它是第一个既能用于数据加密也能用于数字签名的算法。它易于理解和操作，也很流行。算法的名字以发明者的名字命名：Ron Rivest, Adi Shamir 和Leonard Adleman。但RSA的安全性一直未能得到理论上的证明。它经历了各种攻击，至今未被完全攻破。

一、RSA算法 :

首先, 找出三个数, p, q, r,   
其中 p, q 是两个相异的质数, r 是与 (p-1)(q-1) 互质的数......   
p, q, r 这三个数便是 private key

接著, 找出 m, 使得 rm == 1 mod (p-1)(q-1).....   
这个 m 一定存在, 因为 r 与 (p-1)(q-1) 互质, 用辗转相除法就可以得到了.....   
再来, 计算 n = pq.......   
m, n 这两个数便是 public key

编码过程是, 若资料为 a, 将其看成是一个大整数, 假设 a < n....   
如果 a >= n 的话, 就将 a 表成 s 进位 (s <= n, 通常取 s = 2^t),   
则每一位数均小於 n, 然後分段编码......   
接下来, 计算 b == a^m mod n, (0 <= b < n),   
b 就是编码後的资料......

解码的过程是, 计算 c == b^r mod pq (0 <= c < pq),   
於是乎, 解码完毕...... 等会会证明 c 和 a 其实是相等的

如果第三者进行窃听时, 他会得到几个数: m, n(=pq), b......   
他如果要解码的话, 必须想办法得到 r......   
所以, 他必须先对 n 作质因数分解.........   
要防止他分解, 最有效的方法是找两个非常的大质数 p, q,   
使第三者作因数分解时发生困难.........

<定理>   
若 p, q 是相异质数, rm == 1 mod (p-1)(q-1),   
a 是任意一个正整数, b == a^m mod pq, c == b^r mod pq,   
则 c == a mod pq

证明的过程, 会用到费马小定理, 叙述如下:   
m 是任一质数, n 是任一整数, 则 n^m == n mod m   
(换另一句话说, 如果 n 和 m 互质, 则 n^(m-1) == 1 mod m)   
运用一些基本的群论的知识, 就可以很容易地证出费马小定理的........

<证明>   
因为 rm == 1 mod (p-1)(q-1), 所以 rm = k(p-1)(q-1) + 1, 其中 k 是整数   
因为在 modulo 中是 preserve 乘法的   
(x == y mod z and u == v mod z => xu == yv mod z),   
所以, c == b^r == (a^m)^r == a^(rm) == a^(k(p-1)(q-1)+1) mod pq

1. 如果 a 不是 p 的倍数, 也不是 q 的倍数时,   
则 a^(p-1) == 1 mod p (费马小定理) => a^(k(p-1)(q-1)) == 1 mod p   
a^(q-1) == 1 mod q (费马小定理) => a^(k(p-1)(q-1)) == 1 mod q   
所以 p, q 均能整除 a^(k(p-1)(q-1)) - 1 => pq | a^(k(p-1)(q-1)) - 1   
即 a^(k(p-1)(q-1)) == 1 mod pq   
=> c == a^(k(p-1)(q-1)+1) == a mod pq

2. 如果 a 是 p 的倍数, 但不是 q 的倍数时,   
则 a^(q-1) == 1 mod q (费马小定理)   
=> a^(k(p-1)(q-1)) == 1 mod q   
=> c == a^(k(p-1)(q-1)+1) == a mod q   
=> q | c - a   
因 p | a   
=> c == a^(k(p-1)(q-1)+1) == 0 mod p   
=> p | c - a   
所以, pq | c - a => c == a mod pq

3. 如果 a 是 q 的倍数, 但不是 p 的倍数时, 证明同上

4. 如果 a 同时是 p 和 q 的倍数时,   
则 pq | a   
=> c == a^(k(p-1)(q-1)+1) == 0 mod pq   
=> pq | c - a   
=> c == a mod pq   
Q.E.D.

这个定理说明 a 经过编码为 b 再经过解码为 c 时, a == c mod n (n = pq)....   
但我们在做编码解码时, 限制 0 <= a < n, 0 <= c < n,   
所以这就是说 a 等於 c, 所以这个过程确实能做到编码解码的功能.....

二、RSA 的安全性

RSA的安全性依赖于大数分解，但是否等同于大数分解一直未能得到理论上的证明，因为没有证明破解 RSA就一定需要作大数分解。假设存在一种无须分解大数的算法，那它肯定可以修改成为大数分解算法。目前， RSA 的一些变种算法已被证明等价于大数分解。不管怎样，分解n是最显然的攻击方法。现在，人们已能分解多个十进制位的大素数。因此，模数n 必须选大一些，因具体适用情况而定。

三、RSA的速度

由于进行的都是大数计算，使得RSA最快的情况也比DES慢上倍，无论是软件还是硬件实现。速度一直是RSA的缺陷。一般来说只用于少量数据加密。

四、RSA的选择密文攻击

RSA在选择密文攻击面前很脆弱。一般攻击者是将某一信息作一下伪装( Blind)，让拥有私钥的实体签署。然后，经过计算就可得到它所想要的信息。实际上，攻击利用的都是同一个弱点，即存在这样一个事实：乘幂保留了输入的乘法结构：

( XM )^d = X^d \*M^d mod n

前面已经提到，这个固有的问题来自于公钥密码系统的最有用的特征--每个人都能使用公钥。但从算法上无法解决这一问题，主要措施有两条：一条是采用好的公钥协议，保证工作过程中实体不对其他实体任意产生的信息解密，不对自己一无所知的信息签名；另一条是决不对陌生人送来的随机文档签名，签名时首先使用One-Way HashFunction 对文档作HASH处理，或同时使用不同的签名算法。在中提到了几种不同类型的攻击方法。

五、RSA的公共模数攻击

若系统中共有一个模数，只是不同的人拥有不同的e和d，系统将是危险的。最普遍的情况是同一信息用不同的公钥加密，这些公钥共模而且互质，那末该信息无需私钥就可得到恢复。设P为信息明文，两个加密密钥为e1和e2，公共模数是n，则：

C1 = P^e1 mod n

C2 = P^e2 mod n

密码分析者知道n、e1、e2、C1和C2，就能得到P。

因为e1和e2互质，故用Euclidean算法能找到r和s，满足：

r \* e1 + s \* e2 = 1

假设r为负数，需再用Euclidean算法计算C1^(-1)，则

( C1^(-1) )^(-r) \* C2^s = P mod n

另外，还有其它几种利用公共模数攻击的方法。总之，如果知道给定模数的一对e和d，一是有利于攻击者分解模数，一是有利于攻击者计算出其它成对的e’和d’，而无需分解模数。解决办法只有一个，那就是不要共享模数n。

RSA的小指数攻击。 有一种提高 RSA速度的建议是使公钥e取较小的值，这样会使加密变得易于实现，速度有   
所提高。但这样作是不安全的，对付办法就是e和d都取较大的值。

RSA算法是第一个能同时用于加密和数字签名的算法，也易于理解和操作。RSA是被研究得最广泛的公钥算法，从提出到现在已近二十年，经历了各种攻击的考验，逐渐为人们接受，普遍认为是目前最优秀的公钥方案之一。RSA的安全性依赖于大数的因子分解，但并没有从理论上证明破译RSA的难度与大数分解难度等价。即RSA的重大缺陷是无法从理论上把握它的保密性能如何，而且密码学界多数人士倾向于因子分解不是NPC问题。 RSA的缺点主要有：A)产生密钥很麻烦，受到素数产生技术的限制，因而难以做到一次一密。B)分组长度太大，为保证安全性，n 至少也要 600 bits 以上，使运算代价很高，尤其是速度较慢，较对称密码算法慢几个数量级；且随着大数分解技术的发展，这个长度还在增加，不利于数据格式的标准化。目前，SET( Secure Electronic Transaction )协议中要求CA采用比特长的密钥，其他实体使用比特的密钥。

**前言**

SSL是让人头大的东西，看起来很复杂，我学过信息安全课，但是对SSL仍然是模糊一片。对于数字证书也是一知半解，从来没有去认真研究过。只知道个大概，“反正就是对称加密和非对称加密”，详细的就不懂了。其实这些跟[操作系统](http://www.linuxde.net/tag/%e6%93%8d%e4%bd%9c%e7%b3%bb%e7%bb%9f" \o "操作系统" \t "http://blog.csdn.net/jhonguy/article/details/_blank)，计算机体系结构一样，是基础知识，即使你不是专门研究信息安全的，即使你平时用不到，这些东西也应该是必须了解的~ It’s not rocket science!

我这篇文章首先介绍有关信息安全的一些基本概念。然后分析数字证书的构成，用途以及SSL的工作原理。Enjoy!

### 一、为什么网络是不安全的？

计算机世界是基于网络的，根据目前网络的结构和实现，数据包在世界各地的路由器之间游荡，任何人都可以获得你的发送的数据包，从而获得你发送的数据。局域网内就更方便了，只要你开个[Sniffer](http://en.wikipedia.org/wiki/Sniffer" \t "http://blog.csdn.net/jhonguy/article/details/_blank)在那里监听，别人QQ聊天的信息一览无余啊~为什呢？我来简单解释一下计算机网络是如何传输数据的。

现实生活中，如果你请快递帮你寄东西，一般情况下，快递会把东西送到目的地，而不是其他地方。路由器就像是快递，在Internet上负责送数据。但和真实的快递不同的是，路由器会把你要发送的信息广播给离目的地更近的路由器，可能是一个路由器，也可能是多个路由器（别问我为啥，我不想深入了，有兴趣自己查资料）。这样你的信息就变成多份的了。复制虚拟的信息不值钱那~现实生活中的快递可不能复制你要寄的东西。一般情况下，只有一份数据会被目的计算机接收到，其他的拷贝在网络上游荡一段时间以后就被抛弃了。但是，这给黑客们有了很多可乘之机。他们在网络上监听很多垃圾信息，过滤掉没用的，留下他们感兴趣的，然后就可以偷窥别人隐私了。

正是因为网络有这样的问题，人们就发明了很多加密通信的手段，来保证自己的通信的内容不会被泄露。SSL和数字证书就是用来干这个的。

### 二、信息安全的基本概念

在解释SSL和数字证书之前，我觉得有必要解释一下几个有关信息安全的基本概念。

**安全的定义**

如果说两个人之间的通信是安全的，那么如何定义这个安全呢？

1. **保密性（Confidentiality）**   
   保密性应该很容易理解，就是只有你自己和你允许的人能看到相关的信息。这和物理文件的保密性是一样的。
2. **完整性（Integrity）**   
   所谓完整性就是你的信息没有被破坏或者篡改过。举个例子比如网络聊天，保证对方收到的信息就是你发出的信息也是信息安全的一部分。
3. **可获得性（Availability）**   
   可获得性是指你自己在需要的时候能够访问到信息或者保证对方能够收到你的信息。

通常，我们平时说的“安全”往往只包括第一点，保密性。其实后面两点也是很重要的，特别是在信息安全领域。如果没有完整性和可获得性，光保密又有什么用呢？回到我们的主题，SSL和数字证书主要关注的是前两点。至于可获得性就需要涉及到硬件，管理等等了。

**认证与授权**

现在的问题是，我们如何保证上面所定义的“安全”？通常，有以下两个方法：

1. **认证（Authentication）**   
   认证是证明你就是那个你所声称的那个人。举个例子，你说你是张三，然后去机场登机，机场工作人员怎么知道你就是张三呢？你必须出示你的身份证或者护照，这样就可以证明你就是那个你所声称的张三。在信息安全领域也一样，比如你想去[Google](http://www.linuxde.net/tag/google" \o "Google" \t "http://blog.csdn.net/jhonguy/article/details/_blank)查看zhlmmc的邮件，然后google会问你要zhlmmc的密码，因为只有zhlmmc知道zhlmmc帐户的密码，如果你能说出那个密码，那么你就是zhlmmc，google就会把zhlmmc的邮件返回给你。有些文章把这个过程称为Identification。
2. **授权（Authorization）**   
   授权是指一个系统里面有很多用户，有些用户能做某些事情，有些用户不能做某些事情。比如[Linux](http://www.linuxde.net/" \o "Linux" \t "http://blog.csdn.net/jhonguy/article/details/_blank)，很多用户可以同时通过认证而登录到Linux主机，但是只有root才能修改或删除系统文件，普通用户只能修改自己的home。

这里每一点都可能涉及很多不同的技术来保证过程的顺利进行。“授权”跟业务逻辑的牵扯比较大，SSL和数字证书更多的关注第一点。

### **三、加密与算法**

加密是保护信息安全的常用手段之一。对信息的加密是需要加密算法的，如果加密算法被破解了，那么一切免谈。不过，基本上，要破解一个加密算法是非常非常困难的。至少，目前流行的加密算法还是安全的，所以我们也就不必考虑这个问题了。

**散列（Hash）**

经常用bt下载的人应该很熟悉这个。这就是MD5啊~虽然Hash不只是MD5，常见的还有SHA1。不过MD5最流行所以一般大家说的hash就是它了。值得一提的是，山东大学的王小云在2005年的时候发了一篇“ [How to Break MD5 and Other Hash Functions](http://www.infosec.sdu.edu.cn/paper/md5-attack.pdf" \t "http://blog.csdn.net/jhonguy/article/details/_blank)”引起了信息安全界的轰动。虽然我没仔细研读过这篇paper，不过我相信按照paper里面的说法要破解MD5还是很费劲的，要不早就出乱子了。所以我们就不考虑这个问题了。那么究竟什么是MD5呢？我来简单解释一下。

Hash就是一个工具，能把任意大小的文档变成一个 **固定大小**（MD5是32个字符）的字符串。并且，这个过程是 **不可逆**的，也就是说，没有任何办法从那个字符串得到原来那个文档。还有很重要的一点是， **任意两个文档（哪怕极其相似）得到相同字符串的概率几乎等于0**。现在你有一个10000字的文章，发给你的朋友，那你的朋友怎么判断他收到的文章一个标点符号都没有少呢？你在发送文章的同时把这个文章的Hash字符串也发过去，这样你的朋友收到文章以后，根据收到的文章重新计算一遍这个字符串，如果这个字符串和你发过去的一样，那就证明你朋友收到的文章是和你发送的一模一样。

**对称加密（Symmetric Cryptography）**

所谓加密就是把一段能看懂的东西通过某种变换变成看不懂的东西。当然这种变换是可逆的，否则加密有什么用啊！这里所说的“变换”就是加密算法。目前我们所说的加密算法基本上都是基于密钥的。加密算法不能单独工作，必须有密钥配合。就像现实生活中的锁，同一型号的锁的原理都一样，但是没把锁都有各自的钥匙，用来开锁和关锁。 **加密的算法是公开的，但密钥是保密的**。自己“发明”加密算法是很愚蠢的，除非你是密码学专家。历史上有很多使用自己发明的加密算法的笑话，往往你发明的算法都是自以为是，其实很容易破解的拉。而目前流行的加密算法都是经过时间和众人检验的，一般情况下，只要密钥不泄露，那就是安全的。有一点要说明的是，虽然我们平时一般说“加密算法”，但往往这个加密算法都包含解密算法的。 **“对称加密”是指加密和解密的密钥是同一个。**目前流行的对称加密算法有DES，AES，Blowfish等等。举个例子，你有一篇文章想要发给你朋友，但是你不想让别人看见这篇文章所以你选择AES加密。用的密钥是你和你朋友事先约定的，只有你们两个人知道。在发送之前，你用AES算法和约定好的密钥给文章加密，然后把加密过的文章发送给你的朋友。你朋友收到以后可以用AES算法和那个密钥解密而获得原始的那篇文章。 **对称加密算法的优点是速度快，缺点是密钥管理不方便，要求共享密钥。**

**非对称加密（Asymmetric Cryptography）**

如果你理解了上面讲的对称加密，那么这里的非对称加密就很简单了。从字面上理解就可以猜到， **加密和解密不是用的同一个密钥**，其中一个称为公钥（public key），另一个称为私钥（private key）。公钥就是公开的，大家都知道，而私钥只有你自己知道。这两个密钥在数学上是有联系的，用公钥加密的内容只能由相应的私钥来解密，反过来，用私钥加密的内容只能由相应的公钥来解密。另外很重要的一点是， **不能从公钥推导出私钥，或者说很困难**。常用的非对称加密算法有RSA，ECC等等。举个例子，你想要把一篇文章发送给你的朋友，但是不想让别人看到这篇文章。除了用上面讲的方法以外，你还可以用非对称加密来实现。在发送之前，你把文章用你朋友的公钥加密（公钥是公开的，每个人都知道），然后把加密过后的文章发送给你的朋友，你的朋友可以用他的私钥来解密。其他人获得了你传送的内容都是没有用的，因为只有你朋友有私钥可以解密。 **非对称加密算法的优点是密钥管理很方便，缺点是速度慢。**

**数字签名（Digital Signature）**

我们先来看看现实生活中的签名是如何实现的。比如为信用卡账单签名，商家会打印一张消费单子给你，你看过以后觉得没有问题，于是在这张纸上签上自己的大名，表示你承认了这笔消费，并同意商家从你的信用卡账户扣钱。而商家可以对比你的签名和信用卡背后的签名是否一致来验证你是否冒用别人的信用卡（事实上很多商家不看的哦）。这个流程是基于一个假设的： **只有你自己能重现你的签名**。虽然我们不能每次都签的一摸一样，但是通过笔迹鉴定，我们可以确定这个签名是否出自你手。分析一下，签名具有哪些特点呢？

1. 不可伪造 - 通过笔记鉴定来保证。
2. 不可移植，复制 - 复印，剪贴的签名当然无效咯！
3. 不可否认 – 因为不可伪造，不可移植，不可复制，所以不可否认。

相似的，在虚拟世界里，我们有数字签名来帮助证明某个文档是你创建的，或者是你认可的。 **数字签名所用的技术是散列和非对称加密。**数字签名的假设是： **只有你自己有你的私钥**。根据前面对散列的介绍，我们先为你要签名的信息生成一个Hash字串，Hash1，然后用你的私钥加密得到Encrypted(Hash1)，这就是你对这个文档的数字签名。当别人需要验证某个文档是否是你签名的时候，只需要用你的公钥解密你的签名得到Hash1，并和该文档计算出来的Hash2对比，查看是否一致。如果一致则说明你确实对该文档签过名，否则就是没有。下面来分析一下，数字签名是如何保证上面所讲的签名的特点的。

1. **不可伪造**因为只有你有你自己的私钥，所以任何其他人都无法产生用你的私钥加密过的Hash1。
2. **不可移植，复制**你对文档A的签名不可能对文档B也有效，因为你对文档B的签名必然和对A的签名不一样，这是由Hash的唯一性保证的。拿你对A的签名去验证B是不可能通过的。
3. **不可否认**因为不可伪造，不可移植，不可复制，所以不可否认。

仔细想想数字签名和现实生活中的签名真的蛮像的，逻辑上是一样的。或许你在想，为什么要对Hash加密呢？我直接对文档用我的私钥加密不就完了嘛？对啊，效果是一样的，但是效率不一样哦~别忘了非对称算法是很慢的，加密一个100M的文件要算半天呢！

这里要顺便提一下消息认证码（ **M**essage **A**uthentication **C**ode）。 **它和数字签名很相似，只不过它是用对称加密的而数字签名用的是非对称加密。**

在现实生活中，各种加密手段往往是配合使用以达到最好的效果和效率。比如我将要介绍的SSL和数字证书，就是混合了各种的加密手段。

### 四、数字证书

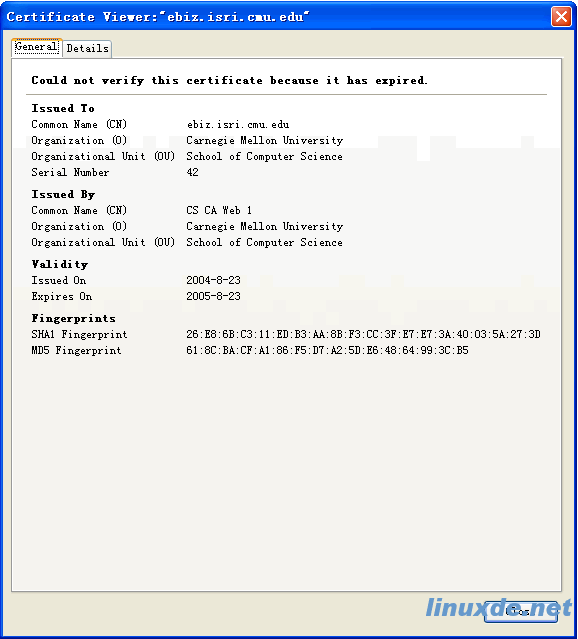
上面讲了这么多都是前戏，现在该到主题了。前面提到的认证（Authentication）的时候说，现实生活中可以用身份证和护照来证明身份， **那么在虚拟世界里，数字证书就是身份证**。和现实生活不同的是，并不是每个上网的用户都有数字证书的，往往只有当一个人需要证明自己的身份的时候才需要用到数字证书。那么什么时候需要证明自己的身份呢？普通用户一般是不需要的，网站并不关心是谁访问了网站，现在的网站只关心流量啊~反过来，网站就需要证明自己的身份了。比如你想要提交信用卡信息给预定航班的网站，那么你如何确定你正在访问的网站就是你所想要访问的那个呢？现在 [钓鱼网站](http://en.wikipedia.org/wiki/Phishing" \t "http://blog.csdn.net/jhonguy/article/details/_blank)很多的。比如你想访问的是“www.ctrip.com”，但其实你访问的是“www.otrip.com”，所以在提交自己的信息之前你需要验证一下网站的身份，要求网站出示数字证书。一般正常的网站都会主动出示自己的数字证书。由于证书在网页浏览中最为常见，所以我下面举的例子都是基于[浏览器](http://www.linuxde.net/tag/%e6%b5%8f%e8%a7%88%e5%99%a8" \o "浏览器" \t "http://blog.csdn.net/jhonguy/article/details/_blank)的。

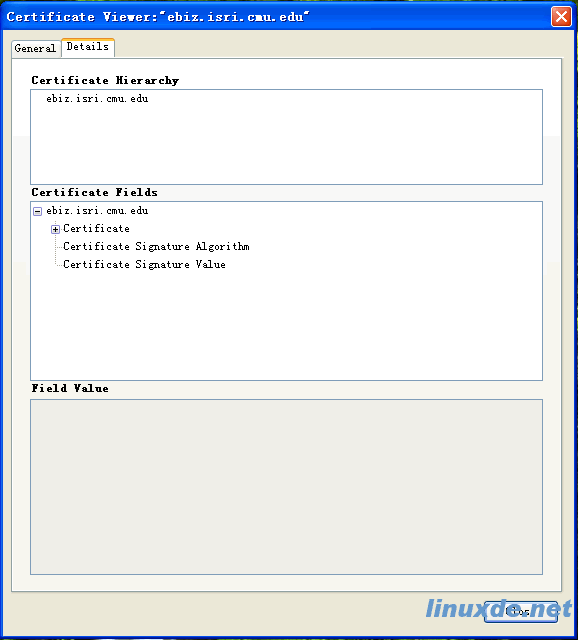
**数字证书的构成**

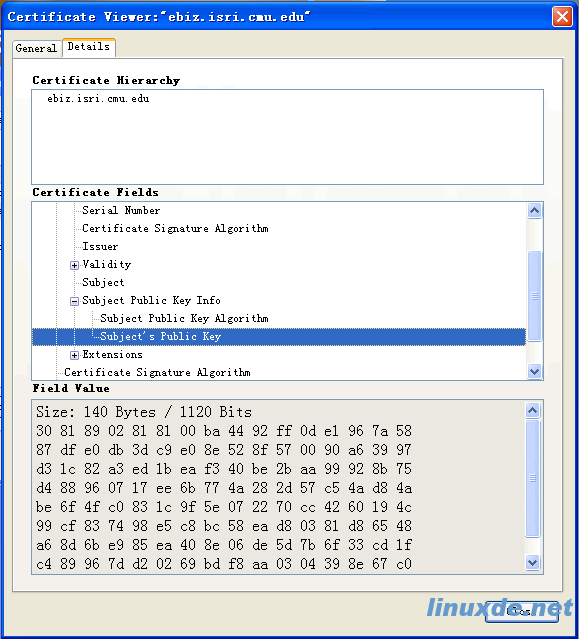
我们的身份证是由公安机关颁发的，并加有很多防伪技术，不能伪造（或者说很难）。同样的，数字证书也有专门的发证机关（Certificate Authority，简称CA，其实是一些商业公司啦）。比较常见的发证机关是 [VeriSign](http://www.verisign.com/" \t "http://blog.csdn.net/jhonguy/article/details/_blank)。数字证书的发证机关会对自己发放的证书加上自己的数字签名，以保证证书不能被伪造。那数字证书到底包含了些什么呢？

* 持有者姓名（Common Name）
* 发证机关（Issuer）
* 有效日期（Validity）
* 证书持有人的公钥（Subject’s Public Key Info）
* 扩展信息 （Extension）
* 用发证机关对该证书的数字签名（Certificate Signature）

基本信息就这些了（这些信息会在后面的章节有所解释），为了更清晰的说明问题，来几张截图：

   
图1：数字证书的基本内容

   
图2：数字证书的结构

   
图3：数字证书详细内容

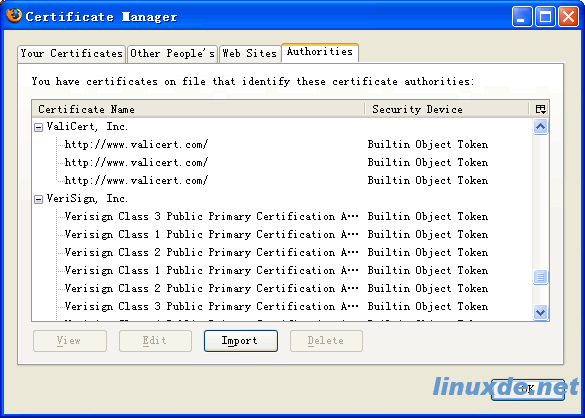
这几张截图都是我从Firefox里面拷贝出来的，这张证书已经过期了，不过不影响理解。从图2我们可以看到，Certificate（证书）和Signature（签名）是分开的，但其实这个Signature也是证书的一部分。可以这么理解，数字证书包含证书主体和数字签名。证书中的签名是对证书主体的签名。

**如何验证数字证书？**

好了，现在我们有了虚拟世界的身份证了，那如何使用呢？和现实生活中检查身份证一样，包含三个步骤：

**1.    检查身份证防伪标记**

数字证书的防伪标记就是发证机关的私钥加密的那段内容。如何验证？首先我们是默认拥有发证机关的公钥的。如果是浏览器的话，常见的发证机关的公钥是内置的。如下图所示：

   
图4：Firefox内置的发证机关的数字证书

虽然Firefox内置的是数字证书，但是有数字证书就有公钥，所以是一样的。当浏览器拿到一个数字证书，先看发证机关，然后找到相应的发证机关的证书，获得发证机关的公钥，用此公钥解密被加密的MD5，这样就获得了此证书的MD5值，我们称它为Hash1。然后浏览器用MD5算法对此证书重新计算一遍MD5，获得Hash2。然后比较Hash1和Hash2是否相等。如果相等就证明这张证书是由发证机关颁发的，并且没有被篡改过。回过头去看看上面讲Hash和MD5的部分，你应该能想明白为什么的。

**2.    核对相貌**

在现实生活中，你的身份证只有一张，你应该好好保管不被别人拿到。但难免钱包丢了，身份证跟着遭殃。所以我们在验证完身份证的真假之后我们要验证持证的人，和身份证上所声明的那个人是不是同一个，我们往往通过比较相貌来辨别。那在虚拟世界又是怎样的呢？你应该已经发现，任何人都可以拥有你的证书就像我们装的Firefox就有很多发证机关的证书。所以核对持有证书人的身份就很重要了。这就要依赖证书里面包含的公钥了。此公钥是这张证书所有者的公钥（注意，我这里指的是所有者，而不是持有者！），我们用此公钥加密一段信息发送给证书的持有者，如果持有者能发送回（可以是被私钥加密，也可以是明文，没有关系）被加密的这段信息的话就证明该持有者拥有该证书对应的私钥，也就是说，该持有者就是该证书的所有者。

**3.    核对姓名**

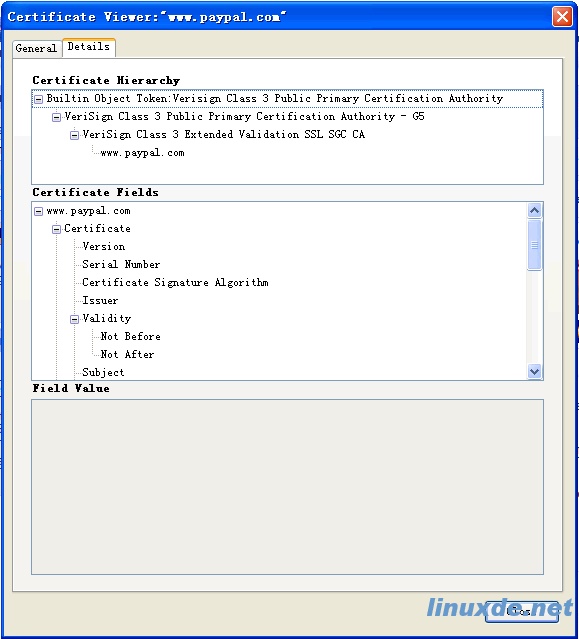
最后一步，也是最重要的一步。看清楚了，站在你面前的人的名字和登记在册的名字一样。举个例子，我拿着护照去机场登机，护照和人都没有问题，问题是我根本没有买机票。如果机场工作人员只核对了前面两步的话，我就可以登机了，岂不是很荒谬？同样的，在虚拟世界，比如那个“ctrip”的例子，你拿到了一个证书，并且验证没有问题，但是证书上的Common Name明明写的是“otrip”，你还继续吗？

如果这三个步骤都没有问题，你就可以确信正在和你通信的对方是可以信任的，是你想要联系的那个人。

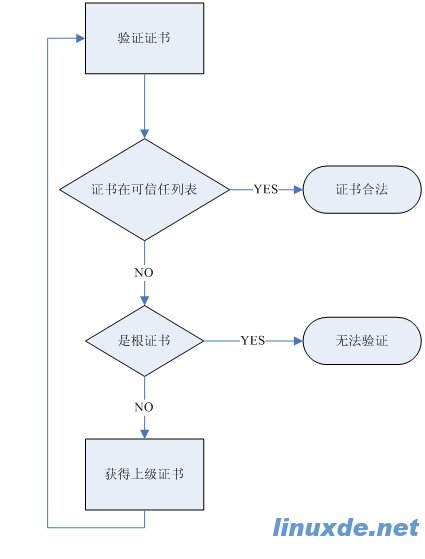
**数字证书的级联（Certificate Chain）**

根据上述讨论，我们可以知道，所有数字证书都是基于另外一张默认为可信任（浏览器内置）的数字证书的。也就是说，我们必须用一张已知合法的数字证书去验证另外一张未知的数字证书。第二节提到的发证机关的数字证书就是默认为可信任的。事实上，发证机关的证书是自己签发给自己的，验证没有意义。因为这些证书是人工配置在我们电脑上的，所以默认为安全的。这些证书称为“根证书”。

由于申请证书的人数众多，发证机关忙不过来，需要一些代理来帮忙签发证书，有可能代理也需要代理来帮忙。这样就产生了证书的层级关系，如下图所示：

   
图5：级联的数字证书

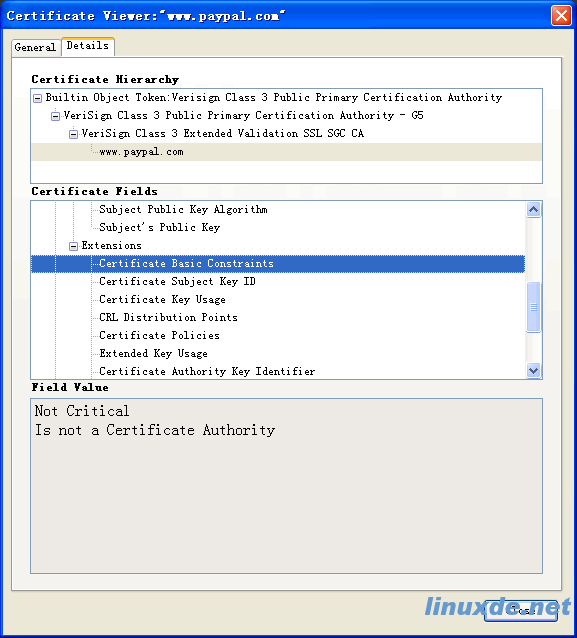
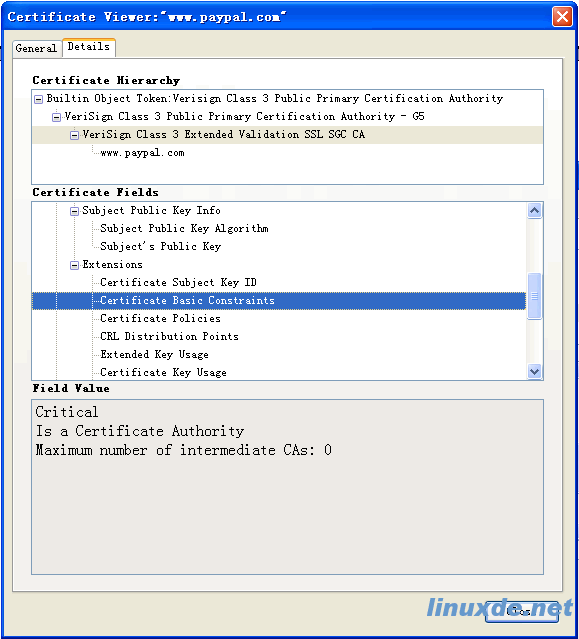
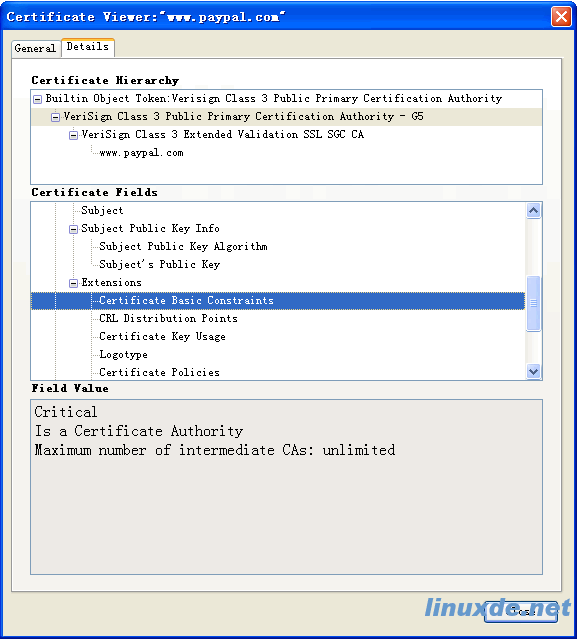
这里的“www.paypal.com”是由二级代理“VeriSign Class 3 Extended Validation SSL SGC CA”签发的，而二级代理的证书又是由一级代理“VeriSign Class 3 Public Primary Certification Authority – G5”签发的，而一级代理的证书是由根证书机关“Buildin Object Token: Verisign Class 3 Public Primary Certification Authority”签发的。不必太关注这里的名字，名字而已，重要的是他们各自的位置。在验证这张证书的时候需要从下往上递归验证。先验证用户证书（最下面的证书，这里就是“www.paypal.com”），如果这样证书在浏览器的可信任列表里面那么验证到此结束，如果不是的话就要检查证书的防伪标记，这需要用到二级代理的证书，同样的，如果二级代理的证书在可信任列表里面，那么直接使用，否则就要检查二级代理证书的防伪标记，这需要用到一级代理的证书……直到根证书为止，如果根证书不在可信任列表里面，那么这张证书就没法验证了。这个过程的简单流程图是这样的：

   
图6：级联证书的验证流程

需要注意的是，这是一个递归过程，所以这里的返回只是返回到上层递归。对于我们这个例子中的级联证书的验证可能是这样的：

IMG_262   
图7：级联证书验证流程示例

这个过程看起来没有什么问题。但是仔细一想，发现有一个大问题。任何一个拥有合法证书的人都可以给别人签发证书了，不就是在这个继承关系上面多加一层吗？黄粱大梦！事情没那么简单啦~能够签发证书的叫CA，不管是否是代理，它都是CA，只有CA的证书才能拥有下级，那如何判断一张证书是否是CA呢？还记得我们在第一节讲过的证书中包含的扩展信息吗？这里可以放很多东西，包括这张证书的合法用途，如下图所示：

   
   
   
图8：证书的扩展信息

这里我们很清楚的可以看到用户证书是不能作为CA来用的，白纸黑字写着“Is not a Certificate Authority”。而CA的证书也有不同的权限。“Maximum number of intermediate CAs”说明了这个CA的权限大小。如果这个数字是0（比如这里的二级代理的证书），说明这个CA只能签发用户证书，而不能再授权其他机构为CA。如果这个数字是1，说明这个CA可以再授权一层代理，以此类推。图中没有显示根证书的Extension，因为根证书没有Extension，没有表示没有限制。在验证级联证书的时候，浏览器会检查证书的Extension，如果某个证书超范围使用了，浏览器会发出警告的。级联证书为证书的签发带来了很大的方便和灵活性。事实上我们可以看到，任何一张用户证书都应该是级联的，至少拥有一个上级证书（根证书）。所以在我们前面的例子中的“ebiz.isir.cmu.edu”这张证书是很奇怪的，不是标准的证书。这张证书是无法验证的，只在学校内部网上使用，默认接收就行了（浏览器会把它当成根证书来处理）。

关于数字证书，我还有几点想要提一下：

1. 数字证书本身不加密，加密的是数字证书的hash。数字证书加密了就很麻烦了，如何获知发证机关呢？不知道发证机关就不知道解密的公钥啊！还有，明文的数字证书可以显示数字证书的信息，即使不能验证数字证书的真伪，但是给了人为判断一个机会。
2. 数字证书中的公钥可以是自己指定的，也可以是发证机关生成的。不同的发证机关可能有不同的要求。
3. 申请证书的过程是安全的。如果申请证书的过程都不安全的话，后面的一切都免谈了。现实生活中，申请数字证书往往要求邮寄，或者电话，传真，甚至当面申请的。

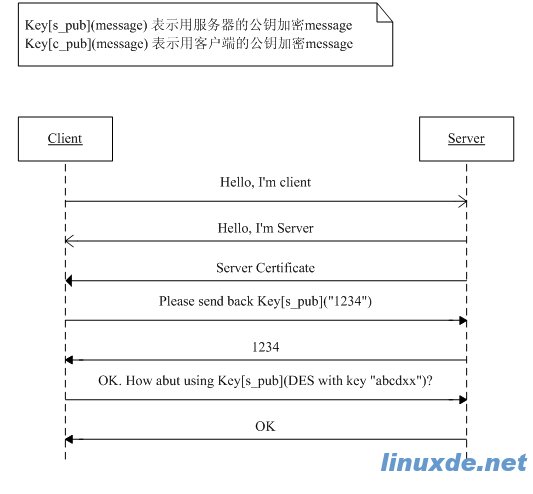
### 五、SSL的基本原理

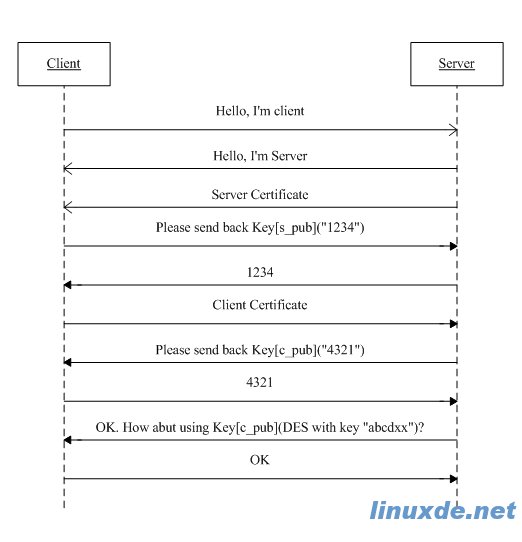
现在回到我们最原始的问题，由于Internet的架构问题，信息在网络上传输是很容易被别人获取的，那如何建立一个安全的传输网络呢？前面我们讨论了很多保证信息安全的技术，而SSL就是建立在这些技术的基础上的一套协议，用来保证通信的安全。SSL全称是 **S**ecure **S**ockets **L**ayer，它是一种间于传输层（比如TCP/IP）和应用层（比如HTTP）的协议。具体的SSL协议很复杂，我这里只讲一个大概。

最简单的方法来保证通信安全是用非对称加密。我们前面讲过数字证书的认证，如果双方都认证了对方的数字证书，那么每次传输信息的时候都用对方的公钥加密，这样就只有对方能解密，从而保证了信息的安全。但是对于日常应用（比如网页浏览）有两个问题：

1. **非对称加密速度缓慢，消耗资源**  
   如果客户端和[服务器](http://www.linuxde.net/tag/%e6%9c%8d%e5%8a%a1%e5%99%a8" \o "服务器" \t "http://blog.csdn.net/jhonguy/article/details/_blank)之间传输文件用非对称加密的话，速度一定慢的忍无可忍。
2. **不可能要求每个用户都去申请数字证书**  
   申请数字证书是一个相当麻烦的过程，要求每个上网的用户都拥有证书是不可能的事情。

SSL通过“握手协议”和“传输协议”来解决上述问题。握手协议是基于非对称加密的，而传输协议是基于对称加密的。根据不同的应用，SSL对证书的要求也是不一样的，可以是单方认证（比如HTTP, FTP），也可以是双方认证（比如网上银行）。通常情况下，服务器端的证书是一定要具备的，客户端的证书不是必须的。下面两张图片显示了SSL握手的过程。

   
图9：SSL握手，单方服务器认证

   
图10：SSL握手，双方认证

握手协议可以看成是客户端和服务器协商的一个过程，结果就是一个对称密钥，然后就进入了传输协议的部分。也许，你在想，“现在还不简单吗？用这个对称密钥加密传输数据呗！”。否，没那么简单。先来看一下结果，等会儿再解释原因。在通信双方协商出一个对称密钥以后，他们用这个密钥来加密传输的数据。同时为每个消息生成时间戳，用此密钥为消息和相应的时间戳生成消息认证码（MAC）。也就是说，每次发送的内容包括 Encrypt(message) + MAC(message + timestamp)

这么做有几个好处：

1. 防止消息的篡改

所谓消息篡改就是有第三者插在通信双方之间，篡改往来的消息。由于消息是加密的，第三者不能获得消息的内容，但是他可以闭着眼睛瞎改。如果没有MAC的话，接受者就无法判断此消息是否被篡改过。

2. 防止消息重放

消息的重放是只第三者记录下通信双方的每一次发送的消息，虽然他不能获得消息的内容。但是它可以通过重新发送客户端或者服务端的信息来把自己装成是客户端或者服务端。如果在MAC里面加上了时间戳，消息接收方验证时间戳就可以阻止消息的重放攻击。

SSL的基本思想是用非对称加密来建立链接（握手阶段），用对称加密来传输数据（传输阶段）。这样既保证了密钥分发的安全，也保证了通信的效率。

通过上面对SSL的分析，我们可以看到，SSL并不能阻止别人获得你传输的数据，但是由于你传输的数据都是加密过的，别人拿到了毫无用处，一样可以保护信息的安全。还有一点需要强调一下，SSL并不依赖于TCP，它可以建立在任何可靠的传输层协议（比如TCP）之上。也就是说SSL是不能建立在UDP之上的。这是显然的，如果传输都不可靠，偶尔丢两个包或者包的顺序换一换的话，怎么保证安全呢？

http://www.iplaysoft.com/encrypt-arithmetic.html