

二

24 固若金汤的根本（上）：对称加密与非对称加密

在上一讲中，我们初步学习了 HTTPS，知道 HTTPS 的安全性是由 TLS 来保证的。

你一定很好奇，它是怎么为 HTTP 增加了机密性、完整性，身份认证和不可否认等特性的呢？

先说说机密性。它是信息安全的基础，缺乏机密性 TLS 就会成为“无水之源”“无根之木”。

实现机密性最常用的手段是“**加密**”（encrypt），就是把消息用某种方式转换成谁也看不懂的乱码，只有掌握特殊“**钥匙**”的人才能再转换出原始文本。

这里的“**钥匙**”就叫做“**密钥**”（key），加密前的消息叫“**明文**”（plain text/clear text），加密后的乱码叫“**密文**”（cipher text），使用密钥还原明文的过程叫“**解密**”（decrypt），是加密的反操作，加密解密的操作过程就是“**加密算法**”。

所有的加密算法都是公开的，任何人都可以去分析研究，而算法使用的“**密钥**”则必须保密。那么，这个关键的“**密钥**”又是什么呢？

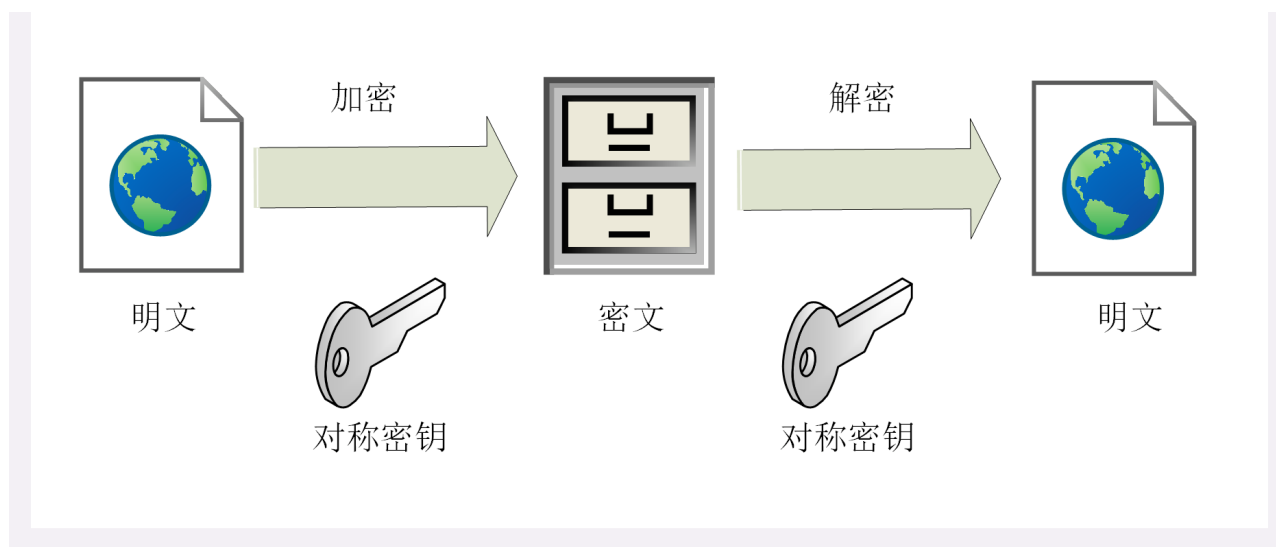
由于 HTTPS、TLS 都运行在计算机上，所以“**密钥**”就是一长串的数字，但约定俗成的度量单位是“**位**”（bit），而不是“**字节**”（byte）。比如，说密钥长度是 128，就是 16 字节的二进制串，密钥长度 1024，就是 128 字节的二进制串。

按照密钥的使用方式，加密可以分为两大类：**对称加密**和**非对称加密**。

对称加密

“**对称加密**”很好理解，就是指加密和解密时使用的密钥都是同一个，是“**对称**”的。只要保证了密钥的安全，那整个通信过程就可以说具有了机密性。

举个例子，你想要登录某网站，只要事先和它约定好使用一个对称密码，通信过程中传输的全是用密钥加密后的密文，只有你和网站才能解密。黑客即使能够窃听，看到的也只是乱码，因为没有密钥无法解出明文，所以就实现了机密性。



TLS 里有非常多的对称加密算法可供选择，比如 RC4、DES、3DES、AES、ChaCha20 等，但前三种算法都被认为是不安全的，通常都禁止使用，目前常用的只有 AES 和 ChaCha20。

AES 的意思是“高级加密标准”（Advanced Encryption Standard），密钥长度可以是 128、192 或 256。它是 DES 算法的替代者，安全强度很高，性能也很好，而且有的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。

ChaCha20 是 Google 设计的另一种加密算法，密钥长度固定为 256 位，纯软件运行性能要超过 AES，曾经在移动客户端上比较流行，但 ARMv8 之后也加入了 AES 硬件优化，所以现在不再具有明显的优势，但仍然算得上是一个不错算法。

加密分组模式

对称算法还有一个“**分组模式**”的概念，它可以让算法用固定长度的密钥加密任意长度的明文，把小秘密（即密钥）转化为大秘密（即密文）。

最早有 ECB、CBC、CFB、OFB 等几种分组模式，但都陆续被发现存在安全漏洞，所以现在基本都不怎么用了。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。

把上面这些组合起来，就可以得到 TLS 密码套件中定义的对称加密算法。

比如，AES128-GCM，意思是密钥长度为 128 位的 AES 算法，使用的分组模式是 GCM；ChaCha20-Poly1305 的意思是 ChaCha20 算法，使用的分组模式是 Poly1305。

你可以用实验环境的 URI“/24-1”来测试 OpenSSL 里的 AES128-CBC，在 URI 后用参数

“key”“plain”输入密钥和明文，服务器会在响应报文里输出加密解密的结果。

<https://www.chrono.com/24-1?key=123456>

```
algo  = aes_128_cbc
plain = hello openssl
enc   = 93a024a94083bc39fb2c2b9f5ce27c09
dec   = hello openssl
```

非对称加密

对称加密看上去好像完美地实现了机密性，但其中有一个很大的问题：如何把密钥安全地传递给对方，术语叫“**密钥交换**”。

因为在对称加密算法中只要持有密钥就可以解密。如果你和网站约定的密钥在传递途中被黑客窃取，那他就可以在之后随意解密收发的数据，通信过程也就没有机密性可言了。

这个问题该怎么解决呢？

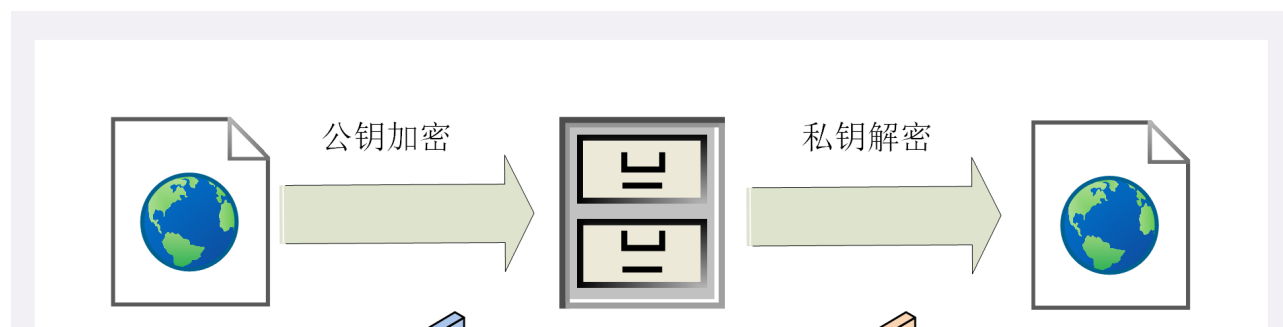
你或许会说：“把密钥再加密一下发过去就好了”，但传输“加密密钥的密钥”又成了新问题。这就像是“鸡生蛋、蛋生鸡”，可以无限递归下去。只用对称加密算法，是绝对无法解决密钥交换的问题的。

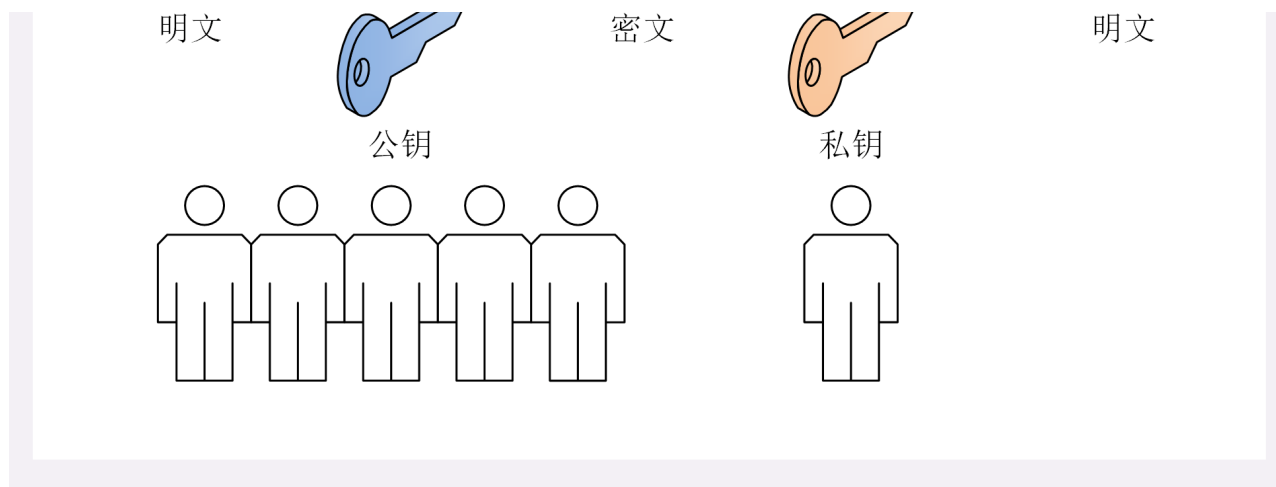
所以，就出现了非对称加密（也叫公钥加密算法）。

它有两个密钥，一个叫“**公钥**”（public key），一个叫“**私钥**”（private key）。两个密钥是不同的，“不对称”，公钥可以公开给任何人使用，而私钥必须严格保密。

公钥和私钥有个特别的“**单向**”性，虽然都可以用来加密解密，但公钥加密后只能用私钥解密，反过来，私钥加密后也只能用公钥解密。

非对称加密可以解决“密钥交换”的问题。网站秘密保管私钥，在网上任意分发公钥，你想要登录网站只要用公钥加密就行了，密文只能由私钥持有者才能解密。而黑客因为没有私钥，所以就无法破解密文。





非对称加密算法的设计要比对称算法难得多，在 TLS 里只有很少的几种，比如 DH、DSA、RSA、ECC 等。

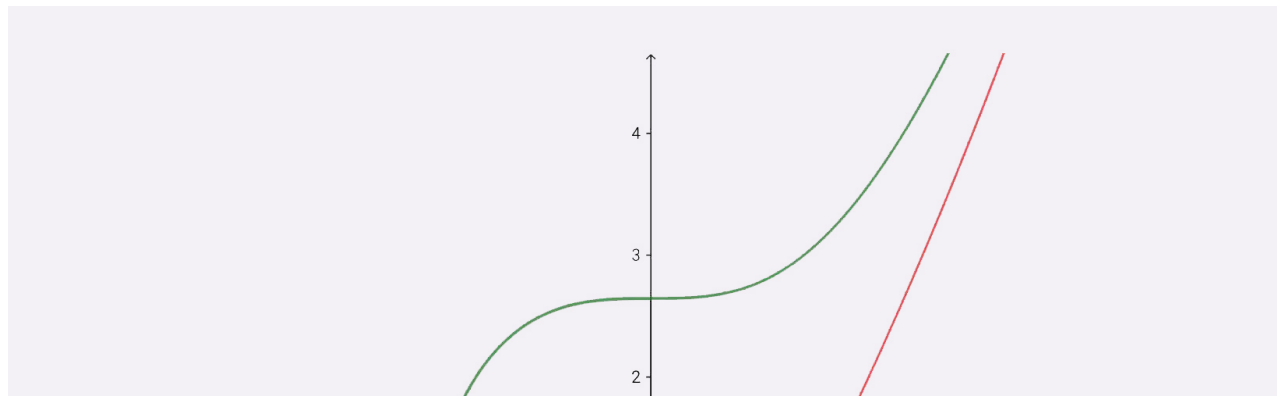
RSA 可能是其中最著名的一个，几乎可以说是非对称加密的代名词，它的安全性基于“**整数分解**”的数学难题，使用两个超大素数的乘积作为生成密钥的材料，想要从公钥推算出私钥是非常困难的。

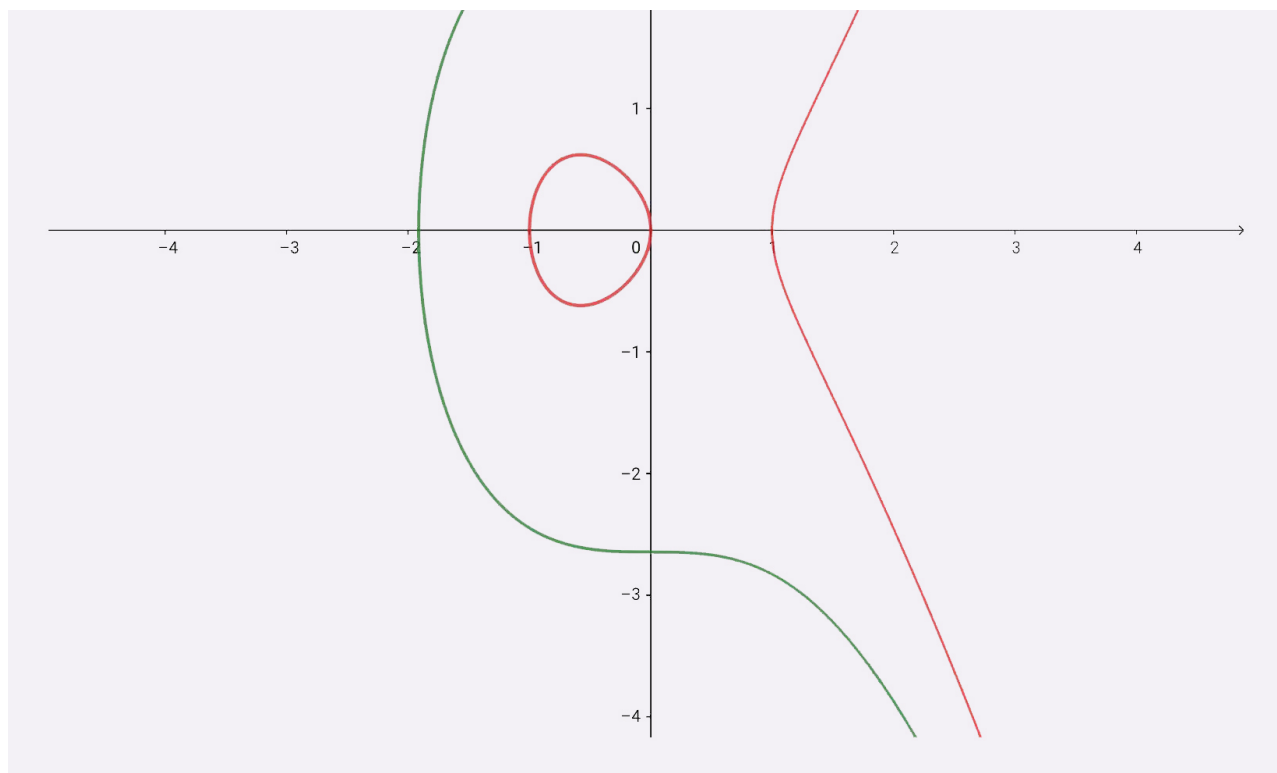
10 年前 RSA 密钥的推荐长度是 1024，但随着计算机运算能力的提高，现在 1024 已经不安全，普遍认为至少要 2048 位。

ECC (Elliptic Curve Cryptography) 是非对称加密里的“后起之秀”，它基于“**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 用于密钥交换，ECDSA 用于数字签名。

目前比较常用的两个曲线是 P-256 (secp256r1，在 OpenSSL 称为 prime256v1) 和 x25519。P-256 是 NIST (美国国家标准技术研究所) 和 NSA (美国国家安全局) 推荐使用的曲线，而 x25519 被认为是最安全、最快速的曲线。

ECC 名字里的“椭圆”经常会引起误解，其实它的曲线并不是椭圆形，只是因为方程很类似计算椭圆周长的公式，实际的形状更像抛物线，比如下面的图就展示了两个简单的椭圆曲线。





两个简单的椭圆曲线： $y^2=x^3+7$ ， $y^2=x^3-x$

比起 RSA，ECC 在安全强度和性能上都有明显的优势。160 位的 ECC 相当于 1024 位的 RSA，而 224 位的 ECC 则相当于 2048 位的 RSA。因为密钥短，所以相应的计算量、消耗的内存和带宽也就少，加密解密的性能就上去了，对于现在的移动互联网非常有吸引力。

实验环境的 URI“/24-2”演示了 RSA1024，你在课后可以动手试一下。

混合加密

看到这里，你是不是认为可以抛弃对称加密，只用非对称加密来实现机密性呢？

很遗憾，虽然非对称加密没有“密钥交换”的问题，但因为它们都是基于复杂的数学难题，运算速度很慢，即使是 ECC 也要比 AES 差上好几个数量级。如果仅用非对称加密，虽然保证了安全，但通信速度有如乌龟、蜗牛，实用性就变成了零。

实验环境的 URI“/24-3”对比了 AES 和 RSA 这两种算法的性能，下面列出了一次测试的结果：

```
aes_128_cbc enc/dec 1000 times : 0.97ms, 13.11MB/s
```

```
rsa_1024 enc/dec 1000 times : 138.59ms, 93.80KB/s  
rsa_1024/aes ratio = 143.17
```

```
rsa_2048 enc/dec 1000 times : 840.35ms, 15.47KB/s
```

```
rsa_2048/aes ratio = 868.13
```

可以看到，RSA 的运算速度是非常慢的，2048 位的加解密大约是 15KB/S（微秒或毫秒级），而 AES128 则是 13MB/S（纳秒级），差了几百倍。

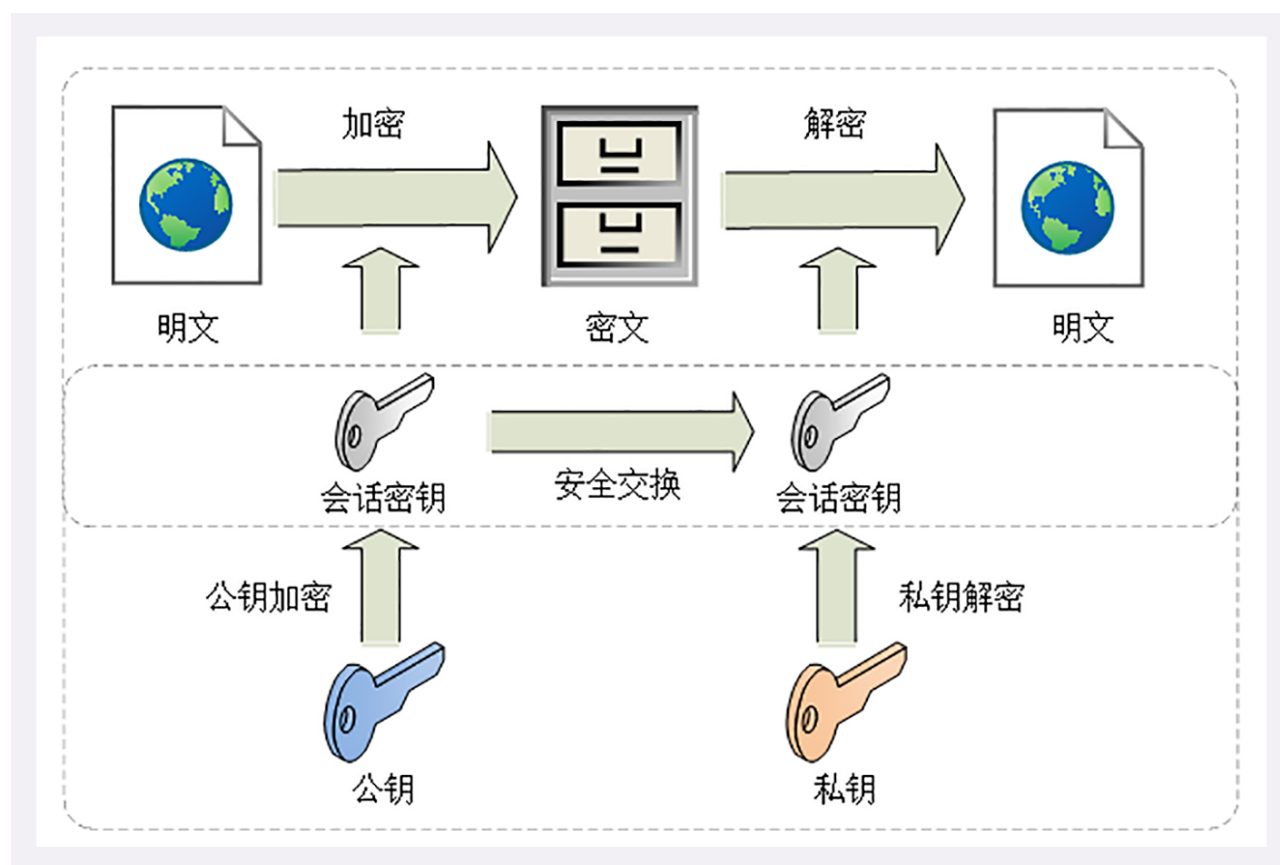
那么，是不是能够把对称加密和非对称加密结合起来呢，两者互相取长补短，即能高效地加密解密，又能安全地密钥交换。

这就是现在 TLS 里使用的**混合加密**方式，其实说穿了也很简单：

在通信刚开始的时候使用非对称算法，比如 RSA、ECDHE，首先解决密钥交换的问题。

然后用随机数产生对称算法使用的“**会话密钥**”（session key），再用公钥加密。因为会话密钥很短，通常只有 16 字节或 32 字节，所以慢一点也无所谓。

对方拿到密文后用私钥解密，取出会话密钥。这样，双方就实现了对称密钥的安全交换，后续就不再使用非对称加密，全都使用对称加密。



这样混合加密就解决了对称加密算法的密钥交换问题，而且安全和性能兼顾，完美地实现了机密性。

不过这只是“万里长征的第一步”，后面还有完整性、身份认证、不可否认等特性没有实现，

所以现在的通信还不是绝对安全，我们下次再说。

小结

1. 加密算法的核心思想是“把一个小秘密（密钥）转化为一个大秘密（密文消息）”，守住了小秘密，也就守住了大秘密；
2. 对称加密只使用一个密钥，运算速度快，密钥必须保密，无法做到安全的密钥交换，常用的有 AES 和 ChaCha20；
3. 非对称加密使用两个密钥：公钥和私钥，公钥可以任意分发而私钥保密，解决了密钥交换问题但速度慢，常用的有 RSA 和 ECC；
4. 把对称加密和非对称加密结合起来就得到了“又好又快”的混合加密，也就是 TLS 里使用的加密方式。

课下作业

1. 加密算法中“密钥”的名字很形象，你能试着用现实中的锁和钥匙来比喻一下吗？
2. 在混合加密中用到了公钥加密，因为只能由私钥解密。那么反过来，私钥加密后任何人都可以用公钥解密，这有什么用呢？

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 严格来说对称加密算法还可以分为块加密算法 (block cipher) 和流加密算法 (stream cipher)，DES、AES 等属于块加密，而 RC4、ChaCha20 属于流加密。

- 02 ECC 虽然定义了公钥和私钥，但不能直接实现密钥交换和身份认证，需要搭配 DH、DSA 等算法，形成专门的 ECDHE、ECDSA。RSA 比较特殊，本身即支持密钥交换也支持身份认证。
- 03 比特币、以太坊等区块链技术里也用到了 ECC，它们选择的曲线是 secp256k1。
- 04 由于密码学界普遍不信任 NIST 和 NSA，怀疑 secp 系列曲线有潜在的弱点，所以研究出了“x25519”，它的名字来源于曲线方程里的参数“ $2^{255} - 19$ ”。另有一个更高强度的曲线“x448”，参数是“ $2^{448} - 2^{224} - 1$ ”。
- 05 在 Linux 上可以使用 OpenSSL 的命令行工具来测试算法的加解密速度，例如“openssl speed aes”“openssl speed rsa2048”等。
- 06 TLS1.2 要求必须实现 TLS_RSA_WITH_AES_128_CBC_SHA，TLS1.3 要求必须实现 TLS_AES_128_GCM_SHA256，并且因为前向安全的原因废除了 DH 和 RSA 密钥交换算法。



[上一页](#)

[下一页](#)