SSL简述

SSL的历史

这其实不重要

只要分清SSL和TLS，还有版本号就行了。

SSL的设计目标

提供安全可靠的数据传输层。

对抗嗅探，内容替换，身份替换。

先从Alice和Bob开始

假设Alice需要和Bob安全的通讯，那么最简单的方法是什么？

方案1

假如Alice和Bob拥有相同的密钥。基于这个密钥，使用AES-CBC模式进行通讯。

问题：

1. 多个Alice和Bob，是否两两都要共享密钥？
2. 如果攻击者记录了他们的通讯过程，某一天他们的密钥泄漏了，是否能够解密出更加关键的信息？

方案2

Alice和Bob现场交换密钥？这样可以避免密钥组合爆炸，分发问题，和密钥泄漏导致的安全性问题。

分析：

这更蠢。因为交换密钥的现场也在监听嗅探中，所以密钥对攻击者是可知的。

Diffie-Hellman密钥交换算法

Alice

1. 数字a，质数p，和原根m
2. 发送p和m
3. 计算p^a mod m = A发送
4. B^a mod m = p^{ab} mod m

于是双方就有了共享的秘密p^{ab} mod m = s

Bob

1. 数字b
2. 接收p和m
3. 计算p^b mod m = B发送
4. A^b mod m = p^{ab} mod m

而攻击者有p，m，A，B，无法算出a或b，进而推算出共享秘密。

方案3

Alice和Bob使用Diffie-Hellman交换密钥，然后通讯。

分析：

虽然对于监听者，这没问题。但是主动攻击者Mallory可以发起MITM(man in the middle)攻击。由此，Alice和Bob的Diffie-Hellman交换过程被换为Alice-Mallory和Mallory-Bob。

对于MITM而言，不能鉴别对方身份的防御手段都是纸老虎。

方案4

Alice和Bob共同持有一个密钥K。他们通过互相发送K来验证对方身份。

分析：

谁先发呢？

剪刀-石头-布

方案5

Alice和Bob共同持有一个密钥K。

Alice发出一个随机数random。Bob应答HMAC(K, random)。反之亦然。由此可以在不传输K的情况下断言对方是否拥有同样的密钥K。攻击者没有K，无法伪造回应。

分析：

听起来挺好，可是，这对防御MITM有什么帮助呢？Mallory可以把Alice和Bob的质询请求和回应互换，从而完成质询。

方案6

Alice和Bob共同持有一个密钥K。

首先进行DH密钥交换，得到共享密钥s，并进入保密通讯过程。

Alice发出一个随机数random。Bob应答HMAC(K, random+s)。反之亦然。交换Alice和Bob的回应会导致混合两个不同的s，导致验证失败。

分析：

理论上没什么问题，所以所有的Alice和Bob都需要持有一对一样的K么？

RSA算法

其实怎么工作的一点都不重要。

简单来说，公钥加密，私钥解密。私钥加密，公钥解密。

由此可以导出“签署”——对内容做Hash，再对Hash做私钥加密。如果能够通过公钥解密并核对Hash，那么这个操作必然出自于拥有私钥的人。

方案7

Alice和Bob互相交换公钥，并且用公钥签署DH协议的到的s。如果攻击者Mallory分别进行DH过程，那么Mallory无法签署有效的签名。

分析：

问题是Alice怎么知道哪个是Bob真实的公钥。Mallory可以把自己的公钥发送给Alice和Bob，然后用私钥分别签署两个s。这又回归到了方案6的问题，所有的Alice-Bob都需要交叉持有信息么？

公钥既然能够被用来验证身份，那么谁来验证公钥的真伪呢？

对于公钥的签署

RSA可以签署信息。公钥也是一种信息。所以RSA可以签署公钥。

方案8

Alice和Bob的公钥都被某个可信机构A所签署了。在交换公钥的时候，也同时附带机构A的签名。如果未见签名，则不被认可。后面基本同方案7。

分析：

通常来说，攻击者Mallory不会“恰好”拥有机构A的签名。但是在机构A对大多数人都提供签名的时候，Mallory可以用合法的方式搞到机构A的签名(例如Mallory也向A机构申请签名)。由此，Mallory可以合法的将自己的证书交给Alice和Bob，从而完成MITM。

因此不但需要对公钥签名，而且必须约束签名的目标实体，可用范围，用途之类的信息。

方案9

Alice和Bob的公钥，连通身份信息(哪个国家，哪个公司，用于什么域名)合并在一起，被机构A签署。这三部分信息合起来，被称为一张“证书”。Alice和Bob通讯伊始，双方先交换证书。Mallory无法获得证书，或者获得的证书无法匹配身份信息(例如域名)，从而无法伪造身份。

分析：

和真实世界的PKI体系已经非常像了。

总结一下

1. Alice和Bob的证书都被可信机构签署了。
2. Alice和Bob互相交换证书，并验证签署和可用范围是否吻合。
3. Alice和Bob进行DH密钥交换，在交换过程中，互相使用密钥签署信息。
4. 通过DH密钥交换，Alice和Bob现在拥有相同的密钥K。
5. 他们使用K进行安全的通讯。

许多注释

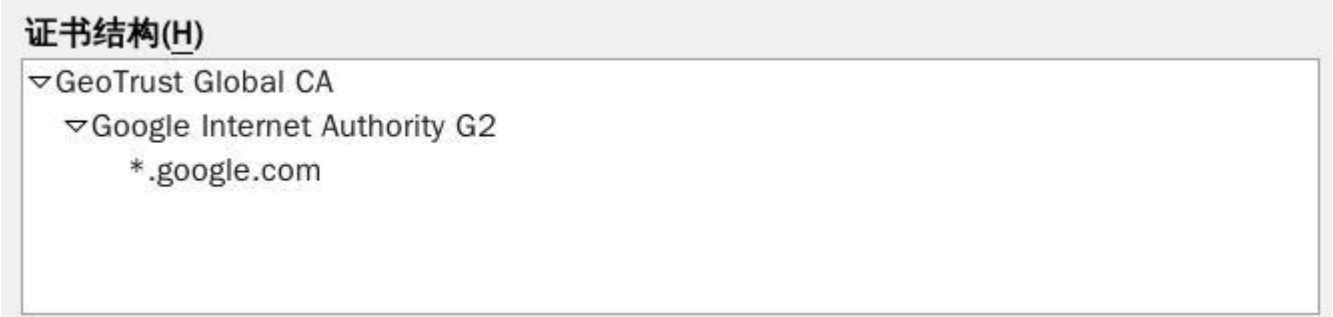
1. 实际的TLS的密钥交换过程并不是DH加个签名就算。
2. TLS支持很多协议族。例如密钥交换算法(Key Exchange)，加密算法(Cipher)，签名算法(Message authentication code)。实际中，将RSA改为ECC算法，前后流程也是一样的。因此在SSL协议框架中，很多具体算法是可变的，需要靠服务器端和客户端协商。
3. 很多算法的安全性实际是不足的，需要调优。
4. 实际生活中并不是Alice和Bob直接由公信机构签署，而是公信机构信任一些中间机构，签署他们并委托他们对最终用户进行审核。

证书链

如我们刚刚所说，Alice和Bob实际是被中间证书签署的。而当证书交换的时候，可能对方并没有中间证书。这样就无法完成完整的证书链验证。所以，我们需要在证书里面构成完整的链条。一个完整的证书，应当包含最终证书，向上的每一级中间证书，和根证书。原则上根证书可以没有，因为根证书必然已经被包含到了目标系统的系统中(否则无法构成信任)。但是通常情况下，我们仍然会附带上根证书。

证书链的例子

Google的泛域名证书



检查服务器证书链

```
$ openssl s_client -connect www.google.com:443
CONNECTED(00000003)
depth=2 C = US, O = GeoTrust Inc., CN = GeoTrust Global CA
verify error:num=20:unable to get local issuer certificate
verify return:0
---
Certificate chain
 0 s:/C=US/ST=California/L=Mountain View/O=Google Inc/CN=www.google.com
   i:/C=US/O=Google Inc/CN=Google Internet Authority G2
 1 s:/C=US/O=Google Inc/CN=Google Internet Authority G2
   i:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
 2 s:/C=US/O=GeoTrust Inc./CN=GeoTrust Global CA
   i:/C=US/O=Equifax/OU=Equifax Secure Certificate Authority
---
```

可以加-showcerts显示证书，再用openssl x509 -in server.crt -text显示证书细节。

安全协议族

简单点说:
ssl_ciphers HIGH:!aNULL:!MD5;

复杂点说:
ssl_ciphers 'EECDH+AESGCM:EDH+AESGCM:AES256+EECDH:AES256+EDH';

最正确的写法(兼容性角度考虑):
ssl_ciphers "EECDH+AESGCM:EDH+AESGCM:ECDHE-RSA-AES128-GCM-SHA256:AES256+EECDH:DHE-RSA-AES128-GCM-SHA256:AES256+EDH:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES128-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA:DHE-RSA-AES128-SHA:ECDHE-RSA-DES-CBC3-SHA:EDH-RSA-DES-CBC3-SHA:AES256-GCM-SHA384:AES128-GCM-SHA256:AES256-SHA256:AES128-SHA256:AES256-SHA:AES128-SHA:DES-CBC3-SHA:HIGH:!aNULL:!eNULL:!EXPORT:!DES:!MD5:!PSK:!RC4";

加固SSL的安全性

1. ssl_protocols TLSv1 TLSv1.1 TLSv1.2;
2. openssl dhparam -out dh.pem 2048
3. ssl_dhparam dh.pem;
4. ssl_prefer_server_ciphers on;
5. ssl_session_cache shared:SSL:10m;
6. ssl_stapling on;

可以参考这个
https://raymii.org/s/tutorials/Strong_SSL_Security_On_nginx.html

证书技巧

证书key长度应正好为2048。1024位的被认为不够安全。而SSL主要开销来自于握手时的RSA算法。key长度越大，握手速度越慢。所以2048位是一个比较好的值。

证书不应使用sha1签署，因为sha1算法有碰撞性疑虑。目前chrome已经不承认sha1证书了。