

26 | 信任始于握手：TLS1.2连接过程解析

2019-07-26 Chrono

《透视HTTP协议》

课程介绍 >



讲述：Chrono

时长 11:47 大小 16.19M



经过前几讲的介绍，你应该已经熟悉了对称加密与非对称加密、数字签名与证书等密码学知识。

有了这些知识“打底”，现在我们就可以正式开始研究 HTTPS 和 TLS 协议了。

HTTPS 建立连接

当你在浏览器地址栏里键入“https”开头的 URI，再按下回车，会发生什么呢？

领资料

回忆一下 [第 8 讲](#) 的内容，你应该知道，浏览器首先要从 URI 里提取出协议名和域名。因为协议名是“https”，所以浏览器就知道了端口号是默认的 443，它再用 DNS 解析域名，得到目标的 IP 地址，然后就可以使用三次握手与网站建立 TCP 连接了。



在 HTTP 协议里，建立连接后，浏览器会立即发送请求报文。但现在是 HTTPS 协议，它需要再用另外一个“握手”过程，在 TCP 上建立安全连接，之后才是收发 HTTP 报文。

这个“握手”过程与 TCP 有些类似，是 HTTPS 和 TLS 协议里最重要、最核心的部分，懂了它，你就可以自豪地说自己“掌握了 HTTPS”。

TLS 协议的组成

在讲 TLS 握手之前，我先简单介绍一下 TLS 协议的组成。

TLS 包含几个子协议，你也可以理解为它是由几个不同职责的模块组成，比较常用的有记录协议、警报协议、握手协议、变更密码规范协议等。

记录协议 (Record Protocol) 规定了 TLS 收发数据的基本单位：记录 (record)。它有点像是 TCP 里的 segment，所有的其他子协议都需要通过记录协议发出。但多个记录数据可以在一个 TCP 包里一次性发出，也并不需要像 TCP 那样返回 ACK。

警报协议 (Alert Protocol) 的职责是向对方发出警报信息，有点像是 HTTP 协议里的状态码。比如，protocol_version 就是不支持旧版本，bad_certificate 就是证书有问题，收到警报后另一方可以选择继续，也可以立即终止连接。

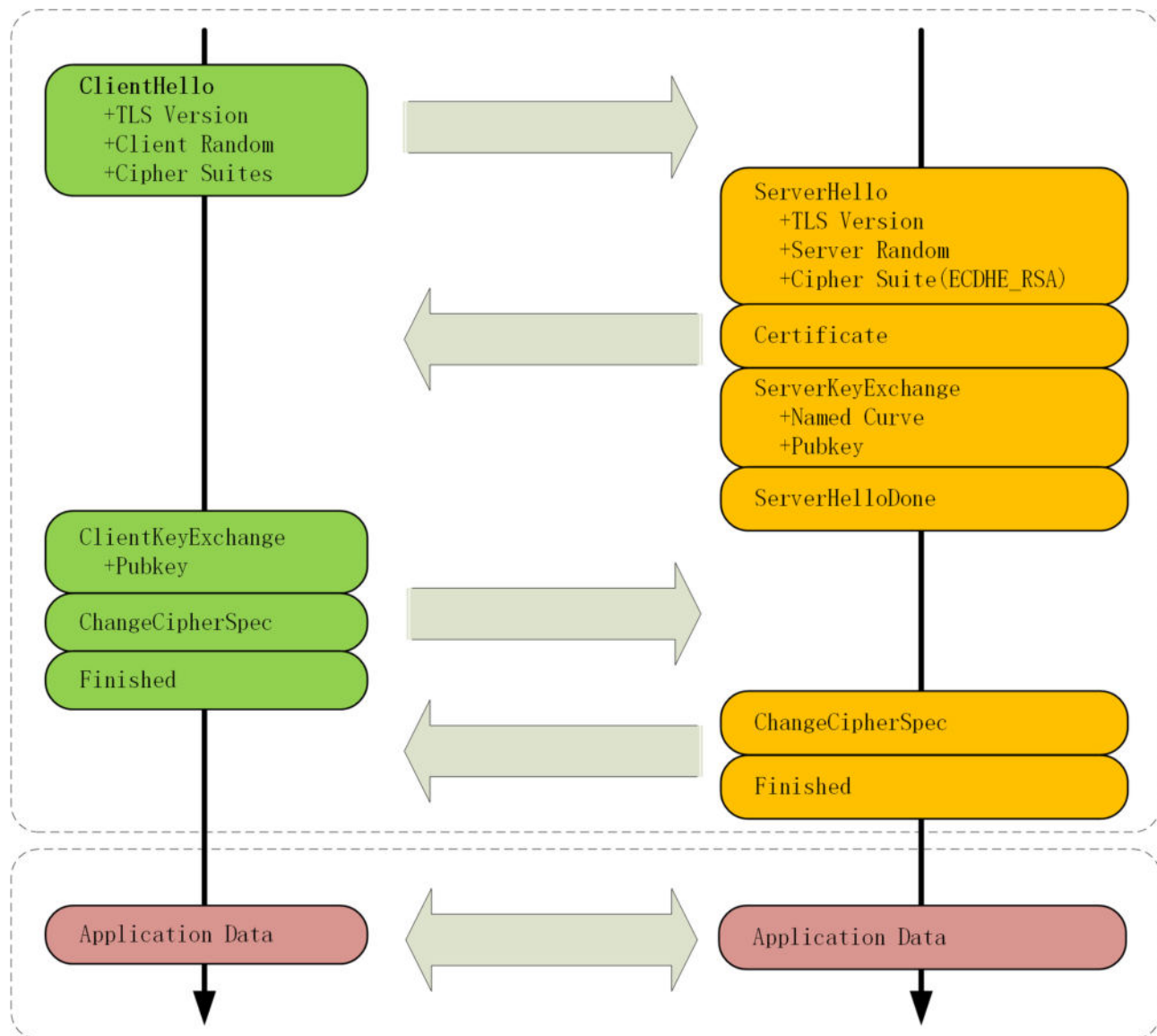
握手协议 (Handshake Protocol) 是 TLS 里最复杂的子协议，要比 TCP 的 SYN/ACK 复杂的多，浏览器和服务端会在握手过程中协商 TLS 版本号、随机数、密码套件等信息，然后交换证书和密钥参数，最终双方协商得到会话密钥，用于后续的混合加密系统。

最后一个是**变更密码规范协议** (Change Cipher Spec Protocol)，它非常简单，就是一个“通知”，告诉对方，后续的数据都将使用加密保护。那么反过来，在它之前，数据都是明文的。

下面的这张图简要地描述了 TLS 的握手过程，其中每一个“框”都是一个记录，多个记录组合成一个 TCP 包发送。所以，最多经过两次消息往返（4 个消息）就可以完成握手，然后就可以在安全的通信环境里发送 HTTP 报文，实现 HTTPS 协议。

领资料





抓包的准备工作

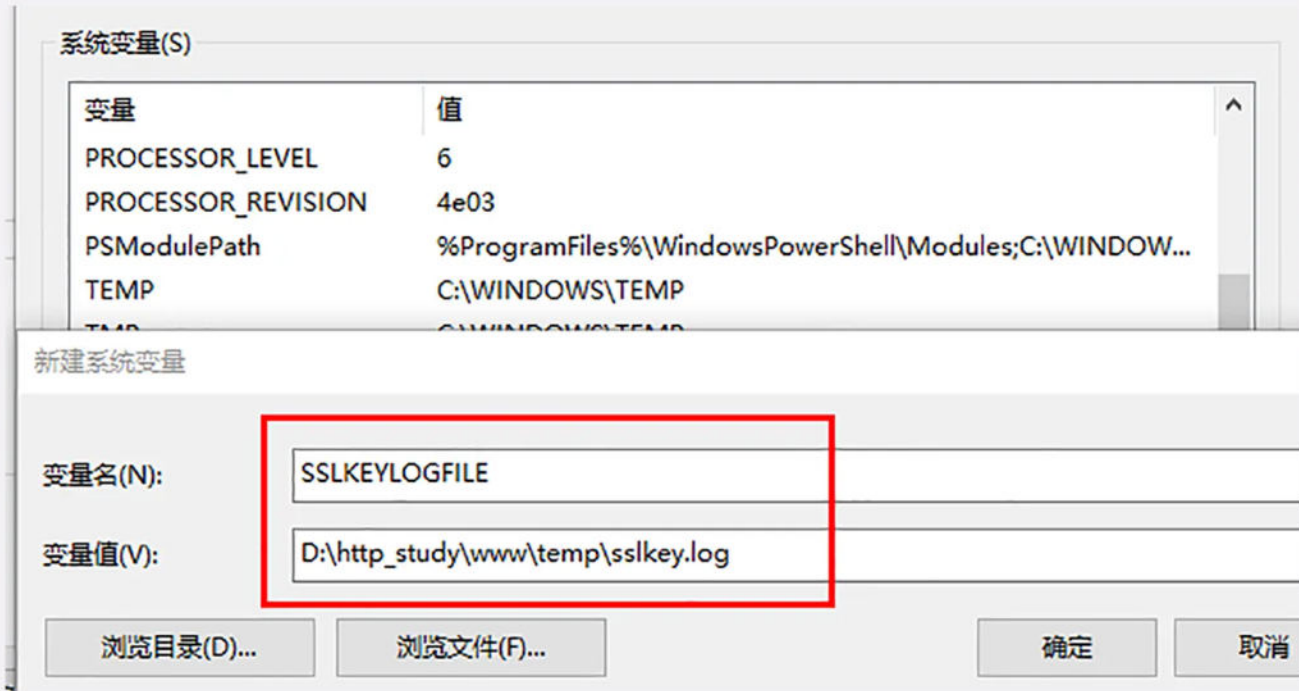
这次我们在实验环境里测试 TLS 握手的 URI 是“/26-1”，看了上面的图你就可以知道，TLS 握手的前几个消息都是明文的，能够在 Wireshark 里直接看。但只要出现了“Change Cipher Spec”，后面的数据就都是密文了，看到的也就会是乱码，不知道究竟是什么东西。

为了更好地分析 TLS 握手过程，你可以再对系统和 Wireshark 做一下设置，让浏览器导出握手过程中的秘密信息，这样 Wireshark 就可以把密文解密，还原出明文。

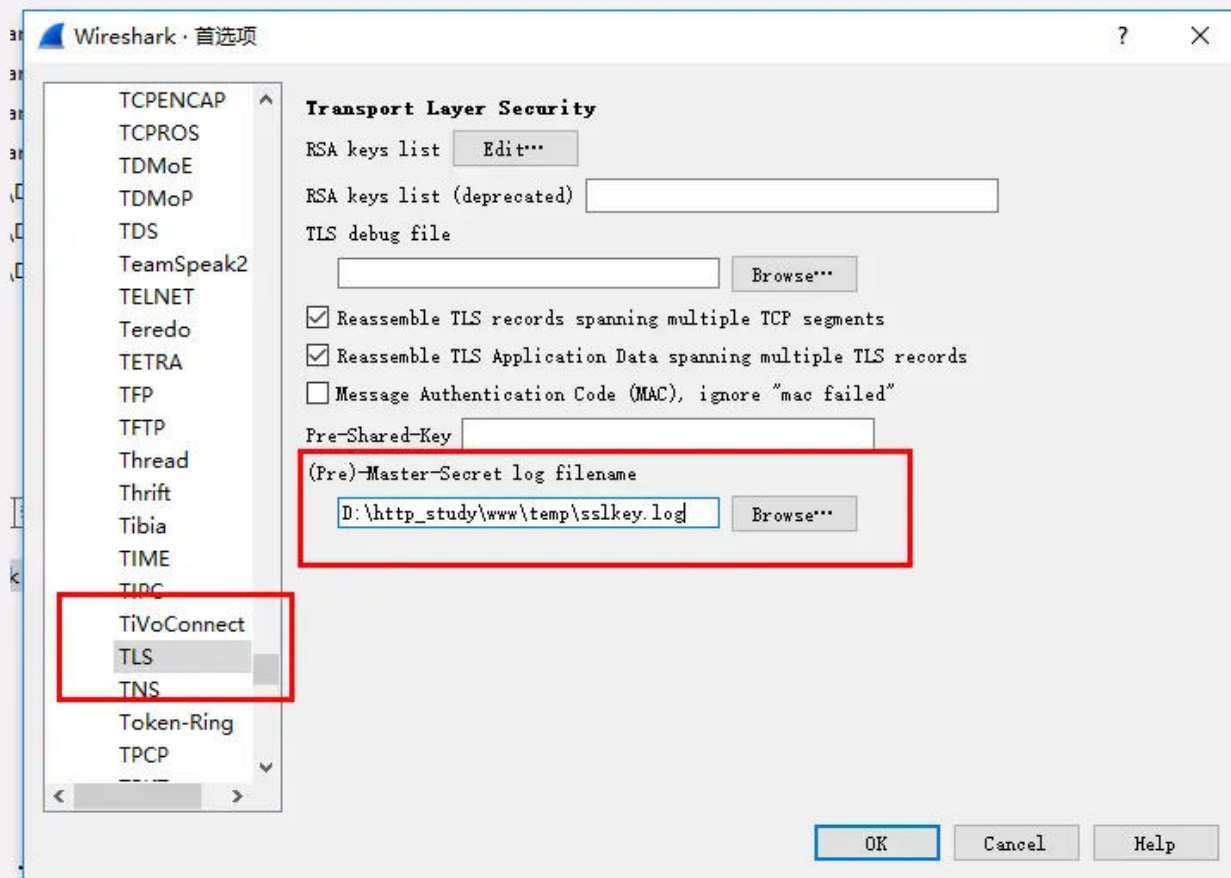
领资料

首先，你需要在 Windows 的设置里新增一个系统变量“SSLKEYLOGFILE”，设置浏览器日志文件的路径，比如“D:\http_study\www\temp\sslkey.log”（具体的设置过程就不详细说了，可以在设置里搜索“系统变量”）。





然后在 Wireshark 里设置“Protocols-TLS”（较早版本的 Wireshark 里是“SSL”），在“(Pre)-Master-Secret log filename”里填上刚才的日志文件。



领资料

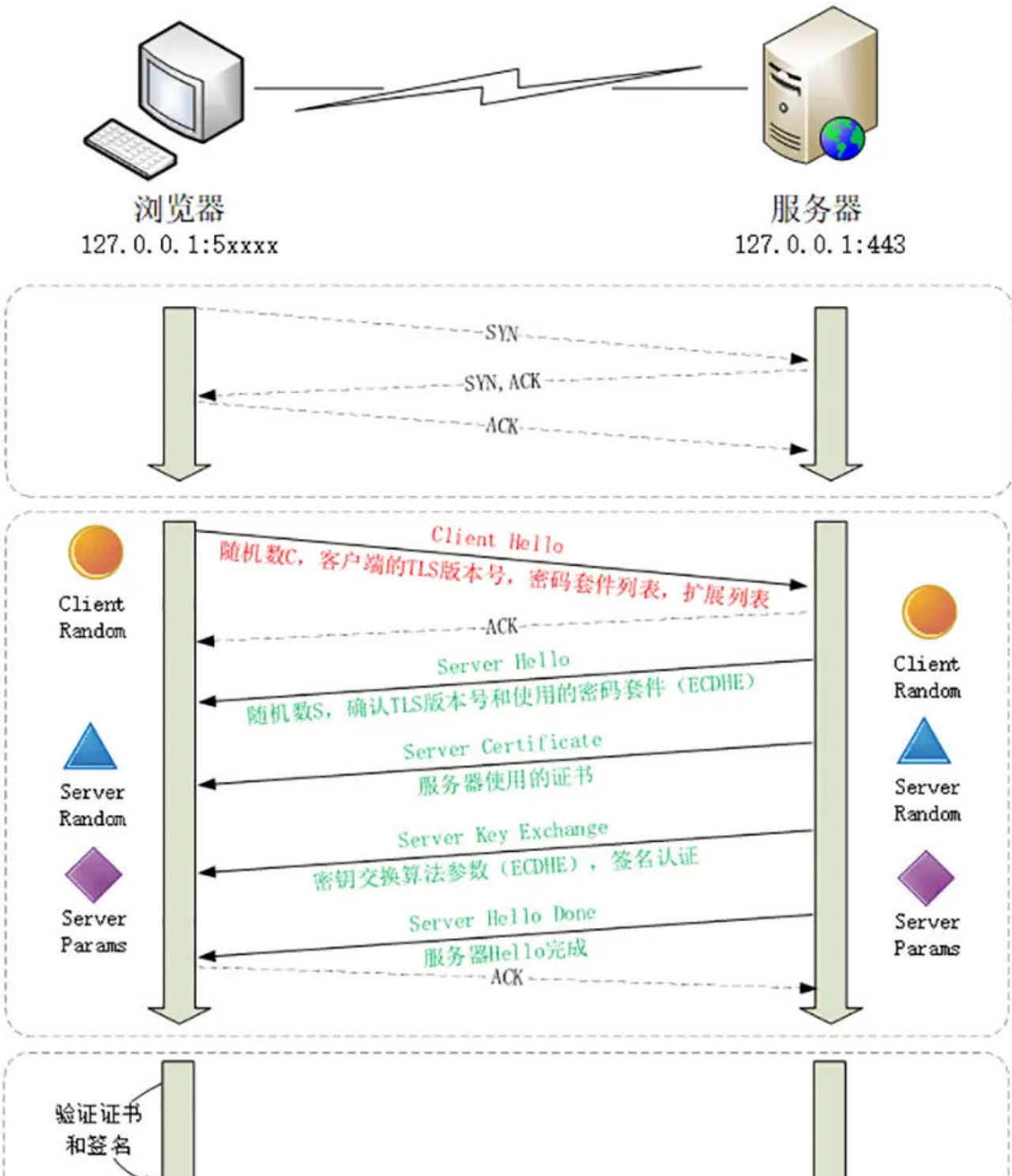


设置好之后，过滤器选择“tcp port 443”，就可以抓到实验环境里的所有 HTTPS 数据了。

如果你觉得麻烦也没关系，GitHub 上有抓好的包和相应的日志，用 Wireshark 直接打开就行。

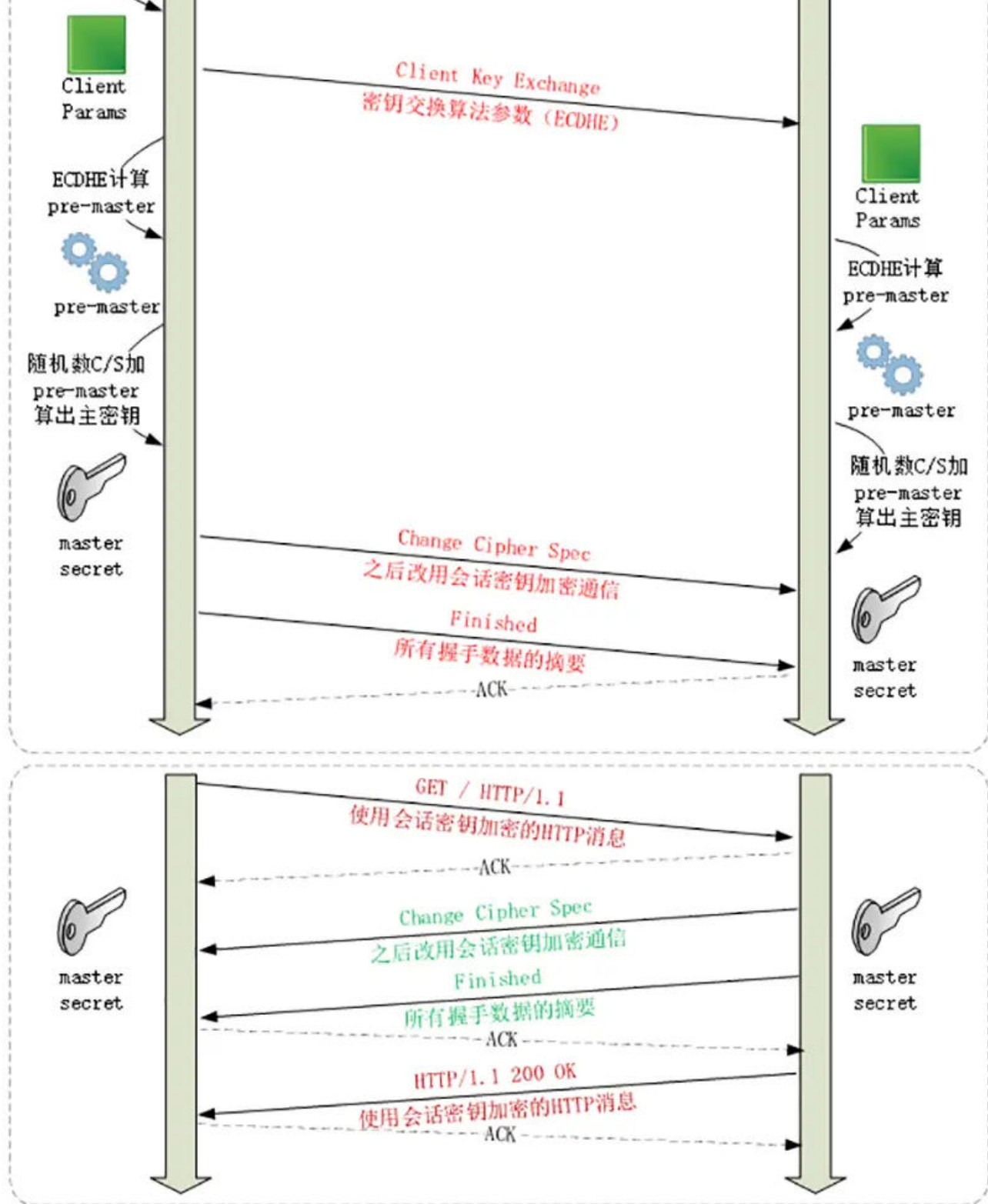
ECDHE 握手过程

刚才你看到的是握手过程的简要图，我又画了一个详细图，对应 Wireshark 的抓包，下面我就用这个图来仔细剖析 TLS 的握手过程。



领资料





在 TCP 建立连接之后，浏览器会首先发一个“**Client Hello**”消息，也就是跟服务器“打招呼”。里面有客户端的版本号、支持的密码套件，还有一个**随机数 (Client Random)**，用于后续生成会话密钥。

领资料



复制代码

```
1 Handshake Protocol: Client Hello
2   Version: TLS 1.2 (0x0303)
```

```
3 Random: 1cbf803321fd2623408dfe...
4 Cipher Suites (17 suites)
5 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (0xc02f)
6 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
```

这个的意思就是：“我这边有这些这些信息，你看看哪些是能用的，关键的随机数可得留着。”

作为“礼尚往来”，服务器收到“Client Hello”后，会返回一个“Server Hello”消息。把版本号对一下，也给出一个**随机数（Server Random）**，然后从客户端的列表里选一个作为本次通信使用的密码套件，在这里它选择了“TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384”。

 复制代码

```
1 Handshake Protocol: Server Hello
2 Version: TLS 1.2 (0x0303)
3 Random: 0e6320f21bae50842e96...
4 Cipher Suite: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384 (0xc030)
```

这个的意思就是：“版本号对上了，可以加密，你的密码套件挺多，我选一个最合适的吧，用椭圆曲线加 RSA、AES、SHA384。我也给你一个随机数，你也得留着。”

然后，服务器为了证明自己的身份，就把证书也发给了客户端（Server Certificate）。

接下来是一个关键的操作，因为服务器选择了 ECDHE 算法，所以它会在证书后发送“**Server Key Exchange**”消息，里面是**椭圆曲线的公钥（Server Params）**，用来实现密钥交换算法，再加上自己的私钥签名认证。

 复制代码

```
1 Handshake Protocol: Server Key Exchange
2 EC Diffie-Hellman Server Params
3 Curve Type: named_curve (0x03)
4 Named Curve: x25519 (0x001d)
5 Pubkey: 3b39deaf00217894e...
6 Signature Algorithm: rsa_pkcs1_sha512 (0x0601)
7 Signature: 37141adac38ea4...
```

领资料



这相当于说：“刚才我选的密码套件有点复杂，所以再给你个算法的参数，和刚才的随机数一样有用，别丢了。为了防止别人冒充，我又盖了个章。”

之后是“**Server Hello Done**”消息，服务器说：“我的信息就是这些，打招呼完毕。”

这样第一个消息往返就结束了（两个 TCP 包），结果是客户端和服务端通过明文共享了三个信息：**Client Random**、**Server Random** 和 **Server Params**。

客户端这时也拿到了服务器的证书，那这个证书是不是真实有效的呢？

这就要用到第 25 讲里的知识了，开始走证书链逐级验证，确认证书的真实性，再用证书公钥验证签名，就确认了服务器的身份：“刚才跟我打招呼的不是骗子，可以接着往下走。”

然后，客户端按照密码套件的要求，也生成一个**椭圆曲线的公钥（Client Params）**，用“**Client Key Exchange**”消息发给服务器。

 复制代码

```
1 Handshake Protocol: Client Key Exchange
2     EC Diffie-Hellman Client Params
3         Pubkey: 8c674d0e08dc27b5eaa...
```

现在客户端和服务端手里都拿到了密钥交换算法的两个参数（Client Params、Server Params），就用 ECDHE 算法一阵算，算出了一个新的东西，叫“**Pre-Master**”，其实也是一个随机数。

至于具体的计算原理和过程，因为太复杂就不细说了，但算法可以保证即使黑客截获了之前的参数，也是绝对算不出这个随机数的。

现在客户端和服务端手里有了三个随机数：**Client Random**、**Server Random** 和 **Pre-Master**。用这三个作为原始材料，就可以生成用于加密会话的主密钥，叫“**Master Secret**”。而黑客因为拿不到“Pre-Master”，所以也就得不到主密钥。

为什么非得这么麻烦，非要三个随机数呢？

领资料



这就必须说 TLS 的设计者考虑得非常周到了，他们不信任客户端或服务器伪随机数的可靠性，为了保证真正的“完全随机”“不可预测”，把三个不可靠的随机数混合起来，那么“随机”的程度就非常高了，足够让黑客难以猜测。

你一定很想知道“Master Secret”究竟是怎么算出来的吧，贴一下 RFC 里的公式：

 复制代码

```
1 master_secret = PRF(pre_master_secret, "master secret",  
2                     ClientHello.random + ServerHello.random)
```

这里的“PRF”就是伪随机数函数，它基于密码套件里的最后一个参数，比如这次的 SHA384，通过摘要算法来再一次强化“Master Secret”的随机性。

主密钥有 48 字节，但它也不是最终用于通信的会话密钥，还会再用 PRF 扩展出更多的密钥，比如客户端发送用的会话密钥（client_write_key）、服务器发送用的会话密钥（server_write_key）等等，避免只用一个密钥带来的安全隐患。

有了主密钥和派生的会话密钥，握手就快结束了。客户端发一个“**Change Cipher Spec**”，然后再发一个“**Finished**”消息，把之前所有发送的数据做个摘要，再加密一下，让服务器做个验证。

意思就是告诉服务器：“后面都改用对称算法加密通信了啊，用的就是打招呼时说的 AES，加密对不对还得你测一下。”

服务器也是同样的操作，发“**Change Cipher Spec**”和“**Finished**”消息，双方都验证加密解密 OK，握手正式结束，后面就收发被加密的 HTTP 请求和响应了。

RSA 握手过程

整个握手过程可真是够复杂的，但你可能会问了，好像这个过程和其他地方看到的不一样呢？

刚才说的其实是如今主流的 TLS 握手过程，这与传统的握手有两点不同。

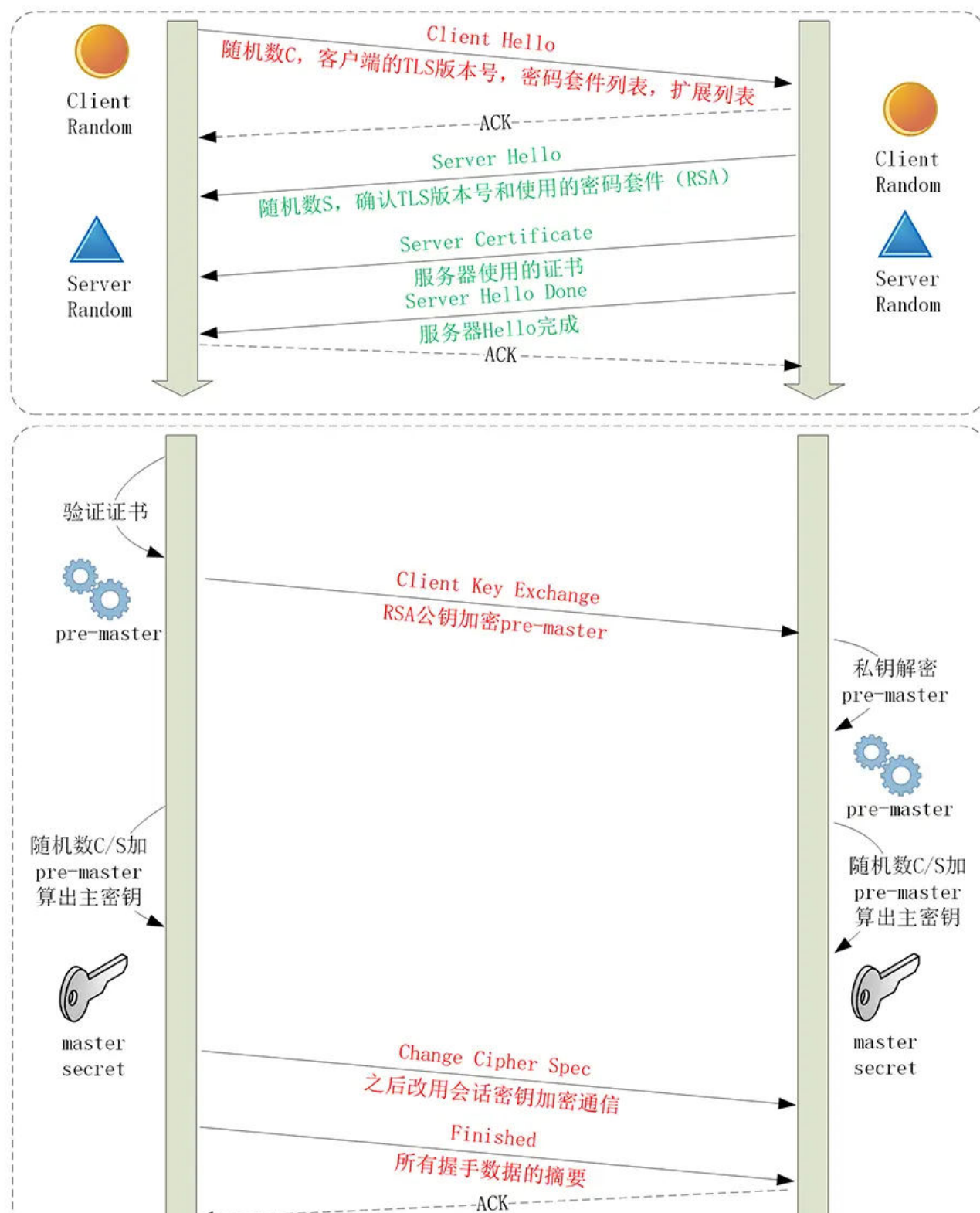
第一个，使用 ECDHE 实现密钥交换，而不是 RSA，所以会在服务器端发出“Server Key Exchange”消息。

领资料



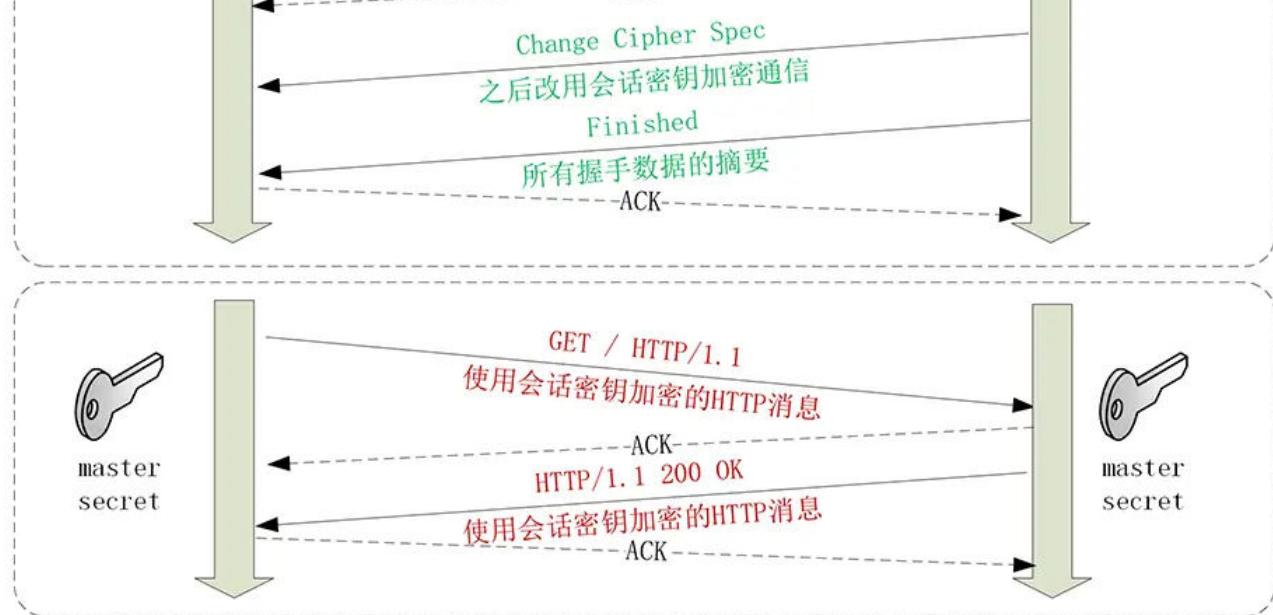
第二个，因为使用了 ECDHE，客户端可以不用等到服务器发回“Finished”确认握手完毕，立即就发出 HTTP 报文，省去了一个消息往返的时间浪费。这个叫“**TLS False Start**”，意思就是“抢跑”，和“TCP Fast Open”有点像，都是不等连接完全建立就提前发应用数据，提高传输的效率。

实验环境在 440 端口（<https://www.chrono.com:440/26-1>）实现了传统的 RSA 密钥交换，没有“False Start”，你可以课后自己抓包看一下，这里我也画了个图。



领资料





大体的流程没有变，只是“Pre-Master”不再需要用算法生成，而是客户端直接生成随机数，然后用服务器的公钥加密，通过“**Client Key Exchange**”消息发给服务器。服务器再用私钥解密，这样双方也实现了共享三个随机数，就可以生成主密钥。

双向认证

到这里 TLS 握手就基本讲完了。

不过上面说的是“**单向认证**”握手过程，只认证了服务器的身份，而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、密码等简单的手段就能够确认用户的真实身份。

但为了防止账号、密码被盗，有的时候（比如网上银行）还会使用 U 盾给用户颁发客户端证书，实现“**双向认证**”，这样会更加安全。

双向认证的流程也没有太多变化，只是在“**Server Hello Done**”之后，“**Client Key Exchange**”之前，客户端要发送“**Client Certificate**”消息，服务器收到后也把证书链走一遍，验证客户端的身份。

小结

今天我们学习了 HTTPS/TLS 的握手，内容比较多、比较难，不过记住下面四点就可以。

领资料



1. HTTPS 协议会先与服务器执行 TCP 握手，然后执行 TLS 握手，才能建立安全连接；
2. 握手的目标是安全地交换对称密钥，需要三个随机数，第三个随机数“Pre-Master”必须加密传输，绝对不能让黑客破解；
3. “Hello”消息交换随机数，“Key Exchange”消息交换“Pre-Master”；
4. “Change Cipher Spec”之前传输的都是明文，之后都是对称密钥加密的密文。

课下作业

1. 密码套件里的那些算法分别在握手过程中起了什么作用？
2. 你能完整地描述一下 RSA 的握手过程吗？
3. 你能画出双向认证的流程图吗？

欢迎你把自己的学习体会写在留言区，与我和其他同学一起讨论。如果你觉得有所收获，也欢迎把文章分享给你的朋友。



课外小贴士

- 01 TLS 中记录协议原本定义有压缩方式，但后来发现存在安全漏洞（CRIME 攻击），所以现在这个字段总是 NULL，即不压缩。
- 02 在 TLS1.2 里，客户端和随机数的长度都是 28 字节，前面的四个字节是 UNIX 时间戳，但并没有实际意义。


领资料



- 03 Chrome 开发者工具的“Security”面板里可以看到 HTTPS 握手时选择的版本号、密码套件和椭圆曲线，例如“ECDHE_RSA with X25519, and AES_256_GCM”。
- 04 ECDHE 即“短暂 – 椭圆曲线 – 迪菲 – 赫尔曼” (ephemeral Elliptic Curve Diffie–Hellman) 算法，使用椭圆曲线增强了 DH 算法的安全性和性能，公钥和私钥都是临时生成的。
- 05 在 Wireshark 抓包里你还会看见“Session ID”“Extension”等字段，涉及会话复用和扩展协议，后面会讲到。

分享给需要的人，Ta 订阅超级会员，你将得 50 元

Ta 单独购买本课程，你将得 20 元

 生成海报并分享

领资料

 赞 11  提建议 

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。



学习推荐

JVM + NIO + Spring

各大厂面试题及知识点详解

限时免费 



精选留言 (115)

 写留言



彩色的沙漠

2019-08-01

浏览留言区，留言区同学和我有一样的疑问“Client Params 和 Server Params 都可以被截获，为何中间人没法通过这两个信息计算 Pre-Master Secret 呢？”

我去网上找了关于ECDHE握手过程，这个可以帮助大家更好的理解ECDHE，具体过程如下：

(1)：客户端随机生成随机值 R_a ，计算 $P_a(x, y) = R_a * Q(x, y)$ ， $Q(x, y)$ 为全世界公认的某个椭圆曲线算法的基点。将 $P_a(x, y)$ 发送至服务器。

(2)：服务器随机生成随机值 R_b ，计算 $P_b(x, y) = R_b * Q(x, y)$ 。将 $P_b(x, y)$ 发送至客户端。

(3)：客户端计算 $S_a(x, y) = R_a * P_b(x, y)$ ；服务器计算 $S_b(x, y) = R_b * P_a(x, y)$

(4)：算法保证了 $S_a = S_b = S$ ，提取其中的 S 的 x 向量作为密钥（预主密钥）。

@引用

作者：Mrpre

来源：CSDN

原文：<https://blog.csdn.net/mrpre/article/details/78025940>

版权声明：本文为博主原创文章，转载请附上博文链接！

领资料



作者回复: 非常好的同学, 大力表扬!

共 19 条评论>

👍 64



J.Smile

2020-01-19

之前面试阿里第二轮的时候, 面试官就问我关于ssl握手的问题, 其实我觉得像这种问题回答不出来也很正常, 毕竟这么复杂的流程谁能记得住呢? 使用现成的nginx+ssl的配置已经可以解决大多数问题了。

总结下TLS的握手过程:

第一阶段: C/S两端共享Client Random、Server Random 和 Server Params信息

客户端--->服务器:

客户端的版本号、支持的密码套件, 还有一个随机数 (Client Random)

服务端--->客户端:

客户端的版本号、选择的客户端列表的密码套件如: TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384、随机数随机数 (Server Random)

服务端--->客户端:

服务端证书 (Server Certificate)

服务端--->客户端:

发送Server Key Exchange类型的请求, 携带椭圆曲线的公钥 (Server Params) 用以实现密钥交换算法, 另附私钥签名

服务端--->客户端:

发送完毕

第二阶段: 证书验证

前验条件: 客户端证书链逐级验证、证书公钥验证签名, 服务端身份验证成功 (证书合法)

客户端--->服务端

发送Client Key Exchange类型的请求, 携带椭圆曲线的公钥 (Client Params) 用以实现密钥交换算法

领资料



第三阶段：主密钥生成

客户端、服务端分别使用Client Params、Server Params通过ECDHE算法计算出随机值pre-master，然后用

Client Random、Server Random 和 Pre-Master三个值作为原材料，用PRF伪随机数函数（利用密码套件的摘要算法再次强化结果

值maser secert的随机性）计算出主密钥Master Secret，

主密钥并不是会话密钥，还会再用PRF扩展出更多的密钥，比如客户端发送用的会话密钥（client_write_key）、服务器发送用的会话密钥（server_write_key）

客户端--->服务端:

客户端发一个“Change Cipher Spec”，然后再发一个“Finished”消息，把之前所有发送的数据做个摘要，再加密一下，让服务器做个验证.

服务端--->客户端:

服务器也是同样的操作，发“Change Cipher Spec”和“Finished”消息，双方都验证加密解密 OK，握手正式结束.

作者回复: 总结的非常好。

领资料

共 5 条评论 >

👍 37



magicnum

2019-07-26

比如TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256：使用ECDHE进行密钥交换（文中已经讲了，用它算出Pre_Master，成会话密钥Master Secret。密钥交换过程中应该会使用到



非对称加密中的公钥加密），RSA进行身份验证（私钥加密公钥解密），使用128位GCM分组工作模式的AES进行消息和会话密钥的对称加密（加密真正的消息），使用SHA256摘要算法（如HMAC、PRM）对数据签名，保证数据完整性

作者回复: 说的很好。

共 3 条评论 >

👍 14



极夜

2019-08-30

老师，我通过nslookup获得百度的一个ip为180.101.49.11，然后用https://180.101.49.11访问，浏览器会提示建立的连接不安全。在chrome浏览器的security页面中，连接走的还是TLS，但该网页缺失认证。我的理解是，证书在访问网页的时候就返回了，但证书只能证明某个公钥是属于某个域名的，不能证明公钥是否归属某个IP，是不是这样呢？

作者回复: 这是因为申请证书的时候一般都是绑定在域名上，证书证明的是域名而不是ip，所以无法验证网址的合法性。

如果申请证书时就绑定ip，那么就没问题了，但几乎没有人会这么做，因为ip地址会变，而域名通常是稳定的。



👍 13



亚洲舞王.尼古拉斯赵...

2019-08-07

第二个问题：客户端使用tcp链接明文将自己的随机数、密码套件、tls版本号发送给服务端，服务端根据自己支持的密码套件从客户端的密码套件中选取一个最合适的密码套件，协商出tls版本，将协商好的密码套件、tls版本以及自己的随机数明文告诉客户端，并将自己的证书发送给客户端，并结束

客户端收到证书之后去ca一级一级验证证书的有效性，验证通过后，客户端使用随机数生成pre-master并用服务器的公钥进行加密传给服务端，服务端使用自己的私钥进行解密，使用解密后的值与客户端随机数，自己的随机数进行计算，得出master secret；这时候，客户端使用三个值也能计算出master secret，客户端告诉服务器我之后都使用加密进行通信了，结束；服务端也告诉客户端，我也要开始使用加密通信了，over

接下来两个人使用计算出来的master secret进行消息加密，两人balabala，并使用master secret进行解密

作者回复: good。



👍 13

领资料





看，有只猪

2019-08-27

老师，你看看我这样理解对吗？

ECDHE中，没有采用服务器公钥来加密数据，而是采用交换两端的椭圆曲线公钥来保证pre_master的安全性

RSA中pre_master由客户端生成，采用服务器公钥加密pre_master来保证pre_master的安全性

作者回复：理解的非常正确。

共 3 条评论 >

👍 10



追风筝的人

2020-05-11

在非对称加密算法中，公钥是公开信息，不需要保密,那用私钥加密，公钥解密的话（验证签名过程），其他知道公钥的人也可以解密，怎么确认发送者的身份？

作者回复：这里确实有点绕，要静下心来慢慢理解。

- 1.私钥只能被一个人秘密持有，别人是不会有。
- 2.任何人都可以用公钥解密私钥加密的数据，那么就证明数据是被对应的私钥加密的。
- 3.从1/2可以推出，数据必然是私钥持有者发出的，否则公钥必然会解密失败。
- 4.从3推出，发送者就是私钥持有者，也就确认了发送者的身份。



👍 8



lesserror

2019-12-18

老师，以下问题，麻烦回答一下：

- 1.为了更好地分析 TLS 握手过程，你可以再对系统和 Wireshark 做一下设置，让浏览器导出握手过程中的秘密信息，这样 Wireshark 就可以把密文解密，还原出明文。这不是明文传输的嘛？ Wireshark 就可以把密文解密这句话是不是有问题啊？
- 2.浏览器直接发送的TLS1.2的版本，那么为什么只发这个，不发TLS1.3的呢？
- 3.这里服务器是不是有两套公私钥，一个是证书的公私钥，一个是椭圆曲线的公私钥。服务器在证书后发送“Server Key Exchange”消息之后的签名用的是证书的私钥加密的？

作者回复：

1.对于tls通信的双方来说，tls只是加密了通信链路，在通信的两个端点必然是要解密的，也就是明文，不然就无法操作了。

设置系统和wireshark就是告诉浏览器，把密钥导出来，然后wireshark用这个密钥来解密，但传输的

领资料



数据仍然是加密的，如果没有这个密钥wireshark也是看不出明文的。
密文解密的前提是有密钥，如果没有密钥通信就是安全的。

2.看后面一讲，介绍了tls1.3，这里就不重复了。

3.是的。椭圆曲线的密钥用于ecdhe交换会话密钥，证书的私钥用来身份验证，对消息签名。
但椭圆曲线的密钥是临时生成的，每次握手都不固定，见答疑篇。



Ben

2019-12-10

有个疑问没有想清楚：client在发送“client key exchange”消息之前需要把client的证书发送给server吗？

我在想server发送给client的“server key exchange”消息需要签名认证，那么client发送给server的“client key exchange”难道不需要签名认证吗？

如果需要签名认证的话，那么是不是就需要先把client的证书发送给server做验证。

作者回复：

1.服务器会发出一个Certificate Request消息，要求客户端提供证书，这样在ServerHelloDone消息后，客户端就会发送Client Certificate提供客户端证书。如果没有Certificate Request客户端就不必提供证书。

2.tls握手分双向认证和单向认证两种，我们通常都用的是单向认证，即客户端认证服务器，服务器不认证客户端。

因为毕竟私钥签名比较费时间，而且给成千上万的客户端都颁发证书不太现实。单向认证通过后，可以再用用户名+口令的方式来验证客户端的身份。

3.少数场合，比如网银，为了加强安全，就会使用双向认证，确保通信双方的身份不被伪造。



领资料



刘政伟

2019-08-17

老师，还是没有明白服务端/客户端发送public key的用途是什么，麻烦老师再重点说一下，感谢！

作者回复：服务器客户端不会直接发送public key，如果你指的是密钥交换的过程，它实际上是ECDHE算法要求的参数，交换这些参数就可以在两边分别算出pre-master，而外部的黑客是无法计算得



到的。

发送证书是为了配合私钥签名验证客户端或服务端身份，只要签名对，就说明对方是证书所标记的实体。



7



乐潇游

2020-10-22

“主密钥有 48 字节，但它也不是最终用于通信的会话密钥，还会再用 PRF 扩展出更多的密钥，比如客户端发送用的会话密钥（client_write_key）、服务器发送用的会话密钥（server_write_key）等等，避免只用一个密钥带来的安全隐患。”这个没太理解，这个不一样的会话密钥，在对称加密算法中怎么解密呢？

作者回复: 因为客户端和服务端都共享了master secret，所以两边可以一致地生成多个密钥，比如key1、key2、key3，两边完全一样，本质上都是master secret。

这样比如客户端发数据用key1加密，服务器就用key1解密；服务器发数据不用key1，而是用key2加密，客户端收到数据用key2解密。

用多个不同的密钥就是为了安全，对抗密码分析。

共 4 条评论 >

5



Teresa

2020-04-27

有几个疑问一直想不明白，还请老师赐教：

- 1.客户端怎么验证服务端发过来的证书就是可信的？客户端在验证服务端发过来得证书得时候，客户端是直接拿取自己系统内得CA根证书，根证书里包含解密一级证书的根公钥（或者他自己就是公钥？），毕竟是证书链，所以它会先拿证书链表的第一个结点，通过刚才的根公钥进行解密（这个解密是他自己有一套算法吗？RSA或者ECC？它解密的过程还需要其他的参数吗？），解密出来的东西是一级证书包含的公钥和描述信息，然后用解密出来的公钥解密第二个结点，解出服务器的公钥和描述信息。只要最后能解析出来一个公钥和其他的描述就认为验证成功了吗？
- 2.我看整个流程下来，后面就没有解出来的证书公钥什么事了，所以这个证书的公钥在整个流程中还有其他的作用吗？
- 3.在server key exchange 那步，服务端给出的私钥签名认证是什么的私钥？客户端用它来做什么？

作者回复:

- 1.证书是自说明的，里面包含了签名算法。根证书验一级证书是可信的，那么再去验服务器证书，也

领资料



是可信的，这就构成了一个信任链。

2.验证证书用的是签名，不是公钥，是利用证书里的公钥来验证签名，因为只有公钥才能验证私钥的签名。公钥是完全公开的，不需要解析，但要保证它是可信的。

3.服务器证书里包含公钥，用来验证握手时的签名，防止中间人篡改，实现身份认证。

4.服务器证书对应的是服务器私钥，证明服务器的身份，它对握手数据签名，客户端拿到证书后验一下，就知道确实是服务器发的数据，因为对应的私钥只能是服务器有，不可能是其他人。

共 3 条评论 >

👍 5



钱

2020-04-04

😓分水岭来了，我看了三遍还是没太明白，先跳过，小本本记一下，之后在回头看看。

作者回复: 多看几遍就能理解，我觉得握手流程图画的还是挺好的。



👍 4



景

2019-07-26

Pre-Master Secret 这个不理解，是说客户端和服务器分别通过 Client Params 和 Server Params 都能计算出一样的 Pre-Master Secret 吗？如果是，为什么中间人算不出？

作者回复: 这是由DH算法决定的，DH算法是专门用作密钥交换的，它本身能够保证交换安全，具体的细节一下子说不清楚，你可以搜一下相关的资料。



👍 4



何用

2019-07-26

Client Params 和 Server Params 都可以被截获，为何中间人没法通过这两个信息计算 Pre-Master Secret 呢？

作者回复: 由密钥交换算法保证，比如rsa、ecdhe。



👍 4

领资料



乘风破浪

2021-02-22

为了验证双向验证，抓了一下招商银行u盾，奇怪的是没有抓到任何客户端证书相关的消息。其它和ecdhe流程一致。

验证百度首页，它用的是tls1.2,过程大师说的连接过程一致，一点小区别是，记录Server Hello几个记录，分散在几个tcp包里，不知道是基于什么考虑？这么做，不是浪费资源吗？

tls连接过程，貌似不是很复杂，主要是有些细节需要琢磨。

简单说，就是交换对称的密钥,并验证对方的身份。

交换对称密钥，是由3个数算出来的

pre-master,c,s

其中pre-master根据ecdhe或rsa有区别

ecdhe，由临时产生的公钥算出来的，虽然是明文，但算法可以保证黑客拿到也算不出。

验证身份就是验证对方传过来的证书

rsa，客户端产生随机数，rsa公钥加密传到对端，这种方式不具备前向安全。

一个疑问，请问大师，客户端client exchange中传递的public key甚至都没有签名，就一个明文，这怎么保证不被黑客篡改利用？

作者回复：自己动手实践的精神值得鼓励！

1.实际的网络通信情况可能会比较复杂，和实验环境不一致也是正常的，我们只要理解了核心原理就行，再深究就要考虑性价比的问题了。

2.黑客篡改握手过程中的public key没有意义，也是无效的，因为最后握手结束的时候两边都会对之前所有发送的数据做个摘要，再用会话密钥加密，发给对方验证，就是在finished那步。如果被篡改在这里就对不上了。



👍 3



俊伟

2020-01-10

握手过程：1.首先客户端发起连接，发送client hello。里面内容包括tls版本号，客户端用于加密的随机数，客户端支持的安全套件信息。

2.服务端收到客户端的请求，发送server hello。包括服务端生成的用于加密的随机数，选择的客户端的加密套件，也就是TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384。然后是server certificate发送服务端的证书信息。之后是server key exchange，里面包含生成对称加密的参数，也就是ECDHE算法生成的公钥，然后服务器用选择的摘要算法(SHA512)将整体信息进行摘要计算，用公钥进行加密，这样做是使用公钥来表示身份防止篡改。然后server hello done结束。

3.客户端收到请求之后，先根据服务端发送的证书链进行服务端的公钥认证，确认身份。确认身份无误后，根据自己本地ECDHE算法生成的Pubkey和服务端的server params也就是服务端的pubkey，生成pre master。然后根据上两步得到的两个随机数生成master secre

领资料



t。计算的同时发送client key exchange，也就是客户端的参数，ECDHE算法生成的公钥。这里由于是单项认证，客户端无需使用生成摘要表明身份。然后发送change cipher spec和finished。

4.服务端收到客户端的请求之后，也类似与客户端，根据收到的client params生成pre master，然后再生成master secret。然后发送change cipher spec和finished。最终完成安全握手连接。

综上：

1.选择的加密套件，TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384中，ECDHE的任务是进行密钥交互，RSA的任务是身份认证与不可否认。AES是256位的密钥，GCM分组模式，它的任务是保存通信安全。最后SHA384的任务是进行摘要计算，保证完整性。有点类似于设计模式里面的单一职责原则，每种算法负责一种职责。

2.RSA加密过程与此类似，只是没有server key exchange。而是由客户端生成pre master，发送client key exchange请求到服务端。参考了答疑篇，这样生成的密钥不具有向前安全。由于密钥是客户端固定生成的，随着时间的增加被破解的风险会越来越高。

作者回复: 说的非常完整，32个赞。

不过其中可能是笔误，第二步括号里的SHA512应该是SHA384。

共 3 条评论 >

👍 3



肥low

2019-10-06

老师 我看rsa握手的时候 舍去了验签的环节 这是没有必要的么

作者回复: 客户端必须要验证服务器的签名，这个步骤是不能省略的，验签发生在server hello done之后。

画的示意图里写的有点省略，“验证证书”其实包含了验证证书和验签名。

可能造成了误解，还请原谅。



👍 3

领资料



唐锋

2021-01-07

1. 为啥客户端发送finished，服务端响应ACK后，还有一个GET请求，这个请求有什么用吗？
2. 另外从抓包的内容看是发起了一个Application Data信息，这个就是对应那个GET请求吗？

作者回复: 看得很仔细。



没错，finished和ack就标志着tls握手正式完成，建立了tls通信链路，后面就是标准的http通信了，所以客户端就会发get请求，走http协议收发数据了。

而因为是tls底层通信，数据都是加密的，所以没有秘钥外界无法解密，看到的就是application data。



2



沧海一声笑

2020-06-09

然后，服务器为了证明自己的身份，就把证书也发给了客户端（Server Certificate）。接下来是一个关键的操作，因为服务器选择了ECDHE算法，所以它会在证书后发送“Server Key Exchange”消息，里面是椭圆曲线的公钥（Server Params），用来实现密钥交换算法，再加上自己的私钥签名认证。

老师这里我有一个 竟然服务端都把证书发给客户端了 按道理客户端就可以拿到服务端的公钥了 为啥服务端不用私钥直接加密Server Params

作者回复: 仔细想一下，私钥加密，任何人都可以用公钥解密，那么它的加密还有什么意义。

Server Params和Client Params都是公开的，不需要加密，只有第三个参数Pre-Master是要保密的。

交换的方式可以用ecdhe算法，或者是rsa的公钥加密，绝对不能用私钥加密。

记住，私钥只能用来签名。

共 3 条评论 >



2

领资料

