14 | 优化TLS/SSL性能该从何下手?

2020-05-29 陶辉

系统性能调优必知必会 进入课程 >

讲述：陶辉
时长 14:16 大小 13.07M

你好，我是陶辉。

从这一讲开始，我们进入应用层协议的处理。

信息安全在当下越来越重要，绝大多数站点访问时都使用 https:// 替代了 http://，这就是在用 TLS/SSL 协议（下文简称为 TLS 协议）来保障应用层消息的安全。但另一方面，你会发现很多图片类门户网站，还在使用 http://，这是因为 TLS 协议在对信息加解密的同时，必然会降低性能和用户体验，这些站点在权衡后选择了性能优先。

实际上，TLS 协议由一系列加密算法及规范组成，这些算法的安全性和性能各不相同，甚至与你的系统硬件相关。比如当主机的 CPU 支持 AES-NI 指令集时，选择 AES 对称加密算法便可以大幅提升性能。然而，要想选择合适的算法，需要了解算法所用到的一些数学知识，而很多同学由于忽视了数学原理便难以正确地配置 TLS 算法。

同时，TLS 协议优化时也需要了解网络和软件工程知识，比如我们可以在网络的不同位置缓存密钥来优化性能。而且，TLS 协议还可以优化其他应用层协议的性能，比如从 HTTP/1 升级到 HTTP/2 协议便可以通过 TLS 协议减少 1 个 RTT 的时间。

优化 TLS 性能究竟该从何下手呢？在我看来主要有两个方向，一是对称加密算法的性能优化，二是如何高效地协商密钥。下面我们来详细看看优化细节。

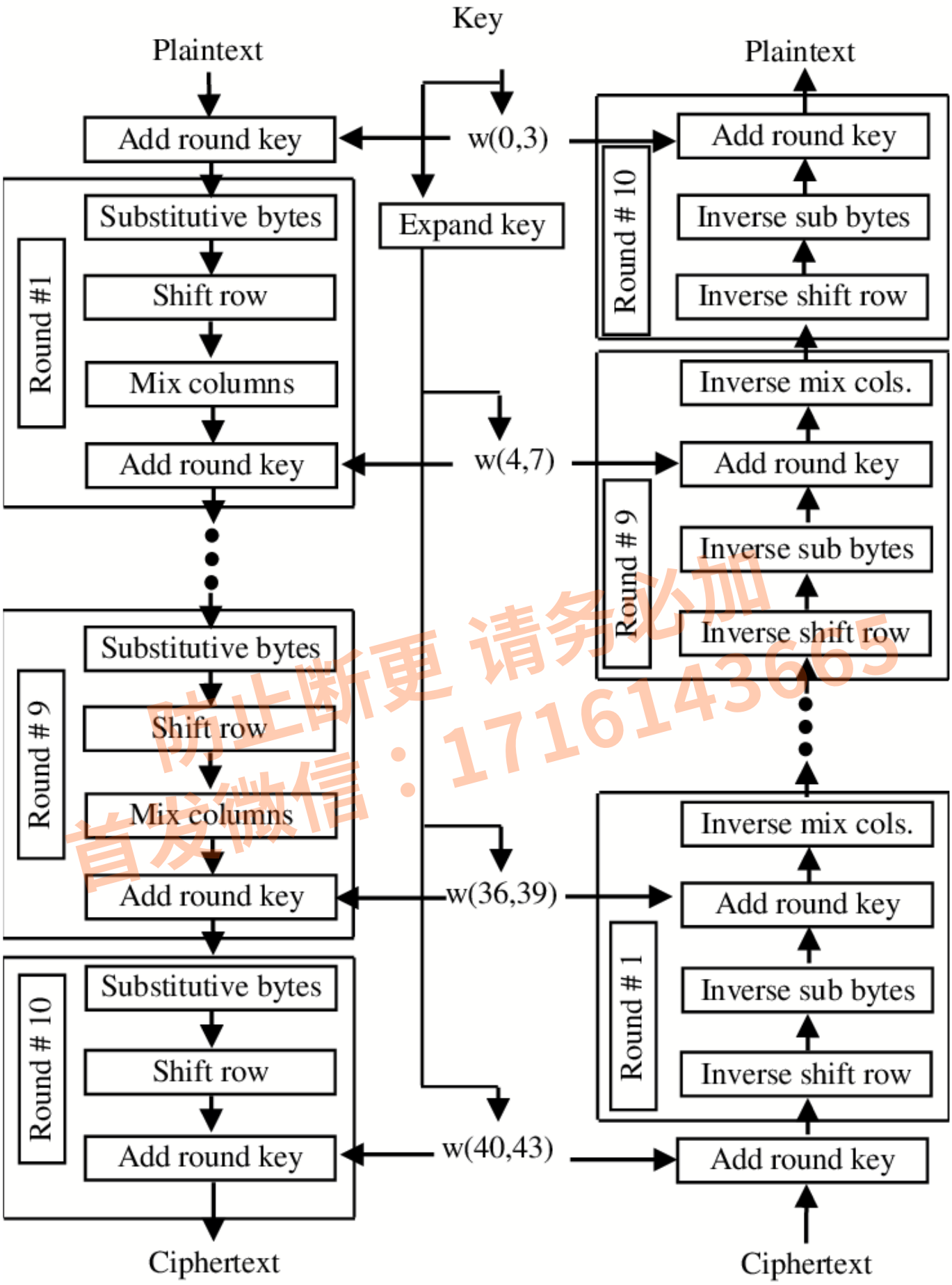
如何提升对称加密算法的性能？

如果你用 Wireshark 等工具对 HTTPS 请求抓包分析，会发现在 TCP 传输层之上的消息全是乱码，这是因为 TCP 之上的 TLS 层，把 HTTP 请求用对称加密算法重新进行了编码。**当然，用 Chrome 浏览器配合 Wireshark 可以解密消息，帮助你分析 TLS 协议的细节**（具体操作方法可参考《Web 协议详解与抓包实战》第 51 课）。

现代对称加密算法的特点是，即使把加密流程向全社会公开，攻击者也从公网上截获到密文，但只要他没有拿到密钥，就无法从密文中反推出原始明文。如何同步密钥我们稍后在谈，先来看如何优化对称加密算法。

目前主流的对称加密算法叫做 AES（Advanced Encryption Standard），它在性能和安全上表现都很优秀。而且，它不只在访问网站时最为常用，甚至你日常使用的 WINRAR 等压缩软件也在使用 AES 算法（见官方 FAQ）。**因此，AES 是我们的首选对称加密算法，**下面来看看 AES 算法该如何优化。

AES 只支持 3 种不同的密钥长度，分别是 128 位、192 位和 256 位，它们的安全性依次升高，运算时间也更长。比如，当密钥为 128 比特位时，需要经过十轮操作，其中每轮要用移位法、替换法、异或操作等对明文做 4 次变换。而当密钥是 192 位时，则要经过 12 轮操作，密钥为 256 比特位时，则要经过 14 轮操作，如下图所示。

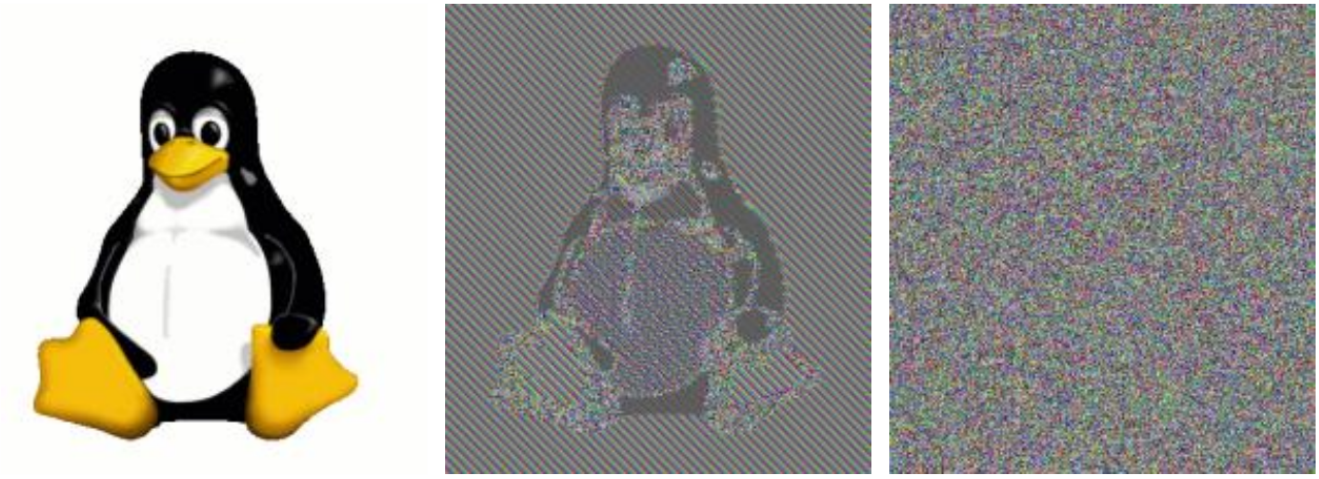


AES128的10轮加密流程

此图由Ahmed Ghanim Wadday上传于www.researchgate.net

密钥越长，虽然性能略有下降，但安全性提升很多。比如早先的 DES 算法只有 56 位密钥，在 1999 年便被破解。**在 TLS1.2 及更早的版本中，仍然允许通讯双方使用 DES 算法，这是非常不安全的行为，你应该在服务器上限制 DES 算法套件的使用**（Nginx 上限制加密套件的方法，参见《Nginx 核心知识 100 讲》第 96 课 和第 131 课）。也正因为密钥长度对安全性的巨大影响，美国政府才不允许出口 256 位密钥的 AES 算法。

只有数百比特的密钥，到底该如何对任意长度的明文加密呢？主流对称算法会将原始明文分成等长的多组明文，再分别用密钥生成密文，最后把它们拼接在一起形成最终密文。而 AES 算法是按照 128 比特（16 字节）对明文进行分组的（最后一组不足 128 位时会填充 0 或者随机数）。为了防止分组后密文出现明显的规律，造成攻击者容易根据概率破解出原文，我们就需要对每组的密钥做一些变换，**这种分组后变换密钥的算法就叫做分组密码工作模式（下文简称为分组模式），它是影响 AES 性能的另一个因素。**



原图　　使用ECB模式加密　　提供了伪随机性的非ECB模式

优秀的分组密码工作模式
更难以从密文中发现规律，图参见wiki

比如，CBC 分组模式中，只有第 1 组明文加密完成后，才能对第 2 组加密，因为第 2 组加密时会用到第 1 组生成的密文。因此，CBC 必然无法并行计算。在材料科学出现瓶颈、单核频率不再提升的当下，CPU 都在向多核方向发展，而 CBC 分组模式无法使用多核的并行计算能力，性能受到很大影响。**所以，通常我们应选择可以并行计算的 GCM 分组模式，这也是当下互联网中最常见的 AES 分组算法。**

由于 AES 算法中的替换法、行移位等流程对 CPU 指令并不友好，所以 Intel 在 2008 年推出了支持AES-NI 指令集的 CPU，能够将 AES 算法的执行速度从每字节消耗 28 个时钟

周期（参见这里），降低至 3.5 个时钟周期（参见这里）。在 Linux 上你可以用下面这行命令查看 CPU 是否支持 AES-NI 指令集：

```
# sort -u /proc/crypto | grep module |grep aes
module       : aesni_intel
```

因此，如果 CPU 支持 AES-NI 特性，那么应选择 AES 算法，否则可以选择 CHACHA20 对称加密算法，它主要使用 ARX 操作（add-rotate-xor），CPU 执行起来更快。

说完对称加密算法的优化，我们再来看加密时的密钥是如何传递的。

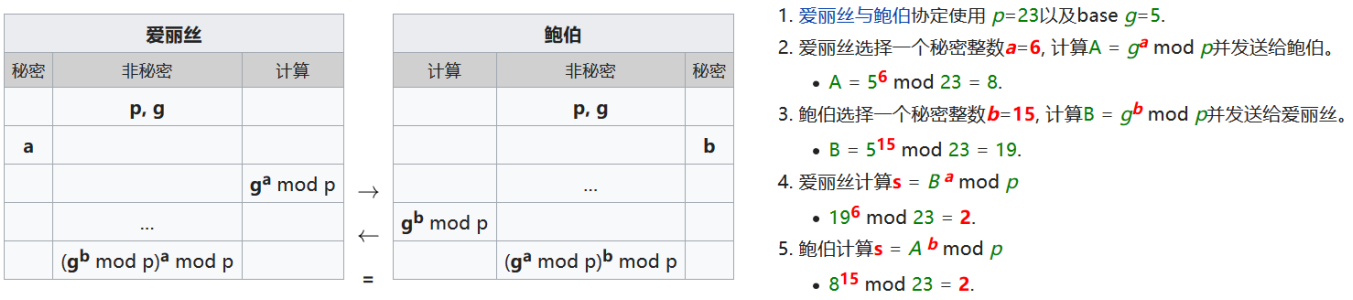
如何更快地协商出密钥？

无论对称加密算法有多么安全，一旦密钥被泄露，信息安全就是一纸空谈。所以，TLS 建立会话的第 1 个步骤是在握手阶段协商出密钥。

早期解决密钥传递的是 RSA 密钥协商算法。当你部署 TLS 证书到服务器上时，证书文件中包含一对公私钥（参见非对称加密），其中，公钥会在握手阶段传递给客户端。在 RSA 密钥协商算法中，客户端会生成随机密钥（事实上是生成密钥的种子参数），并使用服务器的公钥加密后再传给服务器。根据非对称加密算法，公钥加密的消息仅能通过私钥解密，这样服务器解密后，双方就得到了相同的密钥，再用它加密应用消息。

RSA 密钥协商算法的最大问题是不支持前向保密（Forward Secrecy），一旦服务器的私钥泄露，过去被攻击者截获的所有 TLS 通讯密文都会被破解。解决前向保密的是 DH（Diffie–Hellman）密钥协商算法。

我们简单看下 DH 算法的工作流程。通讯双方各自独立生成随机的数字作为私钥，而后依据公开的算法计算出各自的公钥，并通过未加密的 TLS 握手发给对方。接着，根据对方的公钥和自己的私钥，双方各自独立运算后能够获得相同的数字，这就可以作为后续对称加密时使用的密钥。**即使攻击者截获到明文传递的公钥，查询到公开的 DH 计算公式后，在不知道私钥的情况下，也是无法计算出密钥的。**这样，DH 算法就可以在握手阶段生成随机的新密钥，实现前向保密。



DH 算法的计算速度很慢，如上图所示，计算公钥以及最终的密钥时，需要做大量的乘法运算，而且为了保障安全性，这些数字的位数都很长。为了提升 DH 密钥交换算法的性能，诞生了当下广为使用的ECDH 密钥交换算法，**ECDH 在 DH 算法的基础上利用ECC 椭圆曲线特性，可以用更少的计算量计算出公钥以及最终的密钥。**

依据解析几何，椭圆曲线实际对应一个函数，而不同的曲线便有不同的函数表达式，目前不被任何已知专利覆盖的最快椭圆曲线是X25519 曲线，它的表达式是 $y^2 = x^3 + 486662x^2 + x$。因此，当通讯双方协商使用 X25519 曲线用于 ECDH 算法时，只需要传递 X25519 这个字符串即可。在 Nginx 上，你可以使用 ssl_ecdh_curve 指令配置想使用的曲线：

```
ssl_ecdh_curve X25519:secp384r1;
```

选择密钥协商算法是通过 ssl_ciphers 指令完成的：

```
ssl_ciphers 'EECDH+ECDSA+AES128+SHA:RSA+AES128+SHA';
```

可见，ssl_ciphers 可以同时配置对称加密算法及密钥强度等信息。注意，当 ssl_prefer_server_ciphers 设置为 on 时，ssl_ciphers 指定的多个算法是有优先顺序的，**我们应当把性能最快且最安全的算法放在最前面。**

提升密钥协商速度的另一个思路，是减少密钥协商的次数，主要包括以下 3 种方式。

首先，最为简单有效的方式是在一个 TLS 会话中传输多组请求，对于 HTTP 协议而言就是使用长连接，在请求中加入 Connection: keep-alive 头部便可以做到。

其次，客户端与服务器在首次会话结束后缓存下 session 密钥，并用唯一的 session ID 作为标识。这样，下一次握手时，客户端只要把 session ID 传给服务器，且服务器在缓存中找到密钥后（为了提升安全性，缓存会定期失效），双方就可以加密通讯了。这种方式的问题在于，当 N 台服务器通过负载均衡提供 TLS 服务时，客户端命中上次访问过的服务器的概率只有 1/N，所以大概率它们还得再次协商密钥。

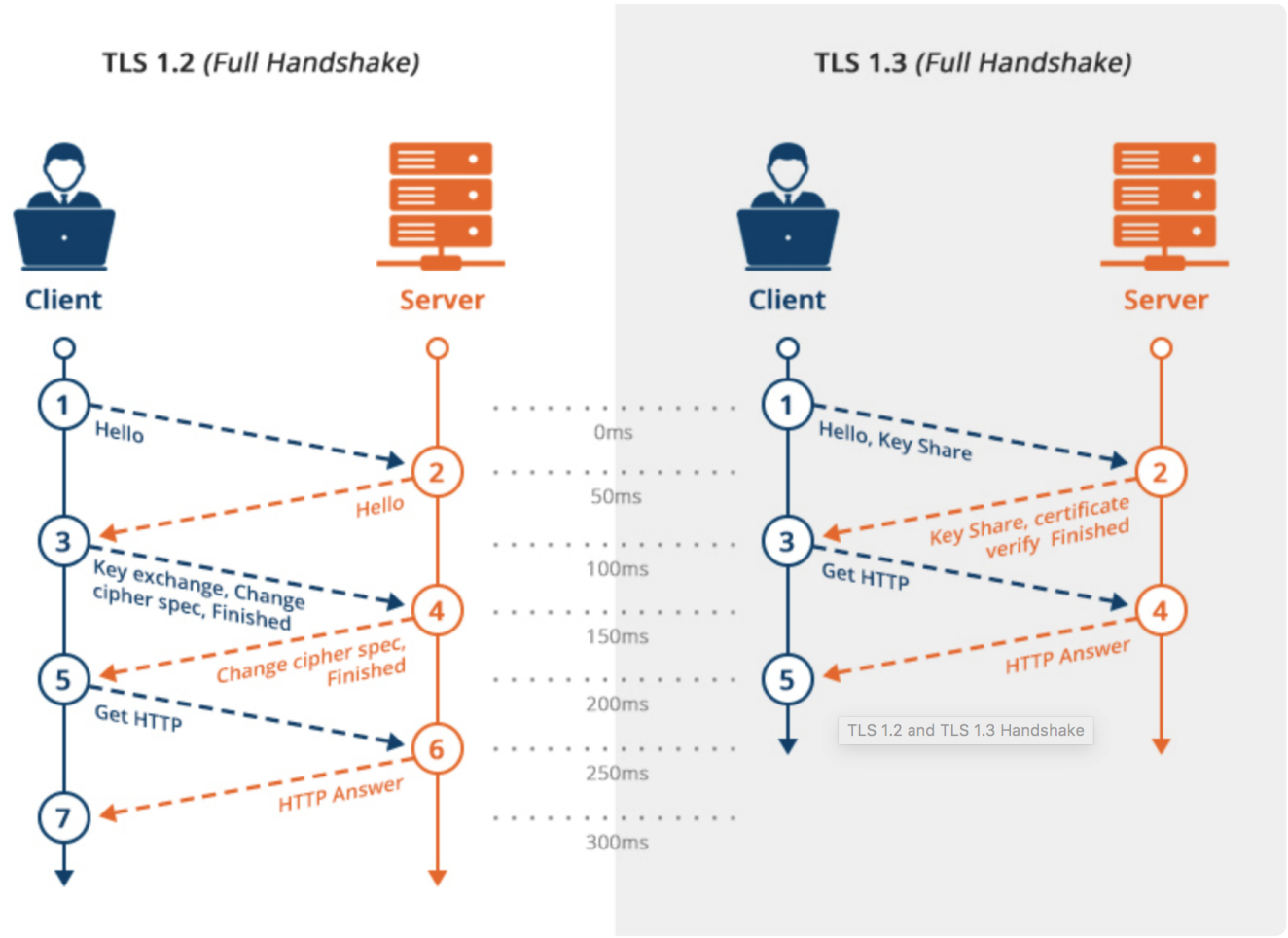
session ticket 方案可以解决上述问题，它把服务器缓存密钥，改为由服务器把密钥加密后作为 ticket 票据发给客户端，由客户端缓存密文。其中，集群中每台服务器对 session 加密的密钥必须相同，这样，客户端携带 ticket 密文访问任意一台服务器时，都能通过解密 ticket，获取到密钥。

当然，使用 session 缓存或者 session ticket 既没有前向安全性，应对重放攻击也更加困难。提升 TLS 握手性能的更好方式，是把 TLS 协议升级到 1.3 版本。

为什么应当尽快升级到 TLS1.3?

TLS1.3（参见RFC8446）对性能的最大提升，在于它把 TLS 握手时间从 2 个 RTT 降为 1 个 RTT。

在 TLS1.2 的握手中，先要通过 Client Hello 和 Server Hello 消息协商出后续使用的加密算法，再互相交换公钥并计算出最终密钥。**TLS1.3 中把 Hello 消息和公钥交换合并为一步，这就减少了一半的握手时间**，如下图所示：



TLS1.3相对TLS1.2，减少了1个RTT的握手时间
图片来自www.ssl2buy.com

那 TLS1.3 握手为什么只需要 1 个 RTT 就可以完成呢？因为 TLS1.3 支持的密钥协商算法大幅度减少了，这样，客户端尽可以把常用 DH 算法的公钥计算出来，并与协商加密算法的 HELLO 消息一起发送给服务器，服务器也作同样处理，这样仅用 1 个 RTT 就可以协商出密钥。

而且，TLS1.3 仅支持目前最安全的几个算法，比如 openssl 中仅支持下面 5 种安全套件：

TLS_AES_256_GCM_SHA384

TLS_CHACHA20_POLY1305_SHA256

TLS_AES_128_GCM_SHA256

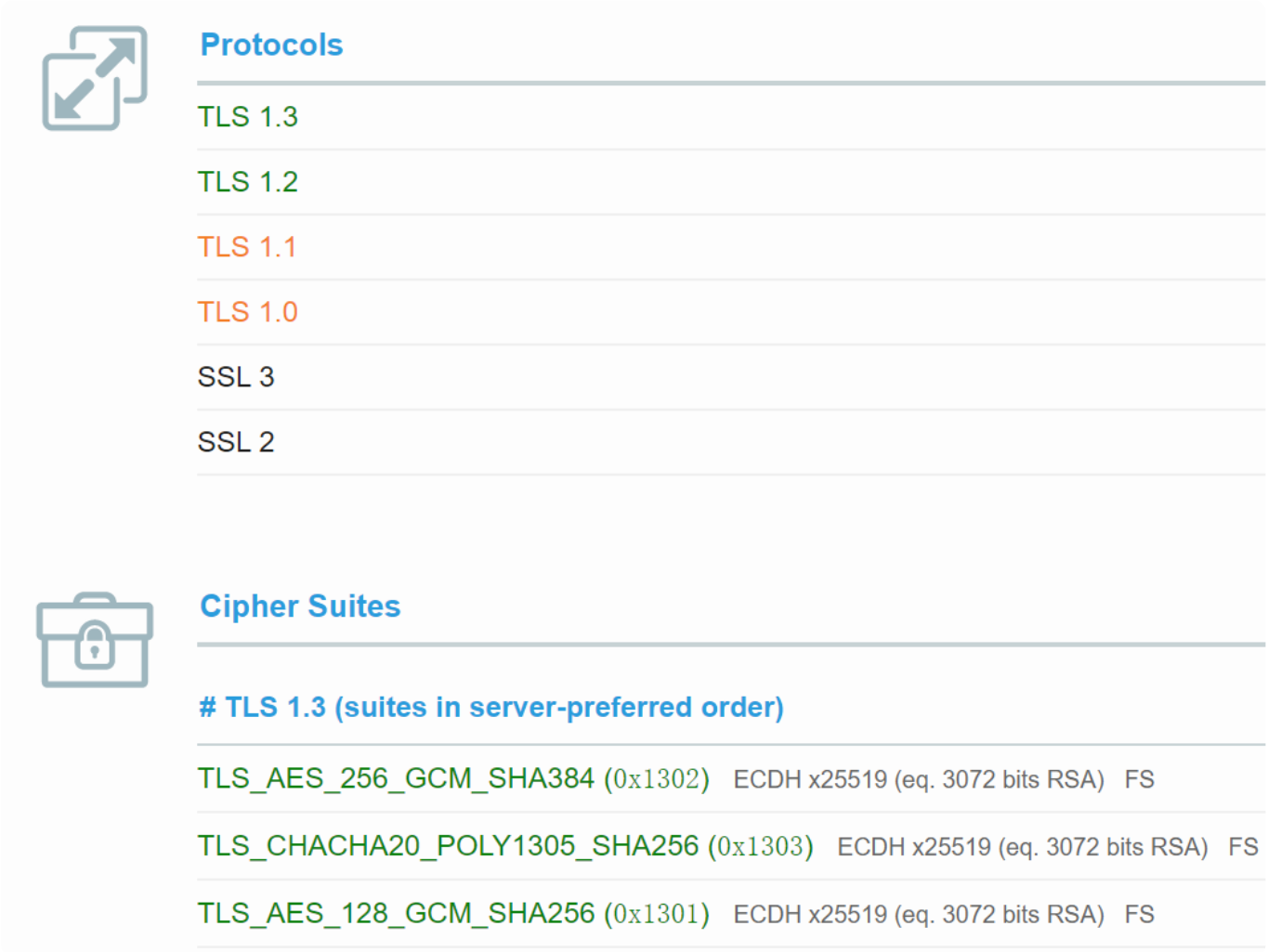
TLS_AES_128_CCM_8_SHA256

TLS_AES_128_CCM_SHA256

相较起来，TLS1.2 支持各种古老的算法，中间人可以利用降级攻击，在握手阶段把加密算法替换为不安全的算法，从而轻松地破解密文。如前文提到过的 DES 算法，由于密钥位

数只有 56 位，很容易破解。

因此，**无论从性能还是安全角度上，你都应该尽快把 TLS 版本升级到 1.3。**你可以用这个网址测试当前站点是否支持 TLS1.3。



如果不支持，还可以参见每日一课《TLS1.3 原理及在 Nginx 上的应用》，升级 Nginx 到 TLS1.3 版本。

小结

这一讲，我们介绍了 TLS 协议的优化方法。

应用消息是通过对称加密算法编码的，而目前 AES 还是最安全的对称加密算法。不同的分组模式也会影响 AES 算法的性能，而 GCM 模式能够充分利用多核 CPU 的并行计算能力，所以 AES_GCM 是我们的首选。当你的 CPU 支持 AES-NI 指令集时，AES 算法的执行会非常快，否则，可以考虑对 CPU 更友好的 CHACHA20 算法。

再来看对称加密算法的密钥是如何传递的，它决定着 TLS 系统的安全，也对 HTTP 小对象的传输速度有很大影响。DH 密钥协商算法速度并不快，因此目前主要使用基于椭圆曲线的 ECDH 密钥协商算法，其中，不被任何专利覆盖的 X25519 椭圆曲线速度最快。为了减少密钥协商次数，我们应当尽量通过长连接来复用会话。在 TLS1.2 及早期版本中，session 缓存和 session ticket 也能减少密钥协商时的计算量，但它们既没有前向安全性，也更难防御重放攻击，所以为了进一步提升性能，应当尽快升级到 TLS1.3。

TLS1.3 将握手时间从 2 个 RTT 降为 1 个 RTT，而且它限制了目前已经不再安全的算法，这样中间人就难以用降级攻击来破解密钥。

密码学的演进越来越快，加密与破解总是在道高一尺、魔高一丈的交替循环中发展，当下安全的算法未必在一年后仍然安全。而且，当量子计算机真正诞生后，它强大的并行计算能力可以轻松地暴力破解当下还算安全的算法。然而，这种划时代的新技术出现时总会有一个时间窗口，而在窗口内也会涌现出能够防御住量子破解的新算法。所以，我们应时常关注密码学的进展，更换更安全、性能也更优秀的新算法。

思考题

最后，留给你一道思考题，TLS 体系中还有许多性能优化点，比如在服务器上部署 OSCP Stapling（用于更快地发现过期证书）也可以提升网站的访问性能，你还用过哪些方式优化 TLS 的性能呢？欢迎你在留言区与我探讨。

感谢阅读，如果你觉得这节课对你有一些启发，也欢迎把它分享给你的朋友。

上一篇　13 | 实战：单机如何实现管理百万主机的心跳服务?

精选留言 (4)

写留言

我来也

2020-05-29

最近就遇到过tls协议版本的问题，不过我们是在开倒车。😂

最近升级到k8s后，默认的ingress nignx只支持tls1.2以上的版本，导致某些安卓5.x版本上的app无法与服务器正常通讯，为了兼容这部分用户，只能强制把tls支持的最低版本号调回到1.0。

展开

1

Geek_007

2020-05-29

TLS1.3 感觉其实也不是优化特别大，TLS1.2 有False Start，也能做到1RTT，至于0RTT，现在主流的CDN应该也还有很多厂家不支持PSK，所以0RTT的效果也不一定好。

课后题：
1、ECC证书应该算是一种优化，因为证书更小，加解密更快。（不过好像因为客户端公钥太长，对客户端不友好，尤其是移动端）...
展开

1

东郭
2020-05-30

请问老师，我在nginx配置中，不管ssl_certificate和ssl_certificate_key是否配置ecc证书，抓包查看服务器的server hello响应中的Cipher Suite字段都是TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256，这是正常的吗？
展开

毛立平
2020-05-29

OSCP->OCSP？最近碰到过letsencrypt在国内访问ocsp的问题导致ios端延迟3s的问题。https://mp.weixin.qq.com/s/z_QsomzE3jBtwi8VdVDdEA
展开

2