极客HTTP协议学习笔记安全篇（23-29）

原创

吃货的自我修养 于 2021-07-23 21:03:27 发布 45 收藏

分类专栏： HTTP协议 文章标签： http

本文链接：https://blog.csdn.net/CSDNLYFc/article/details/118941387

23、HTTPS，SSL/TLS

- HTTPS的来由：HTTP协议明文传输，并不安全，所以后来有了HTTPS
  如何才算安全呢
  通常认为通信过程具备四个特性就可以认为是安全的，四个特性如下：

1. 机密性：指对数据保密，只能由可信的人访问，对其他人是不可见的“秘密”
2. 数据完整性：数据在传输过程中没有被篡改
3. 身份认证：指确认对方的身份，保证信息只能发送可信的人
4. 不可否认：不能否认已经发生过的行为

- RFC文档规定新协议名叫“https”，默认端口号443，让HTTP运行在安全的ssl/tls协议上，收发报文不再使用Socket API,而是调用专门的安全接口
- SSL 安全套接层，在OSI模型中处于第五层会话层，网景公司1994年发明，有v2,v3两个版本，v1因为有严重缺陷从未公开过，SSL发展到v3已经证明它自身是一个非常好的安全通信协议，于是1999年把它改名为TLS（传输层安全），正式标准化，版本从1.0重新算起，所以TLS1.0实际上就是SSL v3.1
- TLS 发展到现在有4个版本，分别是1999年初版TLS1.0,2006年 TLS 1.1,2008年TLS 1.2，2018年TLS1.3，每个新版本都紧跟密码学的发展和互联网的现状，持续强化安全和性能，已经成为了信息安全领域中的权威标准。现在应用最广泛的是TLS1.2
- TLS由记录协议，握手协议，警告协议，变更密码规范协议，扩展协议等几个子协议组成，综合使用了对称加密，非对称加密，身份认证等许多密码学前沿技术。
- 浏览器和服务器在使用TLS建立连接时需选择一组恰当的加密算法来实现安全通信，这些算法的组合被称为“密码套间”
- 密码套件格式：秘钥交换算法+签名算法+对称加密算法+摘要算法
  OpenSSL
  它是一个著名的开源密码学程序库和工具包，几乎支持所有公开的加密算法和协议，已经成为事实上的标准，许多应用软件都会使用其作为底层库来实现TLS功能，目前有3个主要分支，1.0.2版本，1.1.0版本，都在2019年不再维护，最新长期支持的版本是1.1.1

24、对称加密与非对称加密

对称加密 加解密的秘钥是同一个，TLS提供了RC4，DES，3DES，AES，ChaCha30等，但前三种算法被认为是不安全的，通常都禁止使用，目前常用的AES和ChaCha20

AES（Advanced Encryption Standard） 即高级加密标准，秘钥长度可以是128,192,256，是DES算法的替代者，安全强度高，性能好，有硬件还会做特殊优化，所以非常流行，是应用最广泛的对称加密算法。

加密分组模式 可以让算法用固定长度的秘钥 加密任意长度的明文，最早有ECB,CBC,CFB,OFB等几种分组模式，但都陆续被发现有安全漏洞，所以现在都不怎么用了，最新的分组模式被称为AEAD(Authenticated Encryption with Associated Data)，在加密的同时增加了认证的功能，常用的是GCM,CCM和Poly1305.

非对称加密

秘钥交换 把秘钥安全的传给对方

特点 它有两个秘钥，一个是公钥，一个是私钥，公钥可以公开给任何人加密，而私钥必须严格保密。加解密具有单向性，公钥私钥虽然都可以用来加解密，但是公钥加密后只能用私钥解密，私钥加密后也只能用公钥解密

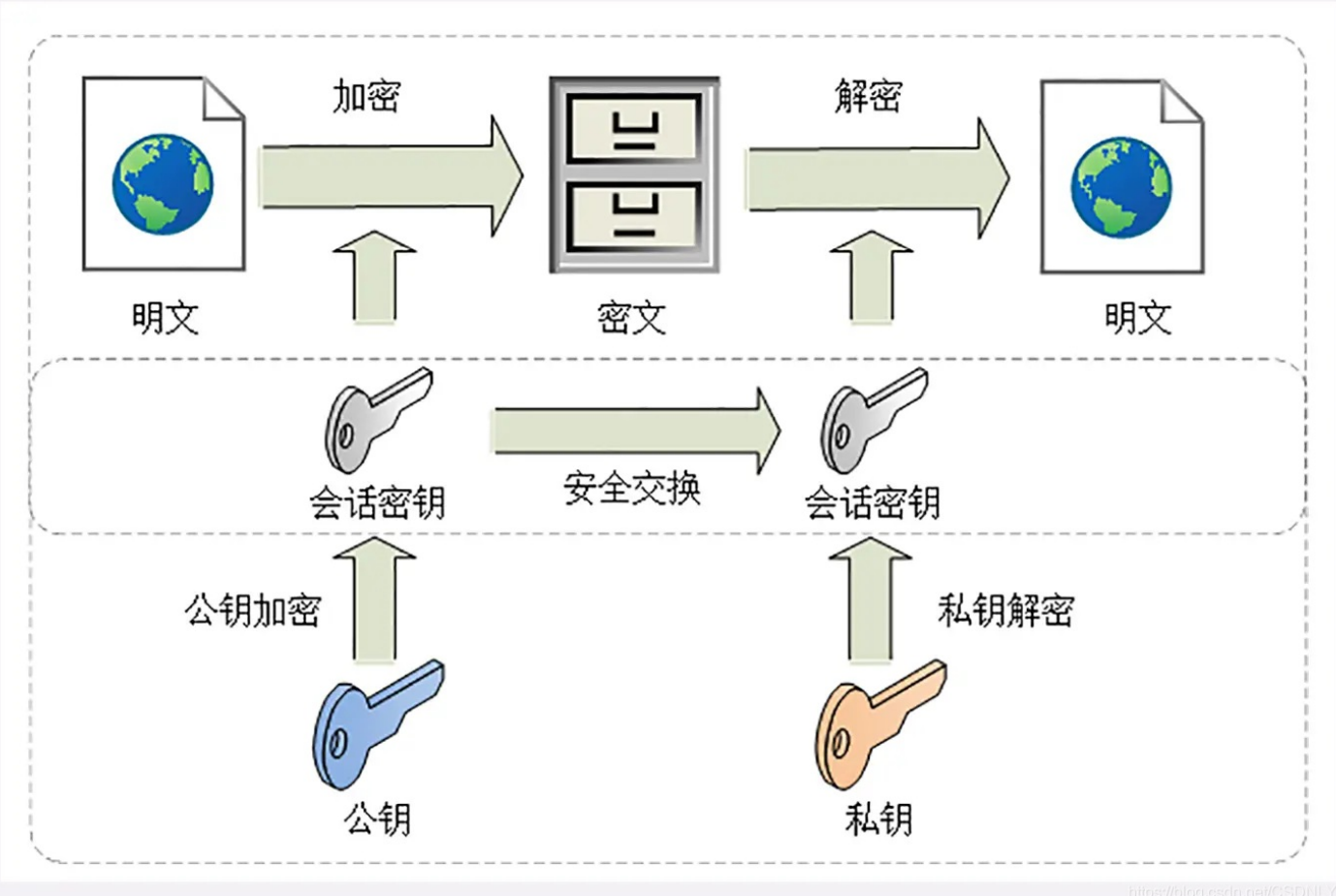
非对称加密算法 在TLS里只有很少的几种，例如 DH，DSA，RSA，ECC等。

RSA 10年前RSA的推荐秘钥长度为1024，现在随着计算机运算能力的提高，普遍认为至少需要2048位。

ECC(Elliptic Curve) 基于“椭圆曲线离散对数”的数学难题，使用特定的曲线方程和基点生成公钥和私钥，子算法ECDHE用于秘钥交换，ECDSA用于数字签名。比起RSA,ECC在安全强度和性能上都有明显的优势，160位ECC相当于1024位RSA，244位ECC相当于2048RSA.

ECC 常用曲线 P-256和x25519，x25519被认为是最安全、最快速的曲线。

混合加密 刚开始使用非对称算法，比如RSA，ECDHE，先解决秘钥交换的问题，然后用随机数产生对称算法使用会话秘钥，再用公钥加密。



非对称加密的理解：私钥签名公钥验证，公钥加密私钥解密，引用知乎上的一段话

> 你只要想：既然是加密，那肯定是不希望别人知道我的消息，所以只有我才能解密，所以可得出公钥负责加密，私钥负责解密；同理，既然是签名，那肯定是不希望有人冒充我发消息，只有我才能发布这个签名，所以可得出私钥负责签名，公钥负责验证。

参考资料:https://www.cnblogs.com/yungyu16/p/13332882.html?utm_source=tuicool

25、数字签名与证书

摘要算法 实现数据完整性的主要手段，也就是常说的散列函数，哈希函数。
摘要算法特点 可以把任意长度的数据“压缩”成固定长度且独一无二的“摘要”字符串；过程不可逆，不能从摘要逆推出原文；对输入具有“单向性”和“雪崩效应”，输入的微小不同会导致输出的剧烈变化，所以也被TLS用来生成伪随机数（PRF，pseudo random function）
摘要算法举例 MD5(Message-Digest 5),SHA-1(Secure Hash Algorithm 1),MD5能生成16字节的摘要，SHA-1能生成20字节的摘要，这两个算法安全强度比较低，在TLS里已经被禁用了，目前TLS推荐使用的时SHA-1的后继者SHA-2
SHA-2 实际时一系列摘要算法的统称，总共有六种，常用的有SHA224,SHA256,SHA384,分别能生成28字节，32字节，48位字节的摘要。
摘要算法不具有机密性 如果明文传输，那么黑客可以修改消息后把摘要也改了，那么还是无法鉴别不出完整性（这里指的完整是数据传输过程中不被篡改），所以完整性必须要建立在机密性之上。
数字签名 非对称加密里的私钥，使用私钥加上摘要算法，就能实现数字签名，同时实现身份认证和不可否认。这里身份认证和不可否认的特性来自于私钥加密，如果可以用其公钥解密，就证明这个信息是用对应私钥来解密的，身份认证->唯一持有私钥的这个人，不可否认->信息是用唯一持有私钥这个人的加密的，而不是别人，无法抵赖。
签名：可以理解为持有私钥的人使用私钥对数据摘要加密的过程。
验签：可以理解为持有对应公钥的人使用公钥解密，并获取数据得到摘要，使用生成的摘要与发送过来摘要比对的这整个过程。
数字证书与CA
以下一段话是我对于数字证书与CA的理解，非笔记

> 假设小红发布了她的公钥，那在互联网上怎么证明这个公钥就是小红的公钥呢？
> 举个例子，我们在生活中怎么证明我是我呢？一般是拿出身份证，然后人脸比对，身份证由权威的公安颁发，我可以用身份证来证明我是我，是基于大家对公安机关，对公安颁发的身份证认可的前提的。
> 那么一样的道理，公钥也需要有这么个机构去建立公钥的信任链，即CA(Certificate Authority，证书认证机构)，其用自身的信誉来保证公钥无法伪造是可信的。

- CA对公钥的签名认证也是有格式的，包含序列号，用途，颁发者，有效时间等，把这些打成一个包再签名，完整的证明公钥关联的各种信息，形成"数字证书"（Certificate）。
- 知名的CA全世界就那么几家，比如DigCert，VeriSign,Entrust,Let’s Encrypt等，它们签发的证书分DV,OV,EV三种，区别在于可信程度。
- DV可信度是最低的，只是域名级别的可信
- EV是最高的，经过了法律和审计的严格核查，可以证明网络拥有者的身份。
- 操作系统和浏览器都内置了各大CA的根证书，上网时只要服务器发过来它的证书，就可以验证证书里的签名，顺着证书链一层层验证，直到找到根证书，就能证明证书是可信的，从而里面的公钥也是可信的。
  证书体系的弱点 如果CA失误或者被欺骗，签发了错误的证书，虽然证书是真的，可它代表的网站却是假的，或者CA被黑客攻陷，CA本身有恶意，那么整个信任链里的所有证书就都不可信了。针对第一种，开发了CRL，证书吊销列表，OCSP在线证书状态协议，及时废止有问题的证书，对于第二种，因为涉及证书太多，撤销对CA的信任，列入黑名单，这样它颁发的所有证书就都会被认为是不安全的。

26、TLS1.2连接过程解析

TLS协议的组成 包含多个子协议，可以理解为它是由几个不同职责的模块组成，比较常用的由记录协议，警报协议，握手协议，变更密码协议等。
记录协议 规定了TLS收发数据的基本单位：记录（record）,其他子协议都需要通过记录协议发出，多个记录可以在一个TCP包里一次性发出，也并不需要像TCP那样返回ACK。
警报协议 职责是向对方发出警报信息，收到警报后另一方可以选择继续，也可以立即终止连接。
握手协议 TLS里最复杂的子协议，浏览器和服务器会在握手过程中协商TLS版本号，随机数，密码套件等信息，然后交换证书和密钥参数，最终双方协商得到会话密钥，用于后续的混合加密系统。
变更密码范围协议 职责为通知对方后续数据都将使用密码保护。
双向认证 单向认证只证明了服务器的身份，没有认证客户端的身份，这是因为通常单向认证通过后已经建立了安全通信，用账号和密码等简单手段就可以确认用户身份了，但为了防止账号密码被盗，有时候还会使用U盾给用户颁发客户端证书，从而实现双向认证。

关于整个TLS连接的过程，因为我了解比较浅，说的也不专业，这里就不讲解了，推荐去看这篇博客了解详细过程：
https://blog.csdn.net/qq_38975553/article/details/112987893

27、TLS1.3特性解析

- TLS1.3的三个主要改进目标：兼容、安全、性能

28、HTTPS连接太慢怎么办

- HTTPS连接大致可以划分成两个部分：

1. 建立连接时的非对称加密握手（TLS握手）
2. 握手后的对称加密报文传输

- 通常说的“HTTP连接慢”指的是刚开始建立连接的那段时间。
  相较于HTTP，HTTPS多出的耗时
- TLS握手（多消耗1-2个RTT）
- 产出用于秘钥交换的ECDHE公私钥对
- 验证网站数据证书（访问CA获取CRL，OCSP）
- 非对称加解密处理pre-master
  HTTPS优化
  硬件优化
- 更快的CPU，最好内建AES优化，这样可以加速握手、传输
- SSL加速卡：一种减轻处理器过多参与TLS的公开密钥加密所产生负担的办法，是TLS 前身SSL的硬件加速器
- SSL加速服务器 ：用专门的服务器集群来彻底接下TLS握手时的加密解密计算，比单独的SSL加速卡强大得多。
  软件优化：软件升级，协议优化
- 软件升级： 在使用的软件尽量升级到最新版本，不过对于中大型公司来说，逐一升级机器费时费力，也有很高的风险影响正常的线上服务。
- 协议优化：尽量使用TLS1.3，因为它大幅简化了握手的过程，完全握手只需要消息传递一个来回，也更安全。如果只能用TLS1.2，秘钥交换协议应尽量使用椭圆曲线的ECDHE算法，其运算速度快，安全性高。(Nginx里可以设置ssl_ciphers,ssl_ecdh_curve配置服务器使用的密码套间和椭圆曲线，优先使用的放在前面)
  证书优化：证书传输，证书验证
  服务器需要把自己的证书链全发给客户端，然后客户端再逐一验证，这个过程也比较耗时
  服务器的证书可以选择椭圆曲线（ECDSA）证书而不是RSA证书，因为224bit ECC相当于2048bit RSA,传输起来椭圆曲线更加节约带宽，也能减少客户端的运算量。
  CRL证书吊销列表，由CA定期发布，里面是所有被撤销信任的证书序号，可以用来判断证书是否有效。（弊端：因为定时发布，所以有时效性问题，随着CRL里面信息增多，下载的CRL信息越来越大，实用性低，现在基本不用，取而代之的是OCSP）
  OCSP 在线证书协议，用于向CA发送查询请求，让CA返回证书的有效状态。
  OCSP Stapling(OCSP 装订)，可以让服务器预先访问CA获取OCSP响应，然后在握手时随证书一起发给客户端，免去客户端连接CA服务器查询的时间。
  会话复用
- TLS握手主要目的是得到主密钥 Master Secret，而主密钥每次连接都需要重新计算，这样也会降低效率。
- 把这个主密钥缓存一下重用，就可以免去握手和计算的成本，这种做法称为 会话复用，会话复用分为两种：

1. Session ID:服务器和客户端首次连接后各自保存一个会话的ID号，内存里存储主密钥和其他相关信息，当客户端再次连接时发一个ID过来，服务器在内存中找，如果找到就直接用主密钥恢复会话状态，跳过验证和密钥交换，只用一个消息往返就能建立安全通信。
2. Session Ticket,会话票证，类似HTTP的cookie,存储的责任由服务器移到了客户端，服务端加密会话信息，用New Session Ticket消息发给客户端，让客户端保存，重连的时候客户端使用扩展 session_ticket 发送Ticket,服务器解密后验证有效期，就可以恢复会话，开始加密通信。
3. 预共享秘钥 Pre-shared Key，简称 PSK，发送Ticket的同时会带上应用数据，免去服务器确认的步骤（弊端：容易受到重放攻击，解决方案，在消息里加入时间戳，nonce验证，或者一次性票据，限制重放）

- Session ID和Session Ticket这两种会话复用技术在TLS1.3中均已废除，TLS1.3中只能使用PSK实现会话复用。

29、是否应迁移到HTTPS

- 迁移的必要性：迁移到HTTPS已是大潮
- 迁移顾虑：慢（服务器成本，客户端时延，影响客户端体现），贵（证书申请和维护成本高），难（HTTPS涉及知识点太多，太复杂，有一定技术门槛，不能很快上手，关联到密码学，TLS，PKI等多个领域）
  关键难点解决
- 申请证书（Let's Encrypt）
- 证书注意事项：1、同时申请RSA和ECDSA,在nginx配置成双证书验证，以做兼容 2、如果申请RSA证书，私钥至少2048位，摘要算法应选择SHA-2 3、处于安全考虑，Let's Encrypt证书有效期很短，只有90天，时间一到就过期，需要定期更新。
- 配置HTTS
- 在443端口上开启HTTPS服务，例如nginx，listen 443 ssl，再配置上证书文件就可以实现最基本的HTTPS

```
listen 443 ssl;
ssl_certificate xxx_rsa.crt;
ssl_certificate_key xxx_rsa.key;
ssl_certificate xxx_ecc.crt;
ssl_certificate_key xxx_ecc.key;
```

强制只支持TLS1.2以上的协议，打开session ticket会话复用

```
ssl_protocols   TLSv1.2,TLSv1.3;

ssl_session_timeout 5m;
ssl_session_tickets on;
ssl_session_ticket_key ticket.key;
```

密码套间以服务器套件优先，这样可以避免客户端恶意选择安全性较低的安全套件

```
ssl_prefer_server_ciphers on;
ssl_ciphers ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-CHAC
HA20-POLY1305:ECDHE+AES128:!MD5:!SHA1;
```

服务器名称指示
虚拟主机：在HTTP协议里多个域名可以同时在一个IP地址上运行
SNI（Server Name Indication）,补充条款，TLS的扩展，客户端会在Client Hello时带上域名信息，这样服务器就可以根据名字而不是IP地址来选择证书
重定向跳转
切换协议后，原来HTTP站点也不会马上弃用，需要把不安全的HTTP网址用301或者302重定向到新的HTTPS网站，例如nginx里

```
return 301 https://$host$request_url;
rewrite ^ https://$host$request_url permanent;
```

- HSTS (HTTP严格传输，HTTP Strict Transport Security),https服务器需要在发出的响应头里添加一个 Strict-Transport-Security,再设定一个有效期，在有效期内网站必须使用HTTPS协议，不允许用HTTP。