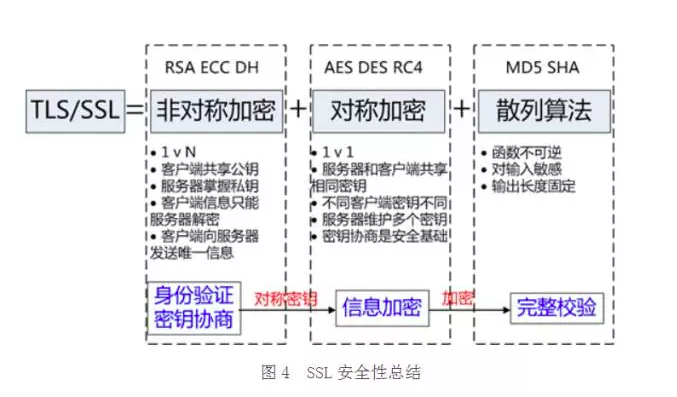
潜在问题：中间人伪装“虚假服务器”与“虚假客户端”通信



2.数字证书[[3]](#footnote-3)（CA）：服务器首先生成公私钥，将公钥提供给相关机构（CA），CA将公钥放入数字证书并将数字证书颁布给服务器，此时服务器就不是简单的把公钥给客户端，而是给客户端一个数字证书，数字证书中加入了一些数字签名的机制，保证了数字证书一定是服务器给客户端的。中间人发送的伪造证书，不能够获得CA的认证。

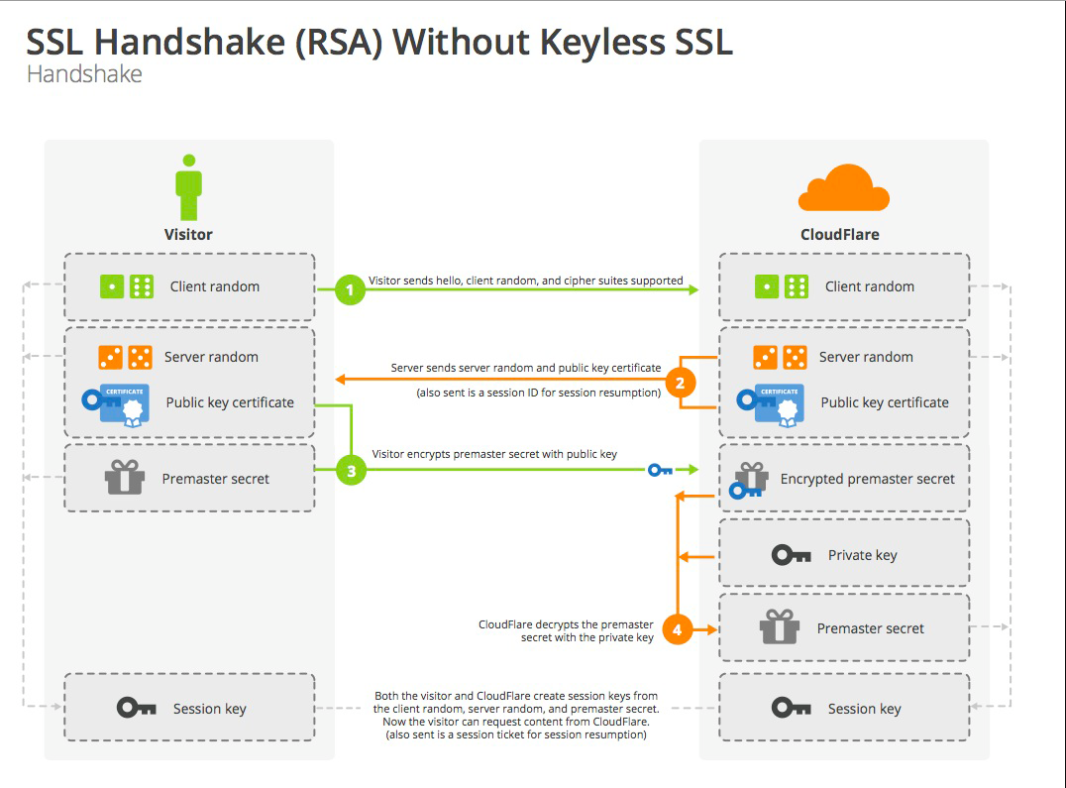


CA数字签名实现：



TLS（transport layer security）TLS以SSL 3.0[[4]](#footnote-4)为基础于1999年作为SSL的新版本推出。

TLS/SSL 握手示意图：



客户端——协议版本号+客户端随机数+客户端支持的加密方法

服务端——数字证书+服务端随机数——客户端

客户端确认CA——新的随机数【使用CA中的公钥加密】——服务端

服务端——服务端私钥——获取客户端随机数

客户端+服务端——使用约定加密方法+上述三个随机数——对话密钥【加密之后对话全过程】

**使用HTTPS有何风险/问题？**

知乎案例：

黑客如何嗅探HTTPS？

采用SSL层剥离技术，黑客阻止用户和使用HTTPS请求的网站之间建立SSL连接，使用户和代理服务器（黑客所控制的服务器）之间采用未加密的HTTP通信。

使用工具SSLStrip

漏洞：ZynOS路由器，导致大批量DNS劫持

极少数情况出现，且是针对路由器web管理器被黑，对HTTPS/SSL本身是否安全无关

潜在风险：

1.HTTPS可以保证传输过程中的信息不被别人截获，但HTTPS是应用层协议，下层采用SSL保证信息安全，但是在客户端和服务端，密文同样是可以被截获的。

2.HTTPS报文在传输过程中，如果客户端被恶意引导安装“中间人”的WEB信任证书，那么HTTPS中的“[中间人攻击](https://www.cnblogs.com/gordon0918/p/5237717.html)”一样会将明文密码泄露给别人。

潜在问题：

1.使用TLS会导致客户端和服务端的性能下降。这些性能瓶颈在IoT设备，RFID[[5]](#footnote-5)设备上较为明显。

2.无法完全信任服务器。

**\*针对HTTPS的web前端劫持**

**（参考<http://netinfo-security.org/article/2016/1671-1122-0-3-15.html>）**

用户通过浏览器访问支持HTTPS协议的网站时有两种不同的访问方式, 分别是HTTPS协议超链接和HTTP超链接重定向, 其详细介绍如下.

1 HTTPS协议超链接

单击类似 https://mail.qq.com 超链接时,首先客户端向DNS服务器发起DNS请求, 收到DNS服务器的DNS响应报文获得所访问的Web服务器的IP地址; 然后与支持 HTTPS 的服务器建立 TCP (服务器监听端口为443)连接, 接着与服务器建立 SSL 安全通道; 最后传输加密数据.针对这种情况, 只能采取基于伪造证书的中间人劫持的方式获取HTTP明文数据

2 HTTP超链接重定向

在导航网页单击类似http://mail.qq.com超链接时，首先客户端向本地DNS服务器发起 DNS请求, 收到服务器的DNS响应报文获得所访问的Web服务器的IP地址; 接着与Web服务器建立 TCP 连接(服务器监听端口为80), 发起明文的 HTTP 请求, 由于该服务器是只支持HTTPS的, 服务器需要将该服务重定向到443端口(HTTPS协议使用443端口).于是服务器响应302重定向报文, 该报文的重定向 Location字段是以https开头的URL, 客户端重新发起DNS请求, 与服务器的443端口建立TCP 连接, 建立SSL安全通道, 传输加密数据.

针对超链接重定向, 可以采取两种不同劫持方式:1)伪造数字证书劫持; 2)重定向会话劫持.。

**\*HTTPS后端劫持及弊端分析**

伪造数字证书劫持

当用户直接输入以HTTPS开头的网址时, 数据通信过程是加密的, 只能通过伪造数字证书的方法获取明文数据. 根据浏览器验证证书的缺陷, 劫持过程如下：

1)中间人监听客户端TCP连接请求, 当收到客户端请求后中间人伪造客户端向真实服务器发起TCP连接.

2)中间人监听到客户端发起的SSL连接请求Client\_Hello时, 记录相关信息, 并转发该数据包以连接服务器, 实现服务器密钥交换和鉴别; 监听到服务器SSL握手响应, 记录服务器下发的证书相关信息, 并根据信息伪造自签名数字证书下发给客户端.

3)浏览器验证证书合法性, 证书不合法则弹出安全警告框, 提醒用户选择关闭或继续浏览; 若用户选择继续浏览, 则中间人与客户端之间的SSL连接成功, 从而实现对HTTPS协议通信的会话劫持.

重定向会话劫持

当用户直接在浏览器输入非HTTPS开头的URL或者直接点击导航页面HTTP超链接时, 我们可以在后端截获重定向报文, 将HTTPS协议降级为HTTP获取明文数据流.

1) 中间人监听80端口, 若有客户端发起TCP连接(服务器监听端口为80), 伪造客户端与真实服务器建立TCP连接.

2)客户端发起明文的GET请求, 中间人将请求转发至服务器.

3)中间人截获服务器的重定向响应报文, 与服务器建立SSL安全通道, 并发起同样GET请求.

4)中间人获得服务器响应内容, 解析相应报文内容, 若发现有https字段, 则替换成http, 并添加标识符表示原来是https安全请求URL.例如, 发现< a href=https://passport.jd.com> , 则替换成< a href=http://passport.jd.com?> , 其中"?"是添加的标志.

5)把修改后的响应报文以HTTP明文形式转发至客户端, 实现客户端与中间人之间的HTTPS通信降级, 最终实现会话劫持.SSLStrip工具就是一个典型案例.

**\*HTTPS前端劫持防御措施**

Web前端劫持首先是通过中间人劫持的方式注入JS脚本, 然后通过JS脚本实现相关功能.因此, 我们可以从前端和后端两方面进行防御.前端防御可疑JS脚本, 后端防御中间人劫持.本文针对前端提出两种防御方法, 后端用过HSTS策略防御中间人劫持.

1)安全脚本探针

其实, 无论是从后端还是前端着手, 仍然有很多网站无法降级.例如, QQ 邮箱登录页面在网页中植入安全探针脚本[7, 8, 9], 需匹配location的协议类型, 若是HTTP协议就修改cookie, 并把URL替换成HTTPS, 这样攻击者很难成功劫持, 具体代码如下:

if(location.protocol=="http:")

{

document.cookie = "edition=; expires=-1; path=/; domain=.mail.qq.com";

location.href="https://mail.qq.com";

}

2)混淆明文

京东登录页面的脚本里, URL是以明文形式出现的, 我们可以利用前后端结合的方法, 对脚本里的https://文本进行替换.但如果我们的URL是通过字符串拼接的, HTTPS协议降级就难以成功.所以在信息传输安全要求较高的时候, 必须对重要网站的地址进行处理, 这样中间人就无法采用常规方式实现劫持, 攻击难度大大提升.

3)HSTS策略

HSTS(HTTP Strict Transport Security)是国际IE组织推行的一种新的Web安全协议.使用HSTS协议的网站, 只要这个头部出现过一次, 浏览器在设定时间里只会以HTTPS 与服务器建立连接.因此, 我们的网站应尽可能开启HSTS功能.事实上流量劫持并非都是100%成功, 如前述提到的使用脚本跳转就很容易出现遗漏.HSTS可以让之后的页面降级彻底失效.对于像超链接这种不可避免的重定向攻击, 建议服务器开启HSHS 功能.如果用户已经用HTTPS访问过站点(支持HSTS功能的站点会在响应头中插入:Strict-Transport-Security), 支持HSTS的浏览器(如Chrome, Firefox)会将这个域名添加到HSTS列表, 下次用户访问这个网站, 浏览器只以HTTPS协议发起请求(前提是用户没有清空缓存), 而不是先发送HTTP, 然后重定向到HTTPS, 从而成功避免302重定向URL被篡改.

**前端加密**

方法：

* JavaScript 加密后传输{

哈希加密【由于哈希算法的不可逆性，中间人无法从截取的数据中得知用户的密码信息。但是这里有一个问题，攻击者仍然可以使用拿到的哈希值进行直接登录。使用前端加密如何避免中间人重放攻击？——使用动态salt】

结合验证码加密【前端先将密码哈希，然后和用户输入的验证码进行哈希，得到的结果作为密码字段发送给服务器。服务器先确认验证码正确，然后再进行密码验证，否则直接返回验证码错误信息。】

}

* 浏览器插件内进行加密传输
* Https 传输

**使用HTTPS协议是否还要前端加密？——否**

* HTTPS协议本身的可靠性
* 有效防范中间人攻击。
* 即使被截获，也是密文信息。【极端情况：服务器被黑】
* 本地cookie泄露，使用HSTS平台[[6]](#footnote-6)，对于特定域名强制进行HTTPS访问
* 其余前端加密方式有效性差。例如js加密：

它可以生成加密密钥，尽管长度小于48位的熵，因此它的输出不会超过48位的熵，即使它的种子超过了48位。”

SecureRandom（）运行的数字，也是通过早已过时的RC4算法得到的，所以密钥其实是可预测、也更容易预测的，因为它们熵的位数较少。

1. 在[密码学](https://zh.wikipedia.org/wiki/%E5%AF%86%E7%A0%81%E5%AD%A6)和[计算机安全](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E5%AE%89%E5%85%A8)领域中，是指攻击者与通讯的两端分别创建独立的联系，并交换其所收到的数据，使通讯的两端认为他们正在通过一个私密的连接与对方直接对话，但事实上整个会话都被攻击者完全控制。 [↑](#footnote-ref-1)
2. 此举可确保在用户和站点之间，或两个系统之间传输的数据无法被读取。它使用加密算法打乱传输中的数据，防止数据通过连接传输时被黑客读取。这里所说的数据是指任何敏感或个人信息，例如信用卡号和其他财务信息、个人姓名和住址等。 [↑](#footnote-ref-2)
3. 身份授权、分发公钥、验证证书合法性 [↑](#footnote-ref-3)
4. SSL2.0和SSL3.0已经被IEFT组织废弃（分别在2011年，2015年）。多年来，在被废弃的SSL协议中一直存在漏洞并被发现 (e.g. [POODLE](https://link.juejin.im?target=https%3A%2F%2Fwww.globalsign.com%2Fen%2Fblog%2Fpoodle-vulnerability-in-ssl-30%2F), [DROWN](https://link.juejin.im?target=https%3A%2F%2Fwww.globalsign.com%2Fen%2Fblog%2Fdrown-attack-sslv2%2F))。大多数现代浏览器遇到使用废弃协议的web服务时，会降低用户体验（红线穿过挂锁标志或者https表示警告）来表现。因为这些原因，你应该在服务端禁止使用SSL协议，仅仅保留TLS协议开启。 [↑](#footnote-ref-4)
5. 射频识别，RFID（Radio Frequency Identification）技术，又称无线射频识别，是一种通信技术，可通过无线电讯号识别特定目标并读写相关数据，而无需识别系统与特定目标之间建立机械或光学接触。 [↑](#footnote-ref-5)
6. HTTP严格传输安全（英语：HTTP Strict Transport Security，[缩写](https://zh.wikipedia.org/wiki/%E7%B8%AE%E5%AF%AB)：HSTS）是一套由[互联网工程任务组](https://zh.wikipedia.org/wiki/%E4%BA%92%E8%81%94%E7%BD%91%E5%B7%A5%E7%A8%8B%E4%BB%BB%E5%8A%A1%E7%BB%84)发布的[互联网](https://zh.wikipedia.org/wiki/%E4%BA%92%E8%81%94%E7%BD%91)安全策略机制。[网站](https://zh.wikipedia.org/wiki/%E7%BD%91%E7%AB%99)可以选择使用HSTS策略，来让[浏览器](https://zh.wikipedia.org/wiki/%E6%B5%8F%E8%A7%88%E5%99%A8)强制使用[HTTPS](https://zh.wikipedia.org/wiki/%E8%B6%85%E6%96%87%E6%9C%AC%E4%BC%A0%E8%BE%93%E5%AE%89%E5%85%A8%E5%8D%8F%E8%AE%AE)与网站进行通信，以减少[会话劫持](https://zh.wikipedia.org/wiki/%E4%BC%9A%E8%AF%9D%E5%8A%AB%E6%8C%81)风险。 [↑](#footnote-ref-6)