

27 | 更好更快的握手： TLS1.3特性解析

2019-07-29 Chrono

《透视HTTP协议》

课程介绍 >



讲述： Chrono

时长 10:38 大小 12.18M

▶

上一讲中我讲了 TLS1.2 的握手过程，你是不是已经完全掌握了呢？

不过 TLS1.2 已经是 10 年前（2008 年）的“老”协议了，虽然历经考验，但毕竟“岁月不饶人”，在安全、性能等方面已经跟不上如今的互联网了。

于是经过四年、近 30 个草案的反复打磨，TLS1.3 终于在去年（2018 年）“粉墨登场”，再次确立了信息安全领域的新标准。

在抓包分析握手之前，我们先来快速浏览一下 TLS1.3 的三个主要改进目标：**兼容、安全与性能**。

最大化兼容性

领资料

✕

☆

由于 1.1、1.2 等协议已经出现了很多年，很多应用软件、中间代理（官方称为“MiddleBox”）只认老的记录协议格式，更新改造很困难，甚至是不可行（设备僵化）。

在早期的试验中发现，一旦变更了记录头字段里的版本号，也就是由 0x303（TLS1.2）改为 0x304（TLS1.3）的话，大量的代理服务器、网关都无法正确处理，最终导致 TLS 握手失败。

为了保证这些被广泛部署的“老设备”能够继续使用，避免新协议带来的“冲击”，TLS1.3 不得不做出妥协，保持现有的记录格式不变，通过“伪装”来实现兼容，使得 TLS1.3 看上去“像是”TLS1.2。

那么，该怎么区分 1.2 和 1.3 呢？

这要用到一个新的**扩展协议**（Extension Protocol），它有点“补充条款”的意思，通过在记录末尾添加一系列的“扩展字段”来增加新的功能，老版本的 TLS 不认识它可以直接忽略，这就实现了“后向兼容”。

在记录头的 Version 字段被兼容性“固定”的情况下，只要是 TLS1.3 协议，握手的“Hello”消息后面就必须有“**supported_versions**”扩展，它标记了 TLS 的版本号，使用它就能区分新旧协议。

其实上一讲 Chrome 在握手时发的就是 TLS1.3 协议，你可以看一下“Client Hello”消息后面的扩展，只是因为服务器不支持 1.3，所以就“后向兼容”降级成了 1.2。

 复制代码

```
1 Handshake Protocol: Client Hello
2   Version: TLS 1.2 (0x0303)
3   Extension: supported_versions (len=11)
4     Supported Version: TLS 1.3 (0x0304)
5     Supported Version: TLS 1.2 (0x0303)
```

 领资料

TLS1.3 利用扩展实现了许多重要的功能，比如“supported_groups”“key_share”“signature_algorithms”“server_name”等，这些等后面用到的时候再说。

强化安全

TLS1.2 在十来年的应用中获得了许多宝贵的经验，陆续发现了很多的漏洞和加密算法的弱点，所以 TLS1.3 就在协议里修补了这些不安全因素。

比如：

- 伪随机数函数由 PRF 升级为 HKDF（HMAC-based Extract-and-Expand Key Derivation Function）；
- 明确禁止在记录协议里使用压缩；
- 废除了 RC4、DES 对称加密算法；
- 废除了 ECB、CBC 等传统分组模式；
- 废除了 MD5、SHA1、SHA-224 摘要算法；
- 废除了 RSA、DH 密钥交换算法和许多命名曲线。

经过这一番“减肥瘦身”之后，TLS1.3 里只保留了 AES、ChaCha20 对称加密算法，分组模式只能用 AEAD 的 GCM、CCM 和 Poly1305，摘要算法只能用 SHA256、SHA384，密钥交换算法只有 ECDHE 和 DHE，椭圆曲线也被“砍”到只剩 P-256 和 x25519 等 5 种。

减肥可以让人变得更轻巧灵活，TLS 也是这样。

算法精简后带来了一个意料之中的好处：原来众多的算法、参数组合导致密码套件非常复杂，难以选择，而现在的 TLS1.3 里只有 5 个套件，无论是客户端还是服务器都不会再犯“选择困难症”了。

领资料



密码套件名	代码
TLS_AES_128_GCM_SHA256	{0x13,0x01}
TLS_AES_256_GCM_SHA384	{0x13,0x02}
TLS_CHACHA20_POLY1305_SHA256	{0x13,0x03}
TLS_AES_128_CCM_SHA256	{0x13,0x04}
TLS_AES_128_CCM_8_SHA256	{0x13,0x05}

这里还要特别说一下废除 RSA 和 DH 密钥交换算法的原因。

上一讲用 Wireshark 抓包时你一定看到了，浏览器默认会使用 ECDHE 而不是 RSA 做密钥交换，这是因为它不具有“**前向安全**”（Forward Secrecy）。

假设有这么一个很有耐心的黑客，一直在长期收集混合加密系统收发的所有报文。如果加密系统使用服务器证书里的 RSA 做密钥交换，一旦私钥泄露或被破解（使用社会工程学或者巨型计算机），那么黑客就能够使用私钥解密出之前所有报文的“Pre-Master”，再算出会话密钥，破解所有密文。

这就是所谓的“**今日截获，明日破解**”。

而 ECDHE 算法在每次握手时都会生成一对临时的公钥和私钥，每次通信的密钥对都是不同的，也就是“一次一密”，即使黑客花大力气破解了这一次的会话密钥，也只是这次通信被攻击，之前的历史消息不会受到影响，仍然是安全的。

所以现在主流的服务器和浏览器在握手阶段都已经不再使用 RSA，改用 ECDHE，而 TLS1.3 在协议里明确废除 RSA 和 DH 则在标准层面保证了“前向安全”。



提升性能

HTTPS 建立连接时除了要做 TCP 握手，还要做 TLS 握手，在 1.2 中会多花两个消息往返（2-RTT），可能导致几十毫秒甚至上百毫秒的延迟，在移动网络中延迟还会更严重。

现在因为密码套件大幅度简化，也就没有必要再像以前那样走复杂的协商流程了。TLS1.3 压缩了以前的“Hello”协商过程，删除了“Key Exchange”消息，把握手时间减少到了“1-RTT”，效率提高了一倍。

那么它是怎么做到的呢？

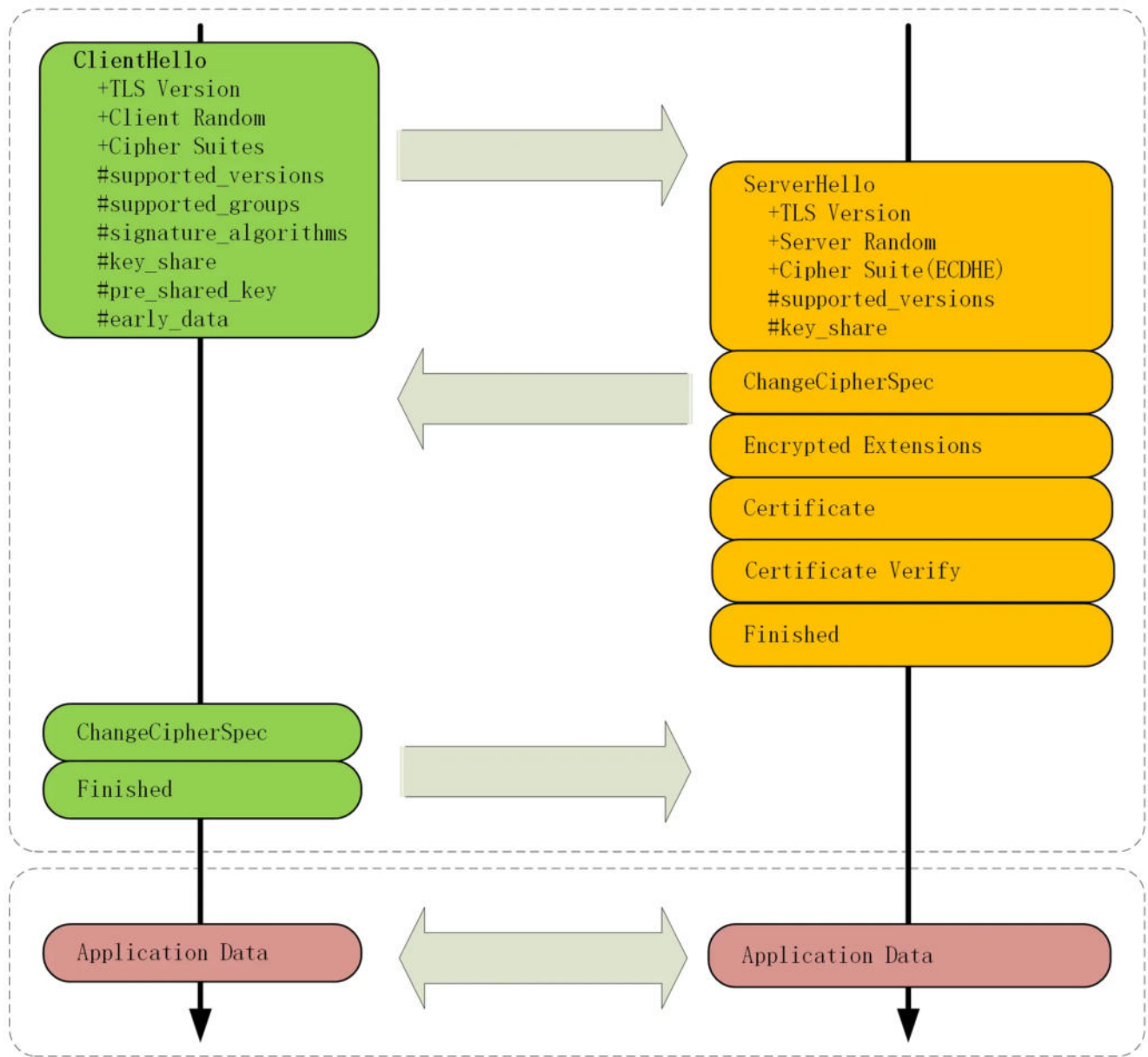
其实具体的做法还是利用了扩展。客户端在“Client Hello”消息里直接用“**supported_groups**”带上支持的曲线，比如 P-256、x25519，用“**key_share**”带上曲线对应的客户端公钥参数，用“**signature_algorithms**”带上签名算法。

服务器收到后在这些扩展里选定一个曲线和参数，再用“key_share”扩展返回服务器这边的公钥参数，就实现了双方的密钥交换，后面的流程就和 1.2 基本一样了。

我为 1.3 的握手过程画了一张图，你可以对比 1.2 看看区别在哪里。

领资料





除了标准的“1-RTT”握手，TLS1.3 还引入了“0-RTT”握手，用“pre_shared_key”和“early_data”扩展，在 TCP 连接后立即就建立安全连接发送加密消息，不过这需要有一些前提条件，今天暂且不说。

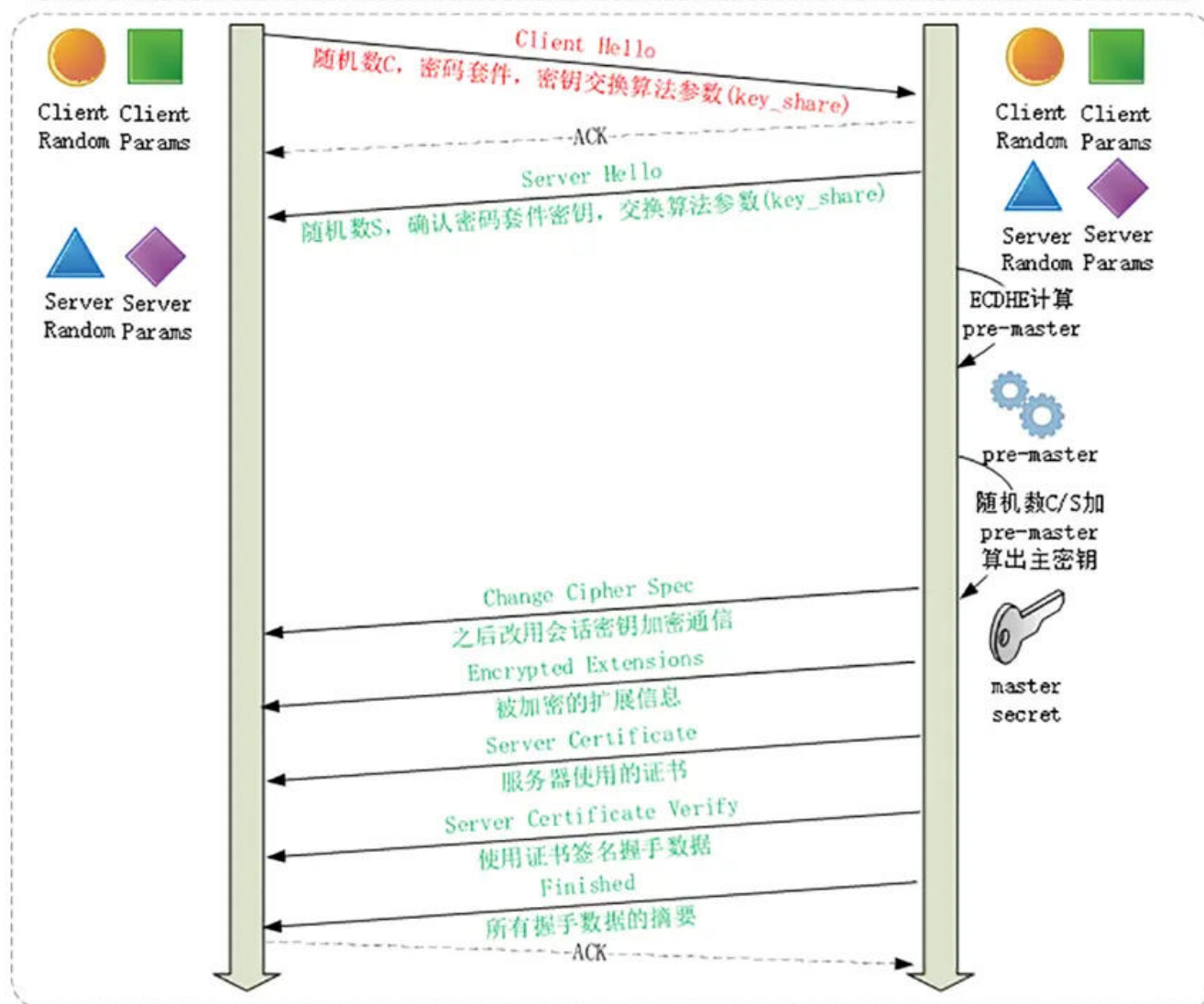
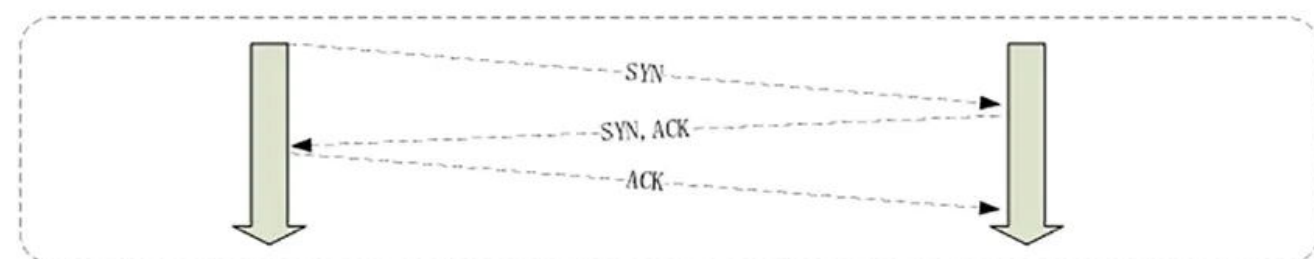
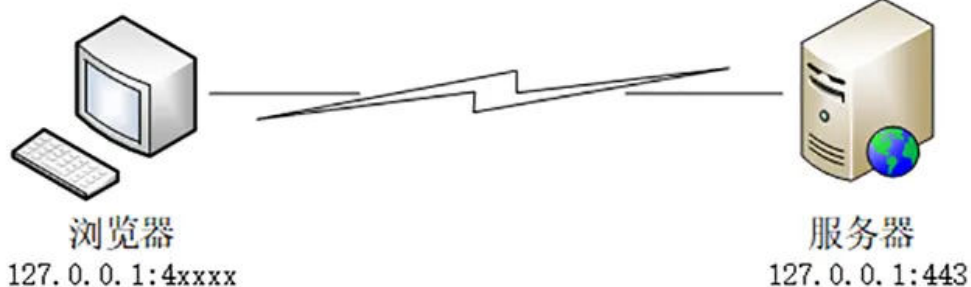
握手分析

目前 Nginx 等 Web 服务器都能够很好地支持 TLS1.3，但要求底层的 OpenSSL 必须是 1.1.1，而我们实验环境里用的 OpenSSL 是 1.1.0，所以暂时无法直接测试 TLS1.3。

不过我在 Linux 上用 OpenSSL1.1.1 编译了一个支持 TLS1.3 的 Nginx，用 Wireshark 抓包存到了 GitHub 上，用它就可以分析 TLS1.3 的握手过程。

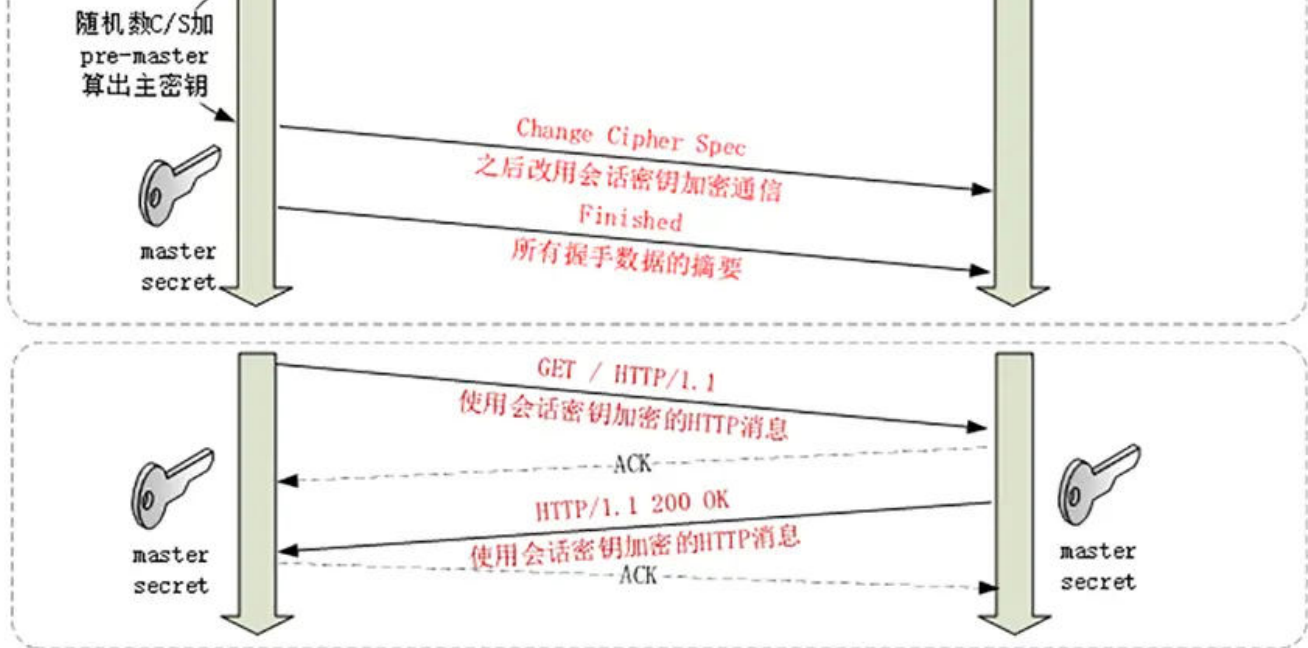
领资料





领资料





在 TCP 建立连接之后，浏览器首先还是发一个“Client Hello”。

因为 1.3 的消息兼容 1.2，所以开头的版本号、支持的密码套件和随机数（Client Random）结构都是一样的（不过这时的随机数是 32 个字节）。

复制代码

```

1 Handshake Protocol: Client Hello
2   Version: TLS 1.2 (0x0303)
3   Random: ceb6c05403654d66c2329...
4   Cipher Suites (18 suites)
5     Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
6     Cipher Suite: TLS_CHACHA20_POLY1305_SHA256 (0x1303)
7     Cipher Suite: TLS_AES_256_GCM_SHA384 (0x1302)
8   Extension: supported_versions (len=9)
9     Supported Version: TLS 1.3 (0x0304)
10    Supported Version: TLS 1.2 (0x0303)
11   Extension: supported_groups (len=14)
12     Supported Groups (6 groups)
13       Supported Group: x25519 (0x001d)
14       Supported Group: secp256r1 (0x0017)
15   Extension: key_share (len=107)
16     Key Share extension
17       Client Key Share Length: 105
18       Key Share Entry: Group: x25519
19       Key Share Entry: Group: secp256r1

```

领资料




注意“Client Hello”里的扩展，“**supported_versions**”表示这是 TLS1.3，“**supported_groups**”是支持的曲线，“**key_share**”是曲线对应的参数。

这就好像是说：

“还是照老规矩打招呼，这边有这些这些信息。但我猜你可能会升级，所以再多给你一些东西，也许后面用的上，咱们有话尽量一口气说完。”

服务器收到“Client Hello”同样返回“Server Hello”消息，还是要给出一个**随机数**（Server Random）和选定密码套件。

 复制代码

```
1 Handshake Protocol: Server Hello
2   Version: TLS 1.2 (0x0303)
3   Random: 12d2bce6568b063d3dee2...
4   Cipher Suite: TLS_AES_128_GCM_SHA256 (0x1301)
5   Extension: supported_versions (len=2)
6     Supported Version: TLS 1.3 (0x0304)
7   Extension: key_share (len=36)
8     Key Share extension
9       Key Share Entry: Group: x25519, Key Exchange length: 32
```

表面上看和 TLS1.2 是一样的，重点是后面的扩展。“**supported_versions**”里确认使用的是 TLS1.3，然后在“**key_share**”扩展带上曲线和对应的公钥参数。

服务器的“Hello”消息大概是这个意思：

“还真让你给猜对了，虽然还是按老规矩打招呼，但咱们来个‘旧瓶装新酒’。刚才你给的我都用上了，我再给几个你缺的参数，这次加密就这么定了。”

这时只交换了两条消息，客户端和服务端就拿到了四个共享信息：**Client Random** 和 **Server Random**、**Client Params** 和 **Server Params**，两边就可以各自用 ECDHE 算出“**Pre-Master**”，再用 HKDF 生成主密钥“**Master Secret**”，效率比 TLS1.2 提高了一大截。

在算出主密钥后，服务器立刻发出“**Change Cipher Spec**”消息，比 TLS1.2 提早进入加密通信，后面的证书等就都是加密的了，减少了握手时的明文信息泄露。

领资料

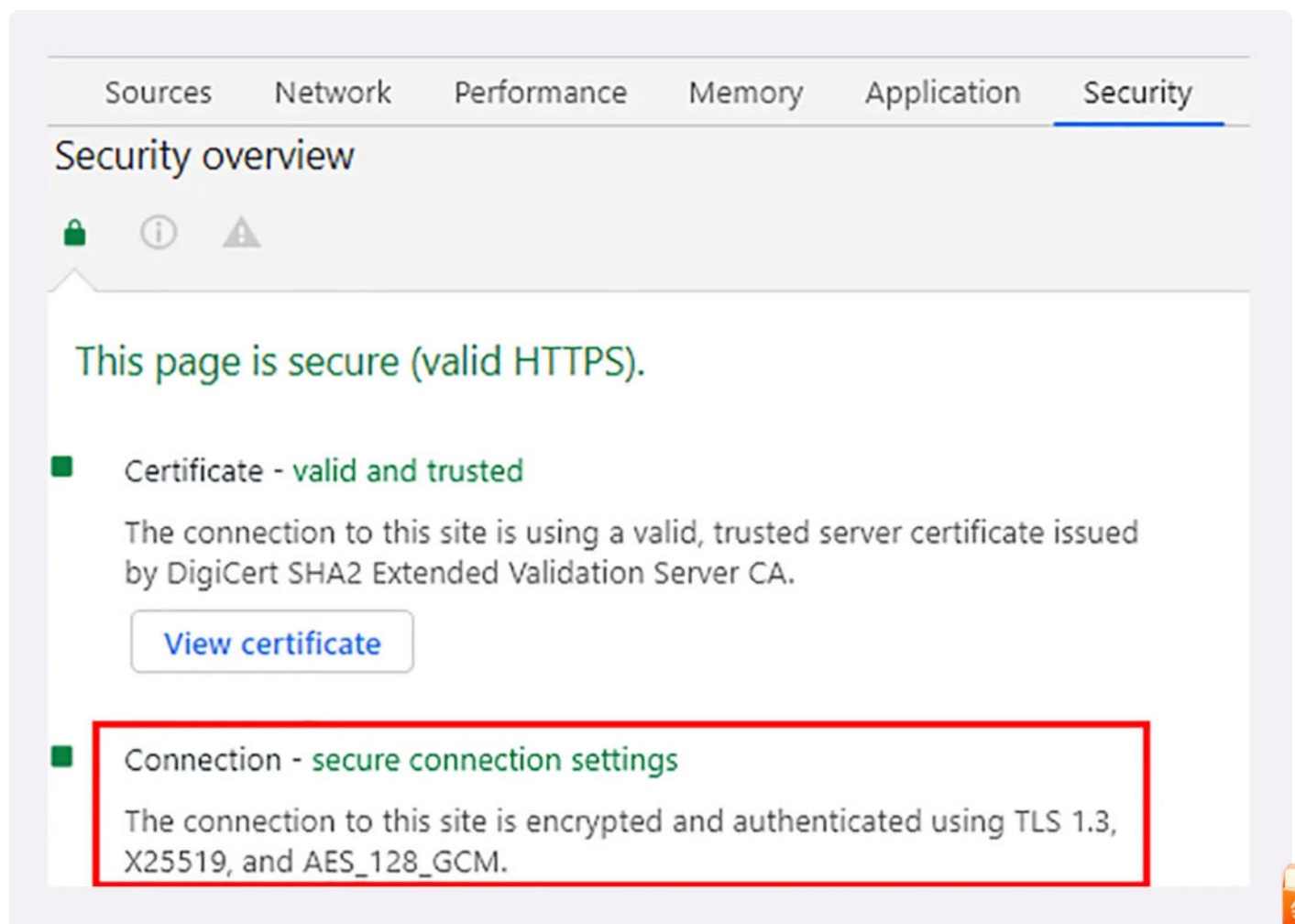


这里 TLS1.3 还有一个安全强化措施，多了个“**Certificate Verify**”消息，用服务器的私钥把前面的曲线、套件、参数等握手数据加了签名，作用和“**Finished**”消息差不多。但由于是私钥签名，所以强化了身份认证和防篡改。

这两个“Hello”消息之后，客户端验证服务器证书，再发“Finished”消息，就正式完成了握手，开始收发 HTTP 报文。

虽然我们的实验环境暂时不能抓包测试 TLS1.3，但互联网上很多网站都已经支持了 TLS1.3，比如 [Nginx](#)、[GitHub](#)，你可以课后自己用 Wireshark 试试。

在 Chrome 的开发者工具里，可以看到这些网站的 TLS1.3 应用情况。



小结

今天我们一起学习了 TLS1.3 的新特性，用抓包研究了它的握手过程，不过 TLS1.3 里的内容很多，还有一些特性没有谈到，后面会继续讲。

领资料



1. 为了兼容 1.1、1.2 等“老”协议，TLS1.3 会“伪装”成 TLS1.2，新特性在“扩展”里实现；
2. 1.1、1.2 在实践中发现了很多安全隐患，所以 TLS1.3 大幅度删减了加密算法，只保留了 ECDHE、AES、ChaCha20、SHA-2 等极少数算法，强化了安全；
3. TLS1.3 也简化了握手过程，完全握手只需要一个消息往返，提升了性能。

课下作业

1. TLS1.3 里的密码套件没有指定密钥交换算法和签名算法，那么在握手的时候会不会有问题呢？
2. 结合上一讲的 RSA 握手过程，解释一下为什么 RSA 密钥交换不具有“前向安全”。
3. TLS1.3 的握手过程与 TLS1.2 的“False Start”有什么异同？

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 对 TLS1.2 已知的攻击有 BEAST、BREACH、CRIME、FREAK、LUCKY13、POODLE、ROBOT 等。
- 02 虽然 TLS1.3 到今天刚满一岁，但由于有之前多个草案的实践，各大浏览器和服务器基本都已经实现了支持，整个互联网也正在快速向 TLS1.3 迁移。


领资料



- 03 关于“前向安全”最著名的案例就是斯诺登于2013年爆出的“棱镜计划”。
- 04 在 TLS1.3 的 RFC 文档里已经删除了“Change Cipher Spec”子协议，但用 Wireshark 抓包却还能看到，这里以抓包为准。
- 05 TLS1.3 还提供了“降级保护机制”，如果“中间人”恶意降级到 1.2，服务器的随机数最后 8 个字节会被设置为“44 4F 57 4E 47 52 44 01”，即“DOWNGRD01”，支持 TLS1.3 的客户端就可以检查发现被降级，然后发出警报终止连接。

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



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 (30)

写留言



djfhchdh

2019-07-29

- 1、TLS1.3精简了加密算法，通过support_groups、key_share、signature_algorithms这些参数就能判断出密钥交换算法和签名算法，不用在cipher suite中协商了
- 2、RSA握手时，client key exchange会使用RSA公钥加密pre master后传给服务端，一旦私钥被破解，那么之前的信息都会被破译，根本原因还是在于RSA的这一对公钥私钥并不是临时的。
- 3、相同点：都在未收到Finished确认消息时就已经向对方发送加密信息了，不同点：TLS1.3将change cipher spec合并到了hello中

作者回复: great。

共 2 条评论 >

39

领资料



Fstar

2019-07-31

2. 结合上一讲的 RSA 握手过程，解释一下为什么 RSA 密钥交换不具有“前向安全”。



答：RSA 握手中，Server Hello 后，客户端拿到服务器的证书，从中提取出服务器的公钥，然后用这个公钥去加密客户端生成的一个随机数（会话密钥）得到密文，然后将其返回给服务器。虽然每次 TLS 握手会话的会话密钥都是不一样的，但服务器的私钥却始终不会变。一旦黑客拿到了服务器私钥，并且截获了之前的所有密文，就能拿到每次会话中的对称密钥，从而得到客户端和服务器的所有“历史会话记录”。

说到底，RSA 握手下，服务器私钥是不变的，从而导致不具有“前向安全”。而 ECDHE 的私钥却是动态的，黑客拿到了一个，也只能解密一个密文。

作者回复: 回答的非常好。



19



彦页

2019-07-31

老师，客户端验证服务器证书，为什么不是pre_master计算出来才检验证书？因为服务器已经把证书加密传输的啊？

作者回复: 这里有个细节没讲，其实tls1.3有多个加密密钥，在握手的时候，服务器发来的数据会用server_handshake_traffic_secret进行加密，而这个密钥也是由HKDF算出来的。

所以客户端先生成server_handshake_traffic_secret，把服务器握手消息解密，取出证书，验证证书，都没问题，才计算pre-master。



4



李鑫磊

2019-12-06

请教老师，一直没看懂，“密钥交换算法参数”究竟是什么？

作者回复: 可以参考“答疑篇”，讲ecdhe算法，参数就是算法的公钥、曲线定义。



3

领资料



Geek_534344

2021-04-06

TLS1.3中，服务端的证书有什么作用呢？

作者回复: 和TLS1.2的作用是一样的，验证服务器身份，对握手数据签名。

1.3只是简化了1.2的握手步骤，基本原理还是一样的。





2



俊伟

2020-01-17

1.1.3在握手时指定了supported groups和key share和signature algorithms, 服务器从这些参数中就能判断出密钥交互算法和摘要算法。

2.因为RSA是客户端算出pre master发送到服务端, 算出来的master secret是固定的, 随着时间的推移, 有被黑客算出来的风险。

3.TLS1.2是客户端率先算出master secret, 然后发送Application data 而TLS1.3是服务端优先算出master secret, 发送Application data。

作者回复: 经过了认真的思考, 回答的非常好!



2



彩色的沙漠

2019-08-05

希望老师对TLS1.3增加一篇补充, 里面涉及的细节大家不是很清楚怎么回事, 网上资料少还容易误导。

1、服务器返回的Encrypted Extensions (被加密的扩展信息), 加密的扩展信息里面不包含key_share和support_groups,这两个关键参数因为加密之后, 无法计算pre-master。问题是加密的扩展信息使用的是哪个密钥对?

2、原文中“在算出主密钥后, 服务器立刻发出“Change Cipher Spec”消息, 比 TLS1.2 提早进入加密通信, 后面的证书等就都是加密的, 减少了握手时的明文信息泄露。问题是, 除了证书还有那些参数使用加密传输, 以及使用的是个密钥对? 客户端不先计算pre-master何master-secret, 怎么解密证书, 进行验证?

3、server certificate verify, 使用证书签名握手数据, Finished也是对握手数据进行摘要签名, 它用的是master-secret进行的签名吗?

作者回复:

这些比较细节和底层, 如果想要认真研究还是建议去看rfc。

我简单解释一下:

1.tls1.3使用了多个对称密钥, 服务器在握手Encrypted Extensions时使用的不是pre-master, 而是server_handshake_traffic_secret。

2.可以参考课程里简略流程图, 里面列出了那些记录, 而具体的扩展字段会因密码套件而变化。

3.Finished消息与tls1.2的一样, 是用会话密钥, 也就是master secret加密的。

领资料





2



Leon

2019-07-30

老师，我对比了下tls1.2和1.3，发现pre master根本就是多余的嘛,双方有个公共K, $A=a*K$ 发给服务端，服务端生成 $B=b*K$,双方就可以利用 $a*b*K=b*a*K$ 的相同密钥进行通信了,1.2是需要客户端把premaster发给服务端，然后双方 $a*b*K*pre\ master=b*a*K*pre\ master$ 的对称密钥进行通信，1.3就是少了pre master，可以这样理解吧

作者回复: 不是的。

还是要有pre-master，注意tls通信使用的master key需要三个随机数生成，其中客户端和服务器的随机数是公开的，而pre-master是加密传输。

tls1.3只是简化了握手过程中的密码套件协商，还是要交换密钥参数算pre-master的。

如果是自己内部的系统，当然可以不用tls那么复杂，直接一个随机数当会话密钥。

可以再看一下tls1.3的握手流程图，里面还有pre-master。

共 3 条评论 >



2



宝仔

2021-03-05

罗老师你好，我们其实也算同事哈哈。我们是被360全资收购的一个创业公司，咨询你一个问题：

客户A有自己的一个域名a.test.com (nginx web，这个nginx web也是我们给搭建的)，访问https://a.test.com会反向代理(proxy_pass https://c.me.com)到我们的站点，c.me.com即为我们的站点。突然有一天客户反馈网站无法访问(问题表象是访问很慢，大概要7s左右，要么就是超时502)，查了很久的问题，各种抓包，最终在客户的nginx上加了proxy_ssl_server_name on;就好了；我们也知道这个参数的含义了，但是我们这边没有做任何变动，包括这家客户的nginx我们也没做任何变动，并且我们的客户不止这么一家，配置都是一样的，其他家都是正常的。难道是客户这边出口做了什么调整吗？老师有解法吗？

作者回复: 先说一下，我已经离开360一段时间了，所以只能是“前同事”。

关于这个问题，从描述上来看，好像与proxy_ssl_server_name on没有什么关系，所以我觉得还是应该从网络环境上找原因，而不是Nginx的问题。

共 2 条评论 >



1

领资料





james

2020-05-30

1. TLS1.3精简了加密算法，通过support_groups、key_share、signature_algorithms这些参数就能判断出密钥交换算法和签名算法，不用在cipher suite中协商了
2. RSA 握手中，Server Hello 后，客户端拿到服务器的证书，从中提取出服务器的公钥，然后用这个公钥去加密客户端生成的一个随机数（会话密钥）得到密文，然后将其返回给服务器。虽然每次 TLS 握手中的会话密钥都是不一样的，但服务器的私钥却始终不会变。一旦黑客拿到了服务器私钥，并且截获了之前的所有密文，就能拿到每次会话中的对称密钥，从而得到客户端和服务器的所有“历史会话记录”。

说到底，RSA 握手下，服务器私钥是不变的，从而导致不具有“前向安全”。而 ECDHE 的私钥却是动态的，黑客拿到了一个，也只能解密一个密文。

3. 相同点：都在未收到Finished确认消息时就已经向对方发送加密信息了，不同点：TLS1.3将change cipher spec合并到了hello中

作者回复: 说的很好。



1



饭粒

2020-05-04

Client Hello 数据中密钥参数 key_share 有两个 Key Share Entry，服务端回复只返回了一个，这个应该是和密钥套件（5选1）一样是需要服务端确定具体使用哪个？

Extension: key_share (len=107)

Type: key_share (51)

Length: 107

Key Share extension

Client Key Share Length: 105

Key Share Entry: Group: x25519, Key Exchange length: 32

Key Share Entry: Group: secp256r1, Key Exchange length: 65

作者回复: 是的，客户端预先给出一些算法参数，让服务器选定，这样就省去了TLS1.2的套件协商过程。

比如这里就是两个椭圆曲线，x25519和P-256，服务器可以从中挑一个，目前大多数都使用最快最安全的X25519。



1

领资料



book尾汁

1. TLS1.3版本在client hello时，已经指定了摘要算法，列出了所支持的椭圆曲线、基点信息。
2. RSA做密钥交换算法时，采用的是证书里面公钥对应的私钥来加密会话密钥，一般私钥被破解，就可以得到之前截获信息的会话密钥，解密消息
- 3 相同点，都是未收到响应前就把 客户端就把密钥交换算法需要的参数传了过去
不同点，因为服务器尚未选择密钥交换算法，因此密钥交换算法的参数有多个，false start中是客户端先得到椭圆曲线的两个参数，算出master key并发起 change cipher spec，而TLS1.3则是服务器先得到这两个参数，发起change cipher spec, TLS1.3change cipher spec合并到了clienthello中

作者回复:

1.tls1.3用扩展协议列出了支持的密钥交换算法和签名算法，所以不需要用密码套件指定。

2.对。

3.对。



1



Geek_1760ca

2021-10-15

小贴士中的降级保护机制真的有用么？中间人本来就既是服务端又是客户端。是不是可以发现提示的随机数再改掉？

作者回复: 可以改，但改之后随机数就和服务器的对不上了，无法完成握手。



Unknown element

2021-09-29

老师client param不是和具体采用的密码套件有关吗 那tls 1.3中客户端是如何在服务器返回采用的密码套件之前把这个参数发给服务器的？还是说客户端把仅有的几个密码套件都生成了一个参数然后都发给服务器让服务器来选？谢谢老师

作者回复: 是的。

因为tls1.3只有很少的几个密码套件，所以客户端就全生成好，发给服务器，然后服务器再挑一个，这样就省去了来回协商的麻烦。

可以再看看抓包数据。

领资料





Geek_68d3d2

2021-09-06

没明白tls1.3如何解决那种今日截获明日破解的问题。私钥黑客就不能自己算？

作者回复: 前向安全与tls1.3无关，跟握手时使用的算法有关。

tls1.3废弃了rsa算法，它不具有前向安全性，而只能使用dhe和ecdhe，所以就具备了前向安全，密钥都是临时生成的，相当于一次一密，黑客破解了也不会对其他会话造成影响。



假于物

2021-08-03

老师，有个地方需要请教下

文中说ECDHE是一次一密，其实RSA也是一次一密吧

RSA因为对称密钥是由客户端决定的，用了服务器的公钥加密；

在服务器私钥被破解，RSA所有的信息都会被破解

作者回复: 一次一密指的是加密用的公私钥是临时生成的。

ecdhe每次都会生成新的公私钥，而rsa总是用服务器的公钥加密私钥解密，不是临时的公钥私钥，所以不是一次一密。

这个就导致了rsa不具有前向安全。



o9

2021-06-04

作者 您好，您在文章里说 TLS1.2需要花费 2 个 RTT，但是我看文章里的图片 里面是有服务端返回的 ACK 包的啊，这个不算吗？ECDHE应该是 3 个消息往返啊

作者回复: 这里说的rtt是指tcp建连后消费的rtt，是tls层面的消息来回，就不考虑tcp的ack了。

tls1.3只用一个rtt就完成了握手，而tls1.2交互过程就麻烦了一点，rsa要用2个rtt完成密钥交换，而ecdhe在第一个rtt之后，第二个消息可以用false start提前发送应用数据，效果和1.3就差不多了。

共 2 条评论 >



领资料





123

2020-12-02

老师能否讲一下，国密tls协议与国际协议的区别？

作者回复: 对国密不是太了解，而且TLS本身就是国际标准，不存在其他版本，我个人理解可能是算法里加入了国密算法吧，但这个就跳出标准的定义了。



BoyiKia

2020-07-03

老师 数字签名，是服务器用私钥加密了摘要。那么 签名算法是指什么，和数字签名有关系吗？

作者回复: 密钥必须要配合算法才能起作用，比如RSA算法要用1024位的私钥，ECDHE算法要用224位的私钥。

只有私钥，它只是一串数字，是无法完成签名动作的，必须要在算法里才能起作用。

就像盖章一样，只有章不行，还得有印泥、盖在正确的地方才能算是签名。



爱学习不害怕

2020-06-22

课程每一篇文章都看了至少两遍，再整理一遍笔记，才能达到初步认识理解的程度。都是干货，对于在校生来说内容很丰富。而且从实践的角度去学习，去掉了很多不需要的枝节，比单纯的读《HTTP权威指南》这样的词典更能抓住重点和本质，喜欢老师的课程

作者回复: 感谢褒奖，今后我们都要持续学习进步。



领资料

