文章已收录我的仓库：Java学习笔记

计算机网络中的安全

前言

本文主要研究具体的算法思想而不关注具体的协议实现，在接下来的文章中我门再把目光投向 SSL/TLS。

在具体分析之前，我们首先来思考一下计算机网络中会面临那些安全问题，只有理清楚了这些问题，我们才能知道为什么需要这些安全技术，并且和面试官谈的时候条理也能更清楚。

报文机密性 - 我们的谈话会被窃听吗？

当然会，你肯定不希望你的谈话会被别人监听，所以你需要一些加密手段让别人听不懂你的谈话。

网络中也是一样，如果不对报文加密，则任何一台中间计算机都可以窃听你的报文，如果报文中包含了你的银行密码这可就完蛋了。

报文完整性 - 我收到的报文被篡改过吗？

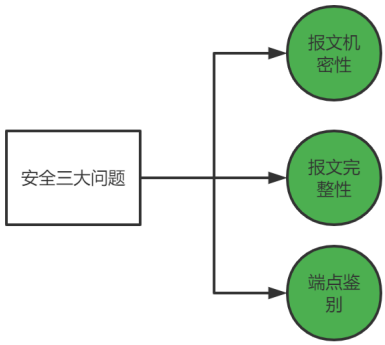
这里的完整性并不是数据丢失问题，数据丢失在应用层之下就应该通过校验码检测出来了，本文的完整性是指：**报文有没有被篡改？**

你肯定不希望你的报文在网络中被别人篡改，所以必须要有些验证方法去验证报文完整性。

端点鉴别 - Alice，真的是你在给我发消息吗？

你要怎么确定对方真的是 Alice，而不是 Bob 伪造的，正如在现实生活中，你要怎么确定和你裸聊的是一个女性而不是一个五旬老汉？

带着上述三大问题，让我们一起来看看计算机科学家们提出了什么样的解决方案。



报文机密性

我们要保证报文是加密的，并且只有通信双方能够解密。

对称密匙密码体制

这种体制中，通信双方协商好相同的密匙和算法，发送方利用密匙加密报文，而接收方利用密匙解密报文，因为其他人不知道密匙，所以其他人无法对报文解密。

加密方法有许多种，例如最简单的凯撒密码，它将所有字母后移 k 位，例如 abc 可能被替换为 bcd，此时 k = 1 就是密匙。

在对称密匙体制中，主要采用**块密码**和**流密码**进行加密，流密码多用于WLAN中，我们主要看看块密码。

块密码

块密码的主要原理是将明文分为固定大小的块，**对每一种块都维护一个映射表以加密数据**。

例如，假设块大小为 3，考虑下列映射表：

表 8-1　一种特定的 3 比特块密码

输入	输出	输入	输出
000	110	100	011
001	111	101	010
010	101	110	000
011	100	111	001

例如源数据为：000, 001, 101 将会被加密成 110, 111, 010。解密逆过程即可。

那么这里的映射表就是通信双方的密匙，上述表有 8 中输入，而每种输入又可能对应 8 种输出，所以想要破解密匙需要尝试 8 的阶乘种可能，阶乘函数上升是非常快的，当块大小变得比较大时，破解块密码在理论上说是不可能的。

但是，出现了新的问题，如果我们让块大小增大，假设为 64bit，那么交互的双方都必须维护一张长达 2^64 大小的映射表(表项为输入输出)，更可怕的情况是如果映射有所修改，那么这将是一个非常大的工作量。

取而代之是采用**函数**来模拟输入输出表，例如对所有二进制输入流进行亦或运算得到加密输出流。当然实际的函数不可能如此简单，它依赖于具体的实现。

获取我们还可以将每个加密块打乱，以实现让更小的函数映射模拟更大的块输入输出表。

例如我们用函数来模拟 64bit 输入输出表，我们将每个 64bit 作为一个块，但我们没有维护 64bit 的输入输出表，而是将这 64bit 分为 8 个 8bit 块，对**每个 8bit 块维护一个特定的函数**(图中的T1、T2...)，而后对这 8个加密的 8bit 块进行 64bit 函数置乱，使得每个 8bit 块被打乱，经过一定次数循环后，**此时的 64bit 块就已经被完全打乱了**，就好像我们采用了 64bit 输入输出表一样。

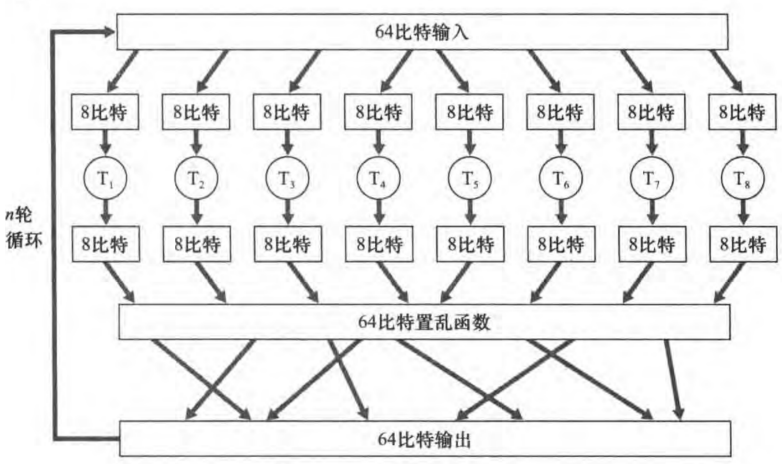


图 8-5　一个块密码的例子

在上图中，密匙是 9 个函数，即 8 个映射函数和 1 个置乱函数(假定循环次数已知)。这些函数都可以由一个唯一的**二进制密匙**生成，现在通信双方只需要维护一个二进制串即可，无须维护一个巨大的映射表。

典型的块密码协议如 DES 与 AES 都是采用函数而非预先设定的表，它们定义了一些列算法以生成这些小型函数映射与置乱函数排列，不过前提是需要一个密匙来生成。DES(就像我们例子中的) 的密匙比特串长度为 56，块大小为 64bit，在现代计算机运行速度下，2^{56}中可能破解已非难事，因此 DES 已经被淘汰成为历史了，取而代之的是 AES，AES 的密匙长度最小为 128 位，理论上是不可能破解的。

> — 如何利用密匙生成 “特定小型表函数映射” 和 “特定排列函数”？
>
> — 我也不知道。

如何提前协商密匙

如何提前协商密匙在对称密匙体制中是最为重要的一个问题，这个过程必须足够安全才行。

不过解决办法总是有的，就像我们马上要讲的，可以利用非对称密匙密码体制来协商一个相同的密匙。

非对称密匙密码体制(公开密匙密码)

在非堆成密匙密码体制中，假定每个网络用户都拥有私匙和公匙，私匙是个人私有的，公匙是公开在网络上的，我们定义利用私匙处理数据算法为 S，而利用公匙处理数据算法为 P。

那么，非对称密匙体制要求有如下公式成立：

$$S(P(data)) = data\ P(S(data)) = data$$

其中 data 是网络上传输的二进制比特流，现在假设 A 要像 B 发送消息 data，那么 A 直接直接用 B 的公匙(公开的)对 data 加密得到 $P_B(data)$，那么 B 可以直接利用其私匙对数据解密 $S_B(P_B(data)) = data$，这样 B 就能够还原原文，而其他人不知道 B 的私匙，所以无法破解密文。

你可能会想，天啊，怎么可能会有这种算法！如果你真有这种想法，说明你低估了数论的魅力。

RSA算法

RSA 已经成为了公开密匙密码的代言人，RSA 算法结论如下：

$$(m^e\ mod\ n)^d\ mod\ n = (m^d\ mod\ n)^e\ mod\ n = m\ e,d是经过特殊运算生成的$$

通过这个神奇的结果，我们选定 e 和 d 为公匙和私匙，假定 m 为 报文 message，那么有：

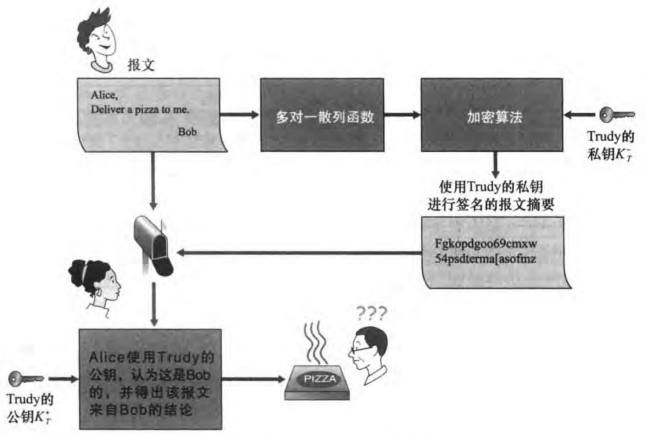
$$P(m) = m^e\ mod\ n\ S(m) = m^d\ mod\ n\ S(P(m)) = P(m)^d\ mod\ n = (m^e\ mod\ n)^d\ mod\ n = m\ P(S(m)) = S(m)^e\ m$$

利用这种的公匙算法和私匙算法就可以完成非对称密匙密码体制加密的任务。

RSA 的安全性依赖于**目前没有一个已知的算法可以快速对一个大数进行因数分解。**如果相对RSA有更深的了解可以翻到文末看看对 RSA 算法的证明，不过不清楚也没事，那毕竟是数学上的事。

RSA的缺陷是，当 e 和 d 较大时，**指数运算与取模运算都是比较慢的操作，并不适合用来频繁通信**，因此在未来的很长一段时间内对称密匙与非对称密匙共存都是共同的主角，在 HTTPS 中我们将看到 SSL 利用非对称密匙体制来协商一系列密匙，而在协商完毕后使用对称密匙进行通信。

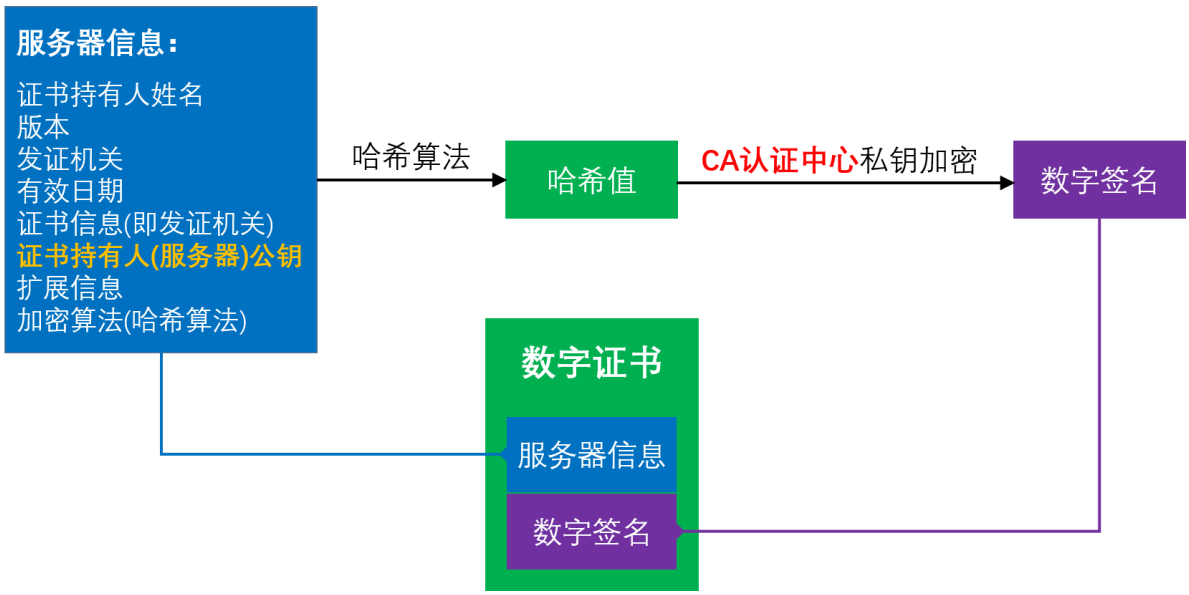
不可信的公匙



Alice 要如何知道自己所知道的公匙真的是 Bob 的呢？攻击者完全宣称自己的公匙和私匙是 B 的！使用数字签名？不好意思，数字签名本身也是依赖公匙私匙体制来实现的。

我们需要一个可以信得过的机构告诉我们公匙是否合法，这种机构就是 CA（认证中心），机构会为每个注册的实体签发一份数字证书，这个数字证书包括注册人的公匙和他的标识信息，CA 会用自己的私匙加密整数以防止服务商伪造整数(数字签名)。

在 Bob 发送公匙给 Alice 的时候会顺便发送 CA 签发的证书，Alice 比对证书来验证 Bob 的合法性。现在攻击者无法冒充 Bob 了，因为攻击者没有证书，即使攻击者也有证书，证书上也会有他的名字，Alice 可以很快知道这个 Bob 是冒充的。



报文完整性

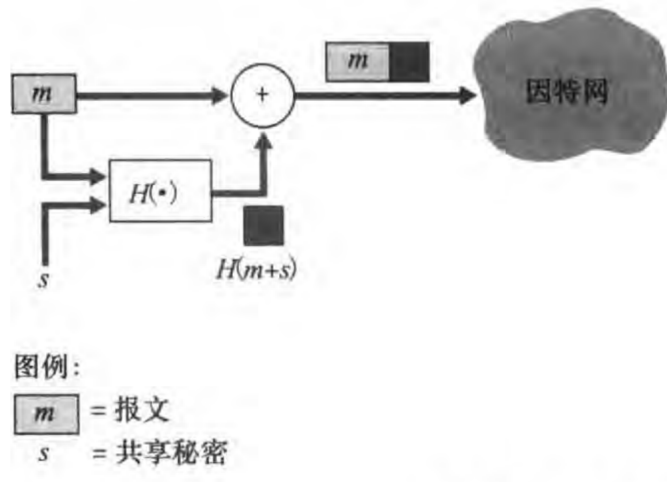
为维护报文完整性，我们需要对报文进行**多对一映射**(散列)，例如将 1000 比特的报文映射为 128 比特的**摘要(也称报文鉴别码，MAC)**，该摘要的每一个比特都与源报文相关，因此发送报文时将**摘要**一起发送，而接受报文时，根据报文重新计算一遍摘要，比对原先的摘要，如果不同则说明报文可能被篡改。

> 摘要长度须是固定的，这样接收方才能正确的取出摘要。

攻击者仅篡改报文或者仅篡改摘要都是不可行的，但如果攻击者知晓了散列函数，**更改报文并同时替换摘要**，此时接收方收到报文并计算摘要，并不会发现有任何问题，然而报文已经被篡改了！

所以，为维护报文完整性，我们仍然需要一个密匙，发送方在计算散列时**将密匙(字符串)附加在报文上计算摘要**(发送的时候不会发送密匙)，而在接收方同时也附加密匙计算摘要，这样便解决了上述问题，因为攻击者无法不知道通信的密匙！

注意，整个报文和摘要也需要保证报文机密性。现在需要对整个报文+摘要进行加密。



图例：

m = 报文

s = 共享秘密

常用的密码散列函数 MD5 和 SHA-1、SHA-2、SHA-3

散列函数必须要足够安全，至少要保证**理论上**不可能有多个不同的报文可以映射到相同的摘要，MD5声称它可以做到这一点，不过随着密码学的发展，在 2004 年后 MD5 变得不安全了，取而代之的是 SHA-1，SHA-1是 MD5 的加强版，思想上与 MD5 类似。不幸的是，2012年后，SHA-1也被指责说是不安全的，在 2015 年，破解 SHA-1甚至只需要不到一个小时，现在 SHA-1 已经被 SHA-2、SHA-3 所取代。

不过 MD5 的思想依赖未改变，后续的算法都是基于 MD5 算法思想加强的，MD5 的算法能够将任意长的报文映射为 128 bit 的摘要，过程如下：

1. 将任意长的报文采用**二进制模运算**模 2^{64} 计算其余数(64位)，追加报文尾部。
2. 在报文与余数之间填充 1~512 bit，使得填充后总长度是 512 的整数倍。填充的首位是1，后面都是0。
3. 把追加后的报文分成一个个 512bit 数据块，每个 512 字节分为 4 个 128bit数据块，与 4 个 32bit 幻数分别进入主循环，每个幻数与每个 128bit 数据块进行交互运算，最终输出 4 个幻数，这四个幻数与这 512 字节存在某种联系，然后继续与下一个 512bit 进行循环，最终 4个幻数组合起来为 128bit 鉴别码。

更多细节可自行了解。

端点鉴别

数字签名

接收方 B 如何知道报文是否是 A 发送的呢？有了非对称密码加密，这点很容易做到。

A 可以用自己的私匙对 m 再次进行加密得到 C_1，而 B 收到报文C_1 后，B 尝试用 A 的公匙进行解密，如果得到的报文是 m，则可以确认 A 的身份，因为只有用 A 的密匙加密才能用 A 的公匙解密，所以可以确定对方持有 A 的密匙。

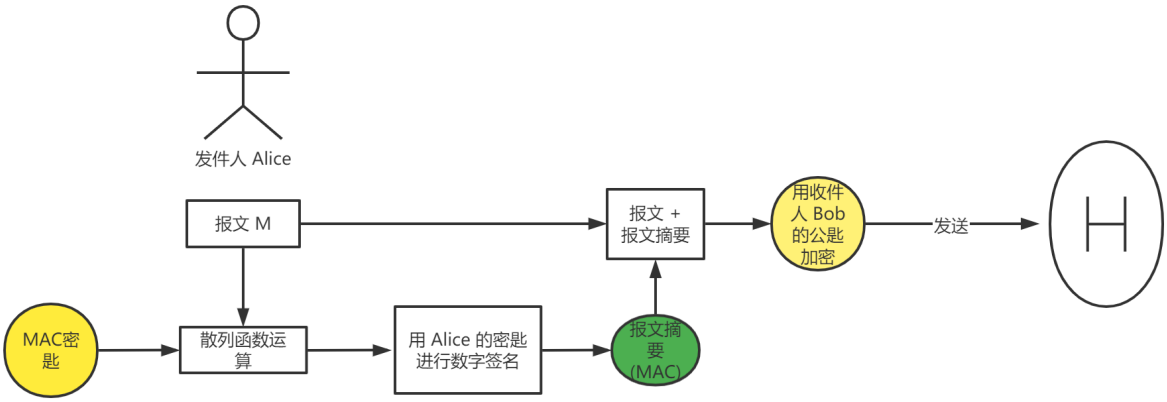
至于这个报文 m 到底该是什么，取决于具体的实现。它可以是确定好的一段数据，也可以是通信时的普通报文，不过大多数情况下，通常会对报文摘要(鉴别码)进行签名，因为对普通报文签名开销太大了，报文可能非常长！

如果对摘要签名，接收方用发送方的公匙解密即可得到摘要，它可以根据报文重新计算摘要以比对正确性，如果比对不成功，可能会有两种可能，要么是签名错误、要么是报文完整性丢失，无论哪种情况 B 肯定都不会信任该报文了。

常见的攻击

攻击在大类上分为主动攻击和被动攻击，主动攻击即主动发起攻击，可能涉及到报文篡改等；被动攻击则是指只是秘密的窃听你的报文，不会更改你的报文。被动攻击通常难以防范，只能采取更复杂的加密算法进行抑制，我们今天谈的主要是主动攻击。

在讨论攻击前，我们先假设我们上述讨论的所有措施全部使用，在此基础上看看还能有什么些攻击。



协议有关的攻击

SYN泛洪攻击

SYN 泛洪我在 TCP 总结中已经讲过了，具体就不在说了，可参考我的博客：万字长文总结TCP

重排序攻击

这仍是针对 TCP 协议的攻击，即使 TCP 数据报已经被加密，但 TCP 首部仍然是明文的，攻击者可以劫持 TCP 报文，**重排序(乱排序) TCP 首部中的序号字段**，使得接收方接收到的字节流是乱序的，且接收方**无法判断出异常**，信以为真。

针对这种攻击的防御就是在 TCP 加密的数据报中继续定义序号，接收方验证加密的数据报中序号以判断顺序。

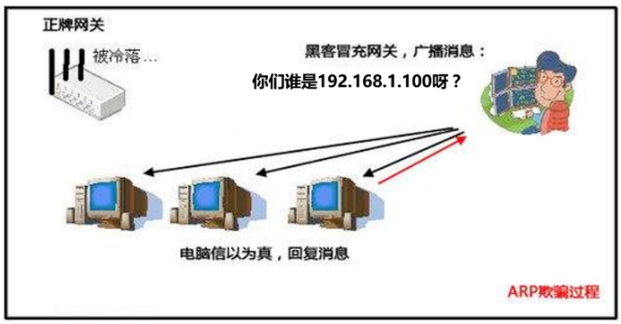
ARP欺骗

ARP 协议是解析 MAC 地址的一种协议，当本地网关在自己的缓存表中没有找到 IP 对应的 MAC 地址，就会在局域网中广播消息，寻找对应 IP 的主机，而所有主机会监听 MAC 报文，如果是在寻找自己则即使回答自己的 MAC 地址。

在 **Linux 上可利用 `arpspoof` 命令来进行 ARP 地址欺骗**(快去试试吧，我成功的攻击了我的好兄弟)，从而达到让局域网内用户断网的攻击，有兴趣的朋友门可以自己去试一试，篇幅有限就不在这里说了。

> **ARP欺骗**（英语：**ARP spoofing**），又称**ARP毒化**（**ARP poisoning**，网络上多译为**ARP病毒**）或**ARP攻击**，是针对以太网地址解析协议（ARP）的一种攻击技术，通过欺骗局域网内访问者PC的网关MAC地址，使访问者PC错以为攻击者更改后的MAC地址是网关的MAC，导致网络不通。此种攻击可让攻击者获取局域网上的**数据包甚至可篡改数据包**，且可让网络上特定计算机或所有计算机无法正常连线。最早探讨ARP欺骗的文章是由Yuri Volobuev所写的《ARP与ICMP转向游戏》（*ARP and ICMP redirection games*）。

冒充网关



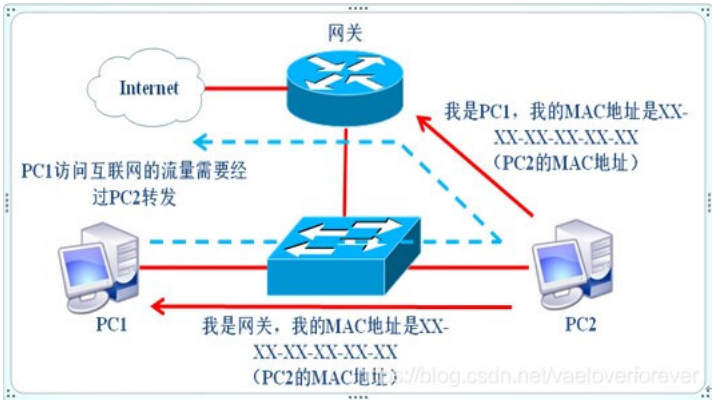
如果存在不法分子冒充网关(局域网内的，局域网外的无法通过网关)发送 ARP Request 报文，则局域网上的主机可能会错把黑客当作自己的网关，这些主机可能会更新自己的 MAC 缓存表，将黑客认为是自己的网关，则之后所有的信息都发给自己的网关。

事实上，如果我们的数据都进行了**加密和报文完整性鉴别**的话，那么黑客即使充当了网关也只能干巴巴的看着我们的加密数据，篡改也没有用，但是，黑客却可以让局域网内所有主机都无法访问外网，造成断网现象(或者是故意让网络变慢)，而且，黑客可以进行重放攻击。

不过，不是什么数据都会进行加密的，一旦你露出马脚，黑客就会立马攻击你，例如我们很快就会说到的 DNS 欺骗

充当中间人 —— 双重欺骗

黑客充当中间人窃听主机与网关的通信，例如下图：



此时主机 PC1 以为 PC2 是网关，而网关以为 PC2 是 PC1，于是接下来所有的信息都将会发送给 PC2，这种欺骗方式是冒充网关的加强版，黑客现在不仅冒充网关，还冒充主机。

防范方法

- 配置 MAC 地址为静态地址，不允许改变。这种方法对网关通常不太可取，不过主机可以将他知道的网关地址配置成静态的。
- 网上设备可借由DHCP保留网络上各计算机的MAC地址，在伪造的ARP数据包发出时即可侦测到。通俗点说，主机在注册IP时同时注册了自己的MAC地址，使得路由完全掌握所有 IP 对应的 MAC地址，因此能够检测假冒ARP包。此方式已在一些厂牌的网上设备产品所支持。
- 网关更新速度更加频繁些，避免“一次跌倒，则放弃挣扎”。
- 嗅探网络 ARP 包，如果有不正当(对网关而言，除自己发的Request之外不应该有其他人发送 Request)或ARP包过多时，则及时告知网络管理员。

> ARP欺骗亦有正当用途。其一是在一个需要登录的网上中，让未登录的计算机将其浏览网页强制转向到登录页面，以便登录后才可使用网上。另外有些设有备援机制的网上设备或服务器，亦需要利用ARP欺骗以在设备出现故障时将讯务导到备用的设备上。[1]

DNS欺骗

DNS 欺骗就是攻击者冒充域名服务器的一种欺骗行为。 原理：如果可以**冒充域名服务器**，然后把查询的 IP 地址设为攻击者的 IP 地址，这样的话，用户上网就只能看到攻击者的主页，而不是用户想要取得的网站的主页了，这就是 DNS 欺骗的基本原理。DNS 欺骗其实并不是真的“黑掉”了对方的网站，而是冒名顶替、招摇撞骗罢了。

例如你正在请求某个银行的网址，你的 DNS 请求被黑客劫持了，黑客将返回了个假冒的 IP 给你，而这个假冒 IP 返回的页面和真实的一模一样，你信以为真，于是你的所有信息都暴露了。

黑客要如何劫持你的 dns 请求呢？通常有两种思路，一种是直接黑进你的 DNS Server，冒充DNS服务器，不过这点难度比较大；第二种就是常见的中间人劫持了，例如 ARP 欺骗。

此外，某些病毒可能会修改你的 host 文件(DNS 缓存文件)。

防范：

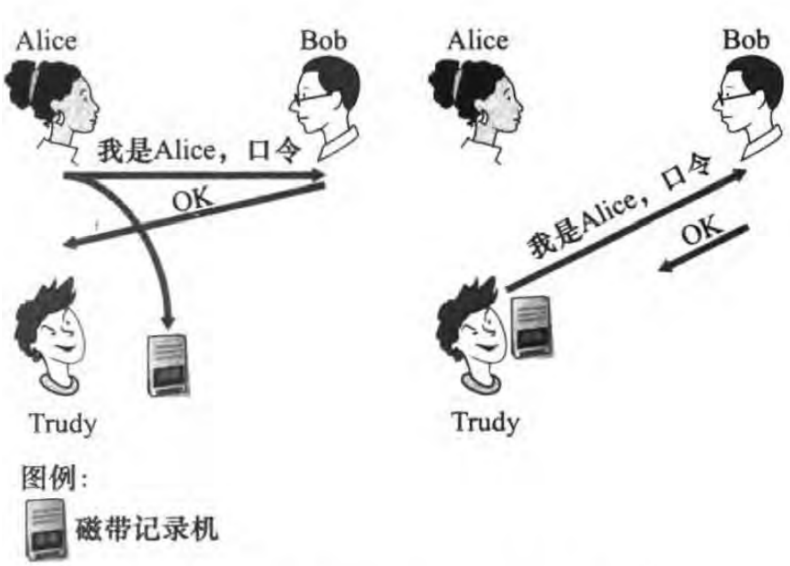
- 直接使用 ip 而不使用域名。
- 定期检查 host 文件。
- 防范 ARP 欺骗。
- 要求DNS服务器使用数字签名。
- 检测 DNS 应答，通常假冒的应答要比真实的应答简单的多。
- 采用 HTTPS 而不是 HTTP，碰到不合法证书提示立马警觉。

协议无关的攻击

重放攻击

重放攻击的基本原理就是**把以前窃听到的数据原封不动地重新发送给接收方**。很多时候，网络上传输的数据是加密过的，此时窃听者无法得到数据的准确意义。但如果他知道这些数据的作用，就可以在不知道数据内容的情况下通过再次发送这些数据达到愚弄接收端的目的。例如，有的系统会将鉴别信息进行简单加密后进行传输，这时攻击者虽然无法窃听密码，但他们却可以首先截取加密后的口令然后将其重放，从而利用这种方式进行有效的攻击。

例如下图：



你可能觉得重放攻击有什么大不了的，黑客又不知道具体的内容，如果这样想你就错了，假如你正给银行发送请求转账给别人，但此时你的所有请求都被黑客嗅探并且保存了下来，在某一时刻，黑客重放这些请求，于是你就发现你的钱全没了。

防范方法：

- **加随机数**。该方法优点是认证双方不需要时间同步，双方记住使用过的随机数，如发现报文中有以前使用过的随机数，就认为是重放攻击。缺点是需要额外保存使用过的随机数，若记录的时间段较长，则保存和查询的开销较大。
- **加时间戳**。该方法优点是不用额外保存其他信息。缺点是认证双方需要准确的时间同步，同步越好，受攻击的可能性就越小。但当系统很庞大，跨越的区域较广时，要做到精确的时间同步并不是很容易。
- **加版本号(或序号、流水号)**。就是双方在报文中添加一个逐步递增的整数(例如 TCP)，只要接收到一个不连续的流水号报文(太大或太小)，就认定有重放威胁。该方法优点是不需要时间同步，保存的信息量比随机数方式小。缺点是一旦攻击者对报文解密成功，就可以获得流水号，从而每次将流水号递增欺骗认证端。
- **一次性口令**。例如，和 cookie、token 机制有点像，但又不完全一样，该口令只在一次服务内有效，当服务完成后口令即为过期失效。

中间人攻击

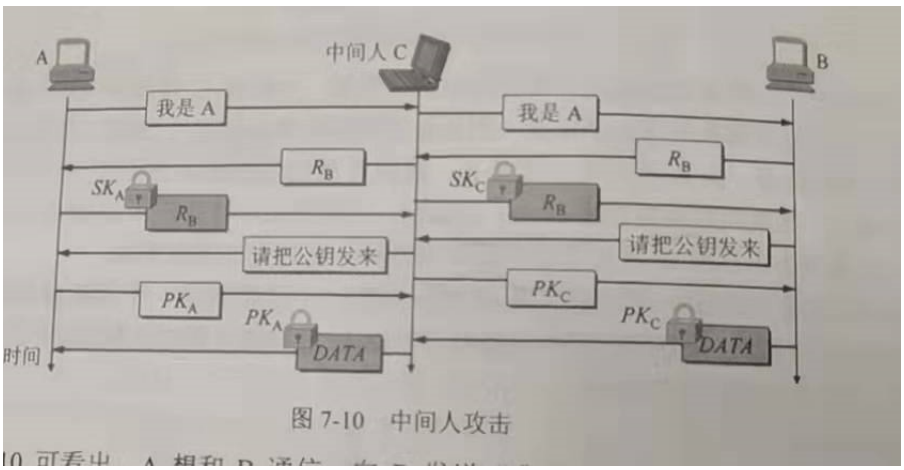
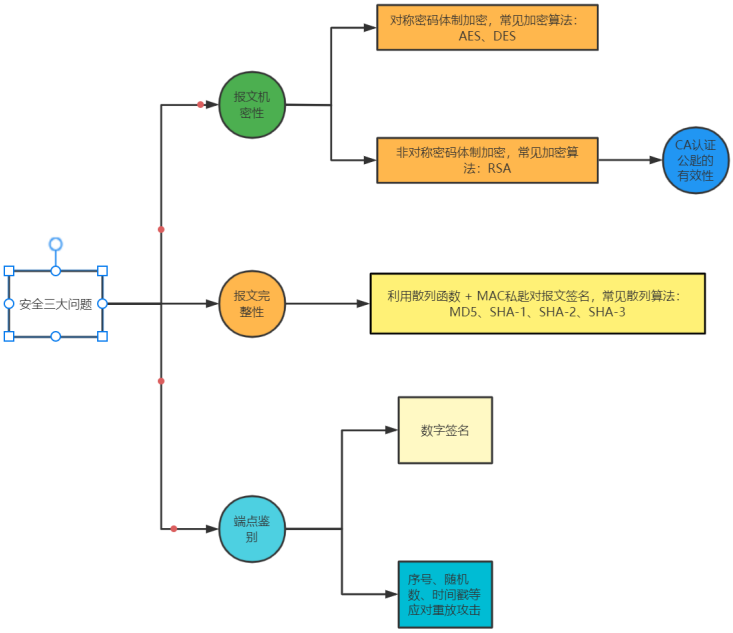


图 7-10　中间人攻击

如上图所示，这里我们假设 A-B 双方没有采用某种方式验证私匙和公匙的合法性，那么此时问题就大了，服务器错把C的公匙当作A的公匙，从而服务器发给 A 的数据完全暴露在 C 的眼皮底下。

这个例子再次告诉我们验证公匙的必要性，同时 CA 的可信程度也显得极为重要。

总结



附 RSA 算法证明

我们的算法结论是：

$$(m^e \ mod \ n)^d \ mod \ n = (m^d \ mod \ n)^e \ mod \ n = m \tag{A}$$

现在我们来指出如何选出 e、d，首先我们选择**两个大素数 p、q**，则取 n、z 满足：

$$n = p \times q \ ，z = (p-1)(q-1) \ ，n > m \tag{1}$$

现在我们选择一个数 **e**，满足：

$$\boxed{\text{e < n \ \&\& \ e 与 n互质}}$$

给定 e，我们可以选择 d，使得：

$$(e \times d) \ mod \ z = 1 \tag{2}$$

这样我们就选出了 e、d 的值，欲证明结果，需要引入数论里的一个结论，即当 n、z 满足我们上面 (1) 式，有：

$$x^y \ mod \ n = x^{(y \ mod \ z)} \ mod \ n \tag{3}$$

> 该结论由 Kaufman 提出

现在，令 x = m，y = ed，带入 (3) 式有：

$$m^{ed} \ mod \ n = m^{(ed \ mod \ z)} \ mod \ n \tag{4}$$

结合 (2)、(4) 可知：

$$m^{ed} \ mod \ n = m \ mod \ n = m \tag{5}$$

我们都知道简单数学中有一条取模运算的分配律：

$$(a \times b) \ mod \ n = [(a \ mod \ n) \times (b \ mod \ n)] \ mod \ n \tag{6}$$

根据 (6) 式可推：

$$(a^d) \ mod \ n = (a \ mod \ n)^d \ mod \ n \tag{7}$$

结合 (5) (7) 而式有：

$$m^{ed} \ mod \ n = (m^e)^d \ mod \ n = (m^e \ mod \ n)^d \ mod \ n = m \ m^{ed} \ mod \ n = (m^d)^e \ mod \ n = (m^d \ mod \ n)^e \ mod \ n = m$$

从而 (A) 式得证。

> 勘误，RSA 的算法应该并不是 1 式，1 式是数论中的一个结论，RSA 算法的核心应该是 e、d 的生成。
>
> $$(m^e \ mod \ n)^d \ mod \ n = m^{ed} \ mod \ n = (m^d \ mod \ n)^e \ mod \ n = m$$

> 这是恒成立的。