# 一个故事教你看懂什么是数字证书，它的原理是什么?它的作用是什么？

原创地址：http://www.cnblogs.com/JeffreySun/archive/2010/06/24/1627247.html

**1**、基础知识

这部分内容主要解释一些概念和术语，最好是先理解这部分内容。

**1.1**、公钥密码体制**(public-key cryptography)**

公钥密码体制分为三个部分，公钥、私钥、加密解密算法，它的加密解密过程如下：

- 加密：通过加密算法和公钥对内容(或者说明文)进行加密，得到密文。加密过程需要用到公钥。
- 解密：通过解密算法和私钥对密文进行解密，得到明文。解密过程需要用到解密算法和私钥。注意，由公钥加密的内容，只能由私钥进行解密，也就是说，由公钥加密的内容，如果不知道私钥，是无法解密的。

公钥密码体制的公钥和算法都是公开的(这是为什么叫公钥密码体制的原因)，私钥是保密的。大家都以使用公钥进行加密，但是只有私钥的持有者才能解密。在实际的使用中，有需要的人会生成一对公钥和私钥，把公钥发布出去给别人使用，自己保留私钥。

**1.2**、对称加密算法**(symmetric key algorithms)**

在对称加密算法中，加密使用的密钥和解密使用的密钥是相同的。也就是说，加密和解密都是使用的同一个密钥。因此对称加密算法要保证安全性的话，密钥要做好保密，只能让使用的人知道，不能对外公开。这个和上面的公钥密码体制有所不同，公钥密码体制中加密是用公钥，解密使用私钥，而对称加密算法中，加密和解密都是使用同一个密钥，不区分公钥和私钥。

// 密钥，一般就是一个字符串或数字，在加密或者解密时传递给加密/解密算法。前面在公钥密码体制中说到的公钥、私钥就是密钥，公钥是加密使用的密钥，私钥是解密使用的密钥。

**1.3**、非对称加密算法**(asymmetric key algorithms)**

在非对称加密算法中，加密使用的密钥和解密使用的密钥是不相同的。前面所说的公钥密码体制就是一种非对称加密算法，他的公钥和是私钥是不能相同的，也就是说加密使用的密钥和解密使用的密钥不同，因此它是一个非对称加密算法。

**1.4**、**RSA**简介

RSA是一种公钥密码体制，现在使用得很广泛。如果对RSA本身有兴趣的，后面看我有没有时间写个RSA的具体介绍。

RSA密码体制是一种公钥密码体制，公钥公开，私钥保密，它的加密解密算法是公开的。 由公钥加密的内容可以并且只能由私钥进行解密，并且由私钥加密的内容可以并且只能由公钥进行解密。也就是说，**RSA**的这一对公钥、私钥都可以用来加密和解密，并且一方加密的内容可以由并且只能由对方进行解密。

**1.5**、签名和加密

我们说加密，是指对某个内容加密，加密后的内容还可以通过解密进行还原。 比如我们把一封邮件进行加密，加密后的内容在网络上进行传输，接收者在收到后，通过解密可以还原邮件的真实内容。

这里主要解释一下签名，签名就是在信息的后面再加上一段内容，可以证明信息没有被修改过，怎么样可以达到这个效果呢？一般是对信息做一个hash计算得到一个hash值，注意，这个过程是不可逆的，也就是说无法通过hash值得出原来的信息内容。在把信息发送出去时，把这个hash值加密后做为一个签名和信息一起发出去。 接收方在收到信息后，会重新计算信息的hash值，并和信息所附带的hash值(解密后)进行对比，如果一致，就说明信息的内容没有被修改过，因为这里hash计算可以保证不同的内容一定会得到不同的hash值，所以只要内容一被修改，根据信息内容计算的hash值就会变化。当然，不怀好意的人也可以修改信息内容的同时也修改hash值，从而让它们可以相匹配，为了防止这种情况，hash值一般都会加密后(也就是签名)再和信息一起发送，以保证这个hash值不被修改。至于如何让别人可以解密这个签名，这个过程涉及到数字证书等概念，我们后面在说到数字证书时再详细说明，这里您先只需先理解签名的这个概念。

**2**、一个加密通信过程的演化

我们来看一个例子，现在假设“服务器”和“客户”要在网络上通信，并且他们打算使用RSA(参看前面的RSA简介)来对通信进行加密以保证谈话内容的安全。由于是使用RSA这种公钥密码体制，“服务器”需要对外发布公钥(算法不需要公布，RSA的算法大家都知道)，自己留着私钥。“客户”通过某些途径拿到了“服务器”发布的公钥，客户并不知道私钥。“客户”具体是通过什么途径获取公钥的，我们后面再来说明，下面看一下双方如何进行保密的通信：

**2.1** 第一回合：

“客户”->“服务器”：你好

“服务器”->“客户”：你好，我是服务器

“客户”->“服务器”：？？？？

因为消息是在网络上传输的，有人可以冒充自己是“服务器”来向客户发送信息。例如上面的消息可以被黑客截获如下：

~~“客户”->“服务器”：你好~~

~~“服务器”->“客户”：你好，我是服务器~~

**“客户”->“黑客”**：你好　　// 黑客在“客户”和“服务器”之间的某个路由器上截获“客户”发给服务器的信息，然后自己冒充“服务器”

**“黑客”->“客户”**：你好，我是服务器

因此“客户”在接到消息后，并不能肯定这个消息就是由“服务器”发出的，某些“黑客”也可以冒充“服务器”发出这个消息。如何确定信息是由“服务器”发过来的呢？有一个解决方法，因为只有服务器有私钥，所以如果只要能够确认对方有私钥，那么对方就是“服务器”。因此通信过程可以改进为如下：

**2.2** 第二回合：

“客户”->“服务器”：你好

“服务器”->“客户”：你好，我是服务器

"客户"->"服务器"：向我证明你就是服务器

"服务器"->"客户"：你好，我是服务器 {你好，我是服务器}[私钥|RSA]

**//** 注意这里约定一下，**{}** 表示**RSA**加密后的内容，**[ | ]**表示用什么密钥和算法进行加密，后面的示例中都用这种表示方式，例如上面的 {你好，我是服务器}[私钥|RSA] 就表示用私钥对"你好，我是服务器"进行加密后的结果。

为了向"客户"证明自己是"服务器"， "服务器"把一个字符串用自己的私钥加密，把明文和加密后的密文一起发给"客户"。对于这里的例子来说，就是把字符串 "你好，我是服务器"和这个字符串用私钥加密后的内容 {你好，我是服务器}[私钥|RSA] 发给客户。

"客户"收到信息后，她用自己持有的公钥解密密文，和明文进行对比，如果一致，说明信息的确是由服务器发过来的。也就是说"客户"把 {你好，我是服务器}[私钥|RSA] 这个内容用公钥进行解密，然后和"你好，我是服务器"对比。因为由"服务器"用私钥加密后的内容，由并且只能由公钥进行解密，私钥只有"服务器"持有，所以如果解密出来的内容是能够对得上的，那说明信息一定是从"服务器"发过来的。

假设"黑客"想冒充"服务器"：

**"黑客"->"客户"**：你好，我是服务器

**"客户"->"黑客"**：向我证明你就是服务器

**"黑客"->"客户"**：你好，我是服务器 **{**你好，我是服务器**}[**？？？ **|RSA]** //这里黑客无法冒充，因为他不知道私钥，无法用私钥加密某个字符串后发送给客户去验证。

**"客户"->"黑客"**：？？？？

由于"黑客"没有"服务器"的私钥，因此它发送过去的内容，"客户"是无法通过服务器的公钥解密的，因此可以认定对方是个冒牌货！

到这里为止，"客户"就可以确认"服务器"的身份了，可以放心和"服务器"进行通信，但是这里有一个问题，通信的内容在网络上还是无法保密。为什么无法保密呢？通信过程不是可以用公钥、私钥加密吗？其实用RSA的私钥和公钥是不行的，我们来具体分析下过程，看下面的演示：

**2.3** 第三回合：

"客户"->"服务器"：你好

"服务器"->"客户"：你好，我是服务器

"客户"->"服务器"：向我证明你就是服务器

"服务器"->"客户"：你好，我是服务器 {你好，我是服务器}[私钥|RSA]

"客户"->"服务器"：{我的帐号是aaa，密码是123，把我的余额的信息发给我看看}[公钥|RSA]

"服务器"->"客户"：{你的余额是100元}[私钥|RSA]

注意上面的的信息 {你的余额是100元}[私钥]，这个是"服务器"用私钥加密后的内容，但是我们之前说了，公钥是发布出去的，因此所有的人都知道公钥，所以除了"客户"，其它的人也可以用公钥对{你的余额是100元}[私钥]进行解密。所以如果"服务器"用私钥加密发给"客户"，这个信息是无法保密的，因为只要有公钥就可以解密这内容。然而"服务器"也不能用公钥对发送的内容进行加密，因为"客户"没有私钥，发送个"客户"也解密不了。

这样问题就又来了，那又如何解决呢？在实际的应用过程，一般是通过引入对称加密来解决这个问题，看下面的演示：

**2.4** 第四回合：

"客户"->"服务器"：你好

"服务器"->"客户"：你好，我是服务器

"客户"->"服务器"：向我证明你就是服务器

"服务器"->"客户"：你好，我是服务器 {你好，我是服务器}[私钥|RSA]

"客户"->"服务器"：{我们后面的通信过程，用对称加密来进行，这里是对称加密算法和密钥}[公钥|RSA] //蓝色字体的部分是对称加密的算法和密钥的具体内容，客户把它们发送给服务器。

"服务器"->"客户"：{OK，收到！}[密钥|对称加密算法]

"客户"->"服务器"：{我的帐号是aaa，密码是123，把我的余额的信息发给我看看}[密钥|对称加密算法]

"服务器"->"客户"：{你的余额是100元}[密钥|对称加密算法]

在上面的通信过程中，"客户"在确认了"服务器"的身份后，"客户"自己选择一个对称加密算法和一个密钥，把这个对称加密算法和密钥一起用公钥加密后发送给"服务器"。注意，由于对称加密算法和密钥是用公钥加密的，就算这个加密后的内容被"黑客"截获了，由于没有私钥，"黑客"也无从知道对称加密算法和密钥的内容。

由于是用公钥加密的，只有私钥能够解密，这样就可以保证只有服务器可以知道对称加密算法和密钥，而其它人不可能知道(这个对称加密算法和密钥是"客户"自己选择的，所以"客户"自己当然知道如何解密加密)。这样"服务器"和"客户"就可以用对称加密算法和密钥来加密通信的内容了。

总结一下，RSA加密算法在这个通信过程中所起到的作用主要有两个：

- 因为私钥只有**"**服务器**"**拥有，因此**"**客户**"**可以通过判断对方是否有私钥来判断对方是否是**"**服务器**"**。
- 客户端通过**RSA**的掩护，安全的和服务器商量好一个对称加密算法和密钥来保证后面通信过程内容的安全。

如果这里您理解了为什么不用RSA去加密通信过程，而是要再确定一个对称加密算法来保证通信过程的安全，那么就说明前面的内容您已经理解了。(如果不清楚，再看下2.3和2.4，如果还是不清楚，那应该是我们说清楚，您可以留言提问。)

到这里，"客户"就可以确认"服务器"的身份，并且双方的通信内容可以进行加密，其他人就算截获了通信内容，也无法解密。的确，好像通信的过程是比较安全了。

但是这里还留有一个问题，在最开始我们就说过，"服务器"要对外发布公钥，那"服务器"如何把公钥发送给"客户"呢？我们第一反应可能会想到以下的两个方法：

a)把公钥放到互联网的某个地方的一个下载地址，事先给"客户"去下载。

b)每次和"客户"开始通信时，"服务器"把公钥发给"客户"。

但是这个两个方法都有一定的问题，

对于a)方法，"客户"无法确定这个下载地址是不是"服务器"发布的，你凭什么就相信这个地址下载的东西就是"服务器"发布的而不是别人伪造的呢，万一下载到一个假的怎么办？另

外要所有的“客户”都在通信前事先去下载公钥也很不现实。

对于b)方法，也有问题，因为任何人都可以自己生成一对公钥和私钥，他只要向“客户”发送他自己的私钥就可以冒充“服务器”了。示意如下：

“客户”->“黑客”：你好　　//黑客截获“客户”发给“服务器”的消息

“黑客”->“客户”：你好，我是服务器，这个是我的公钥　//黑客自己生成一对公钥和私钥，把公钥发给“客户”，自己保留私钥

“客户”->“黑客”：向我证明你就是服务器

“黑客”->“客户”：你好，我是服务器 {你好，我是服务器}[黑客自己的私钥|RSA]　　//客户收到“黑客”用私钥加密的信息后，是可以用“黑客”发给自己的公钥解密的，从而会误认为“黑客”是“服务器”

因此“黑客”只需要自己生成一对公钥和私钥，然后把公钥发送给“客户”，自己保留私钥，这样由于“客户”可以用黑客的公钥解密黑客的私钥加密的内容，“客户”就会相信“黑客”是“服务器”，从而导致了安全问题。这里问题的根源就在于，大家都可以生成公钥、私钥对，无法确认公钥对到底是谁的。 如果能够确定公钥到底是谁的，就不会有这个问题了。例如，如果收到“黑客”冒充“服务器”发过来的公钥，经过某种检查，如果能够发现这个公钥不是“服务器”的就好了。

为了解决这个问题，数字证书出现了，它可以解决我们上面的问题。先大概看下什么是数字证书，一个证书包含下面的具体内容：

- 证书的发布机构
- 证书的有效期
- 公钥
- 证书所有者（Subject）
- 签名所使用的算法
- 指纹以及指纹算法

证书的内容的详细解释会在后面详细解释，这里先只需要搞清楚一点，数字证书可以保证数字证书里的公钥确实是这个证书的所有者**(Subject)**的，或者证书可以用来确认对方的身份。也就是说，我们拿到一个数字证书，我们可以判断出这个数字证书到底是谁的。至于是如何判断的，后面会在详细讨论数字证书时详细解释。现在把前面的通信过程使用数字证书修改为如下：

**2.5** 第五回合：

“客户”->“服务器”：你好

“服务器”->“客户”：你好，我是服务器，这里是我的数字证书　　//这里用证书代替了公钥

“客户”->“服务器”：向我证明你就是服务器

“服务器”->“客户”：你好，我是服务器 {你好，我是服务器}[私钥|RSA]

注意，上面第二次通信，“服务器”把自己的证书发给了“客户”，而不是发送公钥。“客户”可以根据证书校验这个证书到底是不是“服务器”的，也就是能校验这个证书的所有者是不是“服务器”，从而确认这个证书中的公钥的确是“服务器”的。后面的过程和以前是一样，“客户”让“服务器”证明自己的身份，“服务器”用私钥加密一段内容连同明文一起发给“客户”，“客户”把加密内容用数字证书中的公钥解密后和明文对比，如果一致，那么对方就确实是“服务器”，然后双方协商一个对称加密来保证通信过程的安全。到这里，整个过程就完整了，我们回顾一下：

**2.6** 完整过程：

**step1**：“客户”向服务端发送一个通信请求

“客户”->“服务器”：你好

**step2**：“服务器”向客户发送自己的数字证书。证书中有一个公钥用来加密信息，私钥由“服务器”持有

“服务器”->“客户”：你好，我是服务器，这里是我的数字证书

**step3**：“客户”收到“服务器”的证书后，它会去验证这个数字证书到底是不是“服务器”的，数字证书有没有什么问题，数字证书如果检查没有问题，就说明数字证书中的公钥确实是“服务器”的。检查数字证书后，“客户”会发送一个随机的字符串给“服务器”用私钥去加密，服务器把加密的结果返回给“客户”，“客户”用公钥解密这个返回结果，如果解密结果与之前生成的随机字符串一致，那说明对方确实是私钥的持有者，或者说对方确实是“服务器”。

“客户”->“服务器”：向我证明你就是服务器，这是一个随机字符串　　//前面的例子中为了方便解释，用的是“你好”等内容，实际情况下一般是随机生成的一个字符串。

“服务器”->“客户”：{一个随机字符串}[私钥|RSA]

**step4**：验证“服务器”的身份后，“客户”生成一个对称加密算法和密钥，用于后面的通信的加密和解密。这个对称加密算法和密钥，“客户”会用公钥加密后发送给“服务器”，别人截获了也没用，因为只有“服务器”手中有可以解密的私钥。这样，后面“服务器”和“客户”就都可以用对称加密算法来加密和解密通信内容了。

“服务器”->“客户”：{OK，已经收到你发来的对称加密算法和密钥！有什么可以帮到你的？}[密钥|对称加密算法]

“客户”->“服务器”：{我的帐号是aaa，密码是123，把我的余额的信息发给我看看}[密钥|对称加密算法]

“服务器”->“客户”：{你好，你的余额是100元}[密钥|对称加密算法]

…… //继续其它的通信

**2.7** 其它问题：

上面的过程已经十分接近HTTPS的真实通信过程了，完全可以按照这个过程去理解HTTPS的工作原理。但是我为了方便解释，上面有些细节没有说到，有兴趣的人可以看下这部分的内容。可以跳过不看，无关紧要。

【问题1】

上面的通信过程中说到，在检查完证书后，“客户”发送一个随机的字符串给“服务器”去用私钥加密，以便判断对方是否真的持有私钥。但是有一个问题，“黑客”也可以发送一个字符串给“服务器”去加密并且得到加密后的内容，这样对于“服务器”来说是不安全的，因为黑客可以发送一些简单的有规律的字符串给“服务器”加密，从而寻找加密的规律，有可能威胁到私

钥的安全。所以说，“服务器”随随便便用私钥去加密一个来路不明的字符串并把结果发送给对方是不安全的。

〖解决方法〗

每次收到“客户”发来的要加密的的字符串时，“服务器”并不是真正的加密这个字符串本身，而是把这个字符串进行一个hash计算，加密这个字符串的hash值(不加密原来的字符串)后发送给“客户”，“客户”收到后解密这个hash值并自己计算字符串的hash值然后进行对比是否一致。也就是说，“服务器”不直接加密收到的字符串，而是加密这个字符串的一个hash值，这样就避免了加密那些有规律的字符串，从而降低被破解的机率。“客户”自己发送的字符串，因此它自己可以计算字符串的hash值，然后再把“服务器”发送过来的加密的hash值和自己计算的进行对比，同样也能确定对方是否是“服务器”。

【问题2】

在双方的通信过程中，“黑客”可以截获发送的加密了的内容，虽然他无法解密这个内容，但是他可以捣乱，例如把信息原封不动的发送多次，扰乱通信过程。

〖解决方法〗

可以给通信的内容加上一个序号或者一个随机的值，如果“客户”或者“服务器”接收到的信息中有之前出现过的序号或者随机值，那么说明有人在通信过程中重发信息内容进行捣乱，双方会立刻停止通信。有人可能会问，如果有人一直这么捣乱怎么办？那不是无法通信了？ 答案是的确是这样的，例如有人控制了你连接互联网的路由器，他的确可以针对你。但是一些重要的应用，例如军队或者政府的内部网络，它们都不使用我们平时使用的公网，因此一般人不会破坏到他们的通信。

【问题3】

在双方的通信过程中，“黑客”除了简单的重复发送截获的消息之外，还可以修改截获后的密文修改后再发送，因为修改的是密文，虽然不能完全控制消息解密后的内容，但是仍然会破坏解密后的密文。因此发送过程如果黑客对密文进行了修改，“客户”和“服务器”是无法判断密文是否被修改的。虽然不一定能达到目的，但是“黑客”可以一直这样碰碰运气。

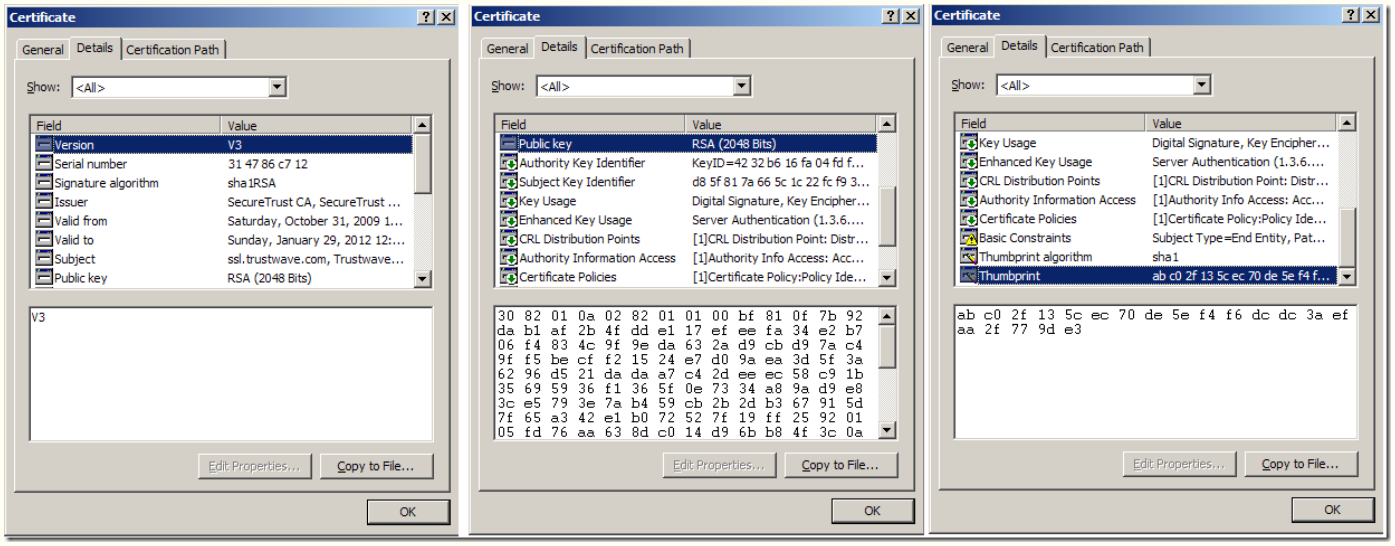
〖解决方法〗

在每次发送信息时，先对信息的内容进行一个hash计算得出一个hash值，将信息的内容和这个hash值一起加密后发送。接收方在收到后进行解密得到明文的内容和hash值，然后接收方再自己对收到信息内容做一次hash计算，与收到的hash值进行对比看是否匹配，如果匹配就说明信息在传输过程中没有被修改过。如果不匹配说明中途有人故意对加密数据进行了修改，立刻中断通话过程后做其它处理。

**3.** 证书的构成和原理

**3.1** 证书的构成和原理

之前已经大概说了一个证书由什么构成，但是没有仔细进行介绍，这里对证书的内容做一个详细的介绍。先看下一个证书到底是个什么东西，在windows下查看一个证书时，界面是这样的，我们主要关注一下Details Tab页，其中的内容比较长，我滚动内容后后抓了三个图，把完整的信息显示出来：




里面的内容比较多——Version、Serial number、Signature algorithm 等等，挑几个重要的解释一下。

◆Issuer (证书的发布机构)

指出是什么机构发布的这个证书，也就是指明这个证书是哪个公司创建的(只是创建证书，不是指证书的使用者)。对于上面的这个证书来说，就是指"SecureTrust CA"这个机构。

◆Valid from , Valid to (证书的有效期)

也就是证书的有效时间，或者说证书的使用期限。 过了有效期限，证书就会作废，不能使用了。

◆Public key (公钥)

这个我们在前面介绍公钥密码体制时介绍过，公钥是用来对消息进行加密的，第2章的例子中经常用到的。这个数字证书的公钥是2048位的，它的值可以在图的中间的那个对话框中看得到，是很长的一串数字。

◆Subject (主题)

这个证书是发布给谁的，或者说证书的所有者，一般是某个人或者某个公司名称、机构的名称、公司网站的网址等。 对于这里的证书来说，证书的所有者是Trustwave这个公司。

◆Signature algorithm (签名所使用的算法)

就是指的这个数字证书的数字签名所使用的加密算法，这样就可以使用证书发布机构的证书里面的公钥，根据这个算法对指纹进行解密。指纹的加密结果就是数字签名(第1.5节中解

释过数字签名)。

◆Thumbprint, Thumbprint algorithm (指纹(hash值)以及指纹算法(hash算法))

这个是用来保证证书的完整性的，也就是说确保证书没有被修改过，这东西的作用和2.7中说到的第3个问题类似。 其原理就是在发布证书时，发布者根据指纹算法(一个hash算法)计算整个证书的hash值(指纹)并和证书放在一起，使用者在打开证书时，自己也根据指纹算法计算一下证书的hash值(指纹)，如果和刚开始的值对得上，就说明证书没有被修改过，因为证书的内容被修改后，根据证书的内容计算的出的hash值(指纹)是会变化的。 注意，这个指纹会使用"SecureTrust CA"这个证书机构的私钥用签名算法(Signature algorithm)加密后和证书放在一起。

注意，为了保证安全，在证书的发布机构发布证书时，证书的指纹和指纹算法，都会加密后再和证书放到一起发布，以防有人修改指纹后伪造相应的数字证书。这里问题又来了，证书的指纹和指纹算法用什么加密呢？他们是用证书发布机构的私钥进行加密的。可以用证书发布机构的公钥对指纹和指纹算法解密，也就是说证书发布机构除了给别人发布证书外，他自己本身也有自己的证书。证书发布机构的证书是哪里来的呢？？？这个证书发布机构的数字证书(一般由他自己生成)在我们的操作系统刚安装好时(例如windows xp等操作系统)，这些证书发布机构的数字证书就已经被微软(或者其它操作系统的开发机构)安装在操作系统中了，微软等公司会根据一些权威安全机构的评估选取一些信誉很好并且通过一定的安全认证的证书发布机构，把这些证书发布机构的证书默认就安装在操作系统里面了，并且设置为操作系统信任的数字证书。这些证书发布机构自己持有与他自己的数字证书对应的私钥，他会用这个私钥加密所有他发布的证书的指纹作为数字签名。

### 3.2 如何向证书的发布机构去申请证书

举个例子方便大家理解，假设我们公司"ABC Company"花了1000块钱，向一个证书发布机构"SecureTrust CA"为我们自己的公司"ABC Company"申请了一张证书，注意，这个证书发布机构"SecureTrust CA"是一个大家公认并被一些权威机构接受的证书发布机构，我们的操作系统里面已经安装了"SecureTrust CA"的证书。"SecureTrust CA"在给我们发布证书时，把Issuer,Public key,Subject,Valid from,Valid to等信息以明文的形式写到证书里面，然后用一个指纹算法计算出这些数字证书内容的一个指纹，并把指纹和指纹算法用自己的私钥进行加密，然后和证书的内容一起发布，同时"SecureTrust CA"还会给一个我们公司"ABC Company"的私钥给到我们。我们花了1000块钱买的这个证书的内容如下：

××××××××××××××证书内容开始××××××××××××××××

Issuer：SecureTrust CA

Subject：ABC Company

Valid from： 某个日期

Valid to： 某个日期

Public Key：一串很长的数字

…… 其它的一些证书内容……

{证书的指纹和计算指纹所使用的指纹算法}[SecureTrust CA的私钥|RSA]　　//这个就是"SecureTrust CA"对这个证书的一个数字签名，表示这个证书确实是他发布的，有什么问题他会负责(收了我们1000块，出了问题肯定要负责任的)

××××××××××××××证书内容结束××××××××××××××××

**//** 记不记得前面的约定？**{}** 表示**RSA**加密后的内容，**[ | ]**表示用什么密钥和算法进行加密

我们"ABC Company"申请到这个证书后，我们把证书投入使用，我们在通信过程开始时会把证书发给对方，对方如何检查这个证书的确是合法的并且是我们"ABC Company"公司的证书呢？首先应用程序(对方通信用的程序，例如IE、OUTLook等)读取证书中的Issuer(发布机构)为"SecureTrust CA"，然后会在操作系统中受信任的发布机构的证书中去找"SecureTrust CA"的证书，如果找不到，那说明证书的发布机构是个水货发布机构，证书可能有问题，程序会给出一个错误信息。 如果在系统中找到了"SecureTrust CA"的证书，那么应用程序就会从证书中取出"SecureTrust CA"的公钥，然后对我们"ABC Company"公司的证书里面的指纹和指纹算法用这个公钥进行解密，然后使用这个指纹算法计算"ABC Company"证书的指纹，将这个计算的指纹与放在证书中的指纹对比，如果一致，说明"ABC Company"的证书肯定没有被修改过并且证书是"SecureTrust CA" 发布的，证书中的公钥肯定是"ABC Company"的。对方然后就可以放心的使用这个公钥和我们"ABC Company"进行通信了。

★这个部分非常重要，一定要理解，您可以重新回顾一下之前的两章**"1、基础知识"**和**" 2、一个加密通信过程的演化"**，然后再来理解这部分的内容。

### 3.3 证书的发布机构

前面已经初步介绍了一下证书发布机构，这里再深入讨论一下。

其实所有的公司都可以发布证书，我们自己也可以去注册一家公司来专门给别人发布证书。但是很明显，我们自己的专门发布证书的公司是不会被那些国际上的权威机构认可的，人家怎么知道你是不是个狗屁皮包公司？因此微软在它的操作系统中，并不会信任我们这个证书发布机构，当应用程序在检查证书的合法信的时候，一看证书的发布机构并不是操作系统所信任的发布机构，就会抛出错误信息。也就是说windows操作系统中不会预先安装好我们这个证书发布机构的证书，不信任我们这个发布机构。

不受信任的证书发布机构的危害

为什么一个证书发布机构受不受信任这么重要？我们举个例子。假设我们开了一个狗屁公司来为别人发布证书，并且我和微软有一腿，微软在他们的操作系统中把我设置为了受信任的证书发布机构。现在如果有个小公司叫Wicrosoft 花了10块钱让我为他们公司申请了一个证书，并且公司慢慢壮大，证书的应用范围也越来越广。然后有个奸商的公司JS Company想冒充Wicrosoft，于是给了我¥10000，让我为他们颁布一个证书，但是证书的名字(Subject)要写Wicrosoft，假如我为了这¥10000，真的把证书给了他们，那么他们以后就可以使用这个证书来冒充Wicrosoft了。

如果是一个优秀的证书发布机构，比如你要向他申请一个名字叫Wicrosoft的证书，它会让你提供很多资料证明你确实可以代表Wicrosoft这个公司，也就是说他回去核实你的身份。证书发布机构是要为他发布出的证书负法律责任的。

到这里，你可能会想，TMD，那我们自己就不能发布证书吗？就一定要花钱去申请？当然不是，我们自己也可以成立证书发布机构，但是需要通过一些安全认证等等，只是有点麻烦。另外，如果数字证书只是要在公司内部使用，公司可以自己给自己生成一个证书，在公司的所有机器上把这个证书设置为操作系统信任的证书发布机构的证书(这句话仔细看清楚，有点绕口)，这样以后公司发布的证书在公司内部的所有机器上就可以通过验证了(在发布证书时，把这些证书的Issuer(发布机构)设置为我们自己的证书发布机构的证书的Subject(主题)就可以了)。但是这只限于内部应用，因为只有我们公司自己的机器上设置了信任我们自己这个所谓的证书发布机构，而其它机器上并没有事先信任我们这个证书发布机构，所以在其它机器上，我们发布的证书就无法通过安全验证。

## 4. 在**windows**中对数字证书进行管理

### 4.1 查看、删除、安装 数字证书

我们在上一章中说到了，我们的操作系统中会预先安装好一些证书发布机构的证书，我们看下在windows中如何找到这些证书，步骤如下：

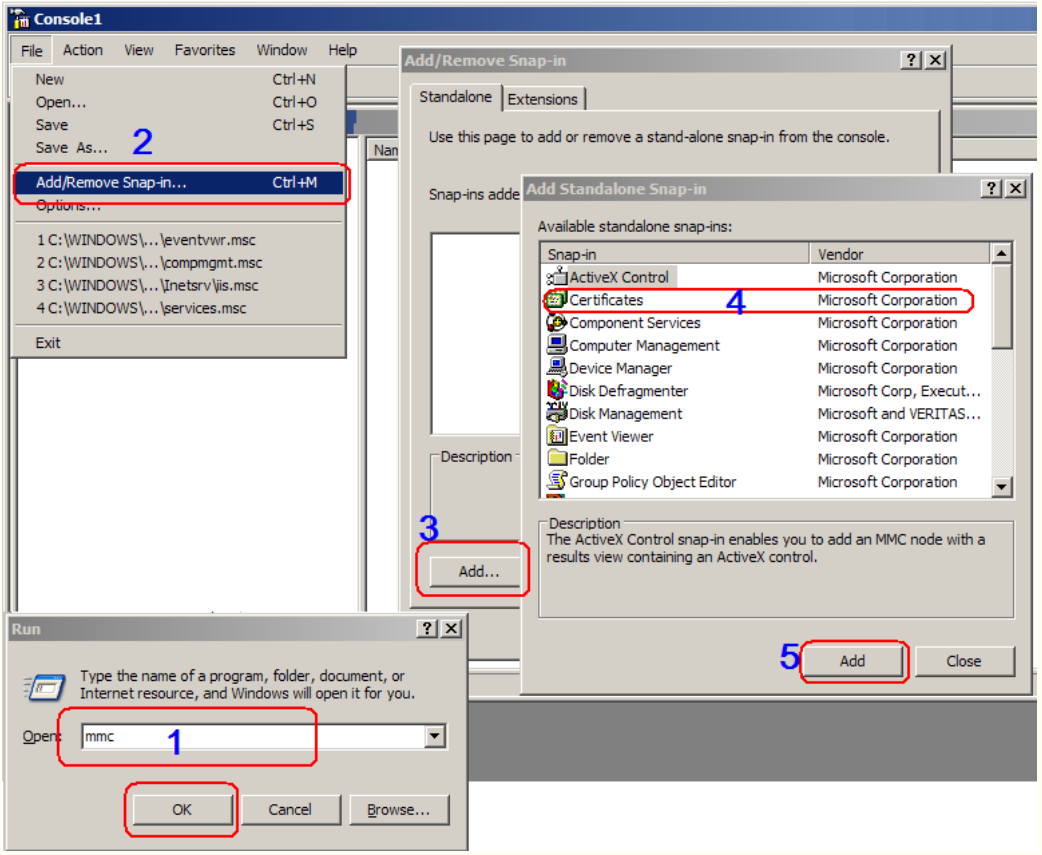
1)开始菜单->运行，输入mmc，回车

2)在打开的窗口中选择 File-> Add/Remove Snap-in…

3)然后在弹出的对话框的 Standalone Tab页里面点击 Add… 按钮

4)在弹出的对对话框中选择 certificates 后点击 Add 按钮

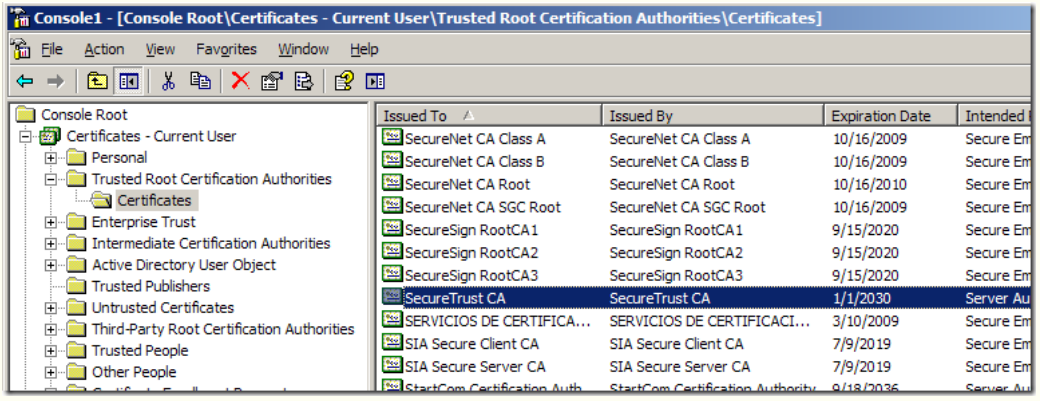
具体的步骤如下图所示：




上面的步骤结束后，会又弹出一个对话框，里面有三个单选按钮如下：

- My user account
- Service account
- Computer account

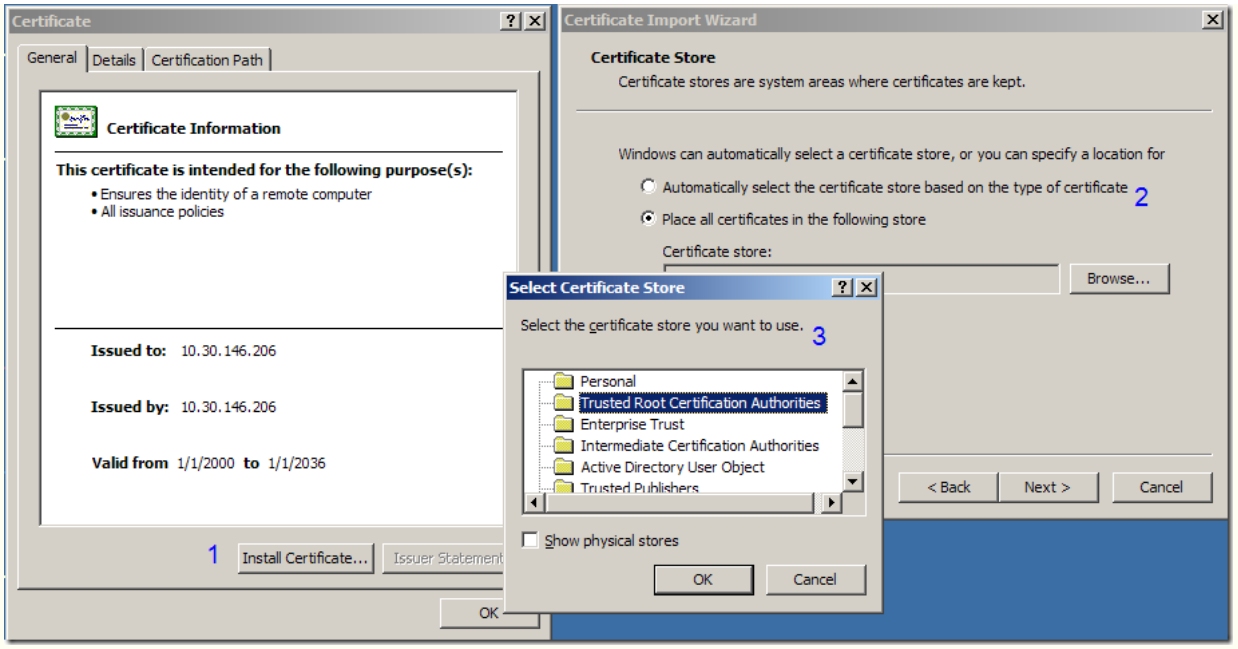
可以选择第一或者第三个选项，用来查看当前用户的证书或整个计算里面安装的证书。我们这里就默认选择第一个，平时一般安装证书的时候都会给所有用户安装，所以选择第一个和第三个选项看到的证书会差不多。我们在左边的导航树中选中受信任的证书发布机构(Trusted Root Certificate Authorities)，然后点击下面的证书(Certificates)，在右边的区域中就可以看到所有的受信任的证书发布机构的证书。




注意上面的图片中，右边我们选中的这个证书发布机构"SecureTrust CA"，我们前面在第3章3.2节中举例子的时候，就是去向这个证书发布机构申请的证书，由于我们申请的证书是这个机构发布的，所以应用程序在检查我们的证书的发布机构时(会检查我们证书的签名，确认是该机构发布的证书)，就会发现是可以信任的证书发布机构，从而就会相信我们证书的真实性。

删除数字证书很简单，直接在右边的列表中右键然后删除就可以了。

数字证书的安装也比较简单，直接双击数字证书文件，会打开数字证书，对话框下面会有一个Install Certificate按钮，点击后就可以根据向导进行安装，如下图所示：




这个证书是我自己生成的测试证书，在证书的导入向导里面，它会让你选择导入到什么位置，如果是一个我们自己信任的证书发布机构自己的证书，只要导入到Certificate Authorities就可以了。Trusted Root Certificate Authorities, Intermediate Certification Authorities, Third-Party Root Certification Authorities 都是可以的，他们只是对证书的发布机构做了一个分类，还有一些其它的证书类型，例如Personal(个人证书)等等，具体就不介绍了。安装的时候一般来说可以用默认的选择项一直"下一步"到底。

### 4.2 如何自己创建证书

每个证书发布机构都有自己的用来创建证书的工具，当然，具体他们怎么去创建一个证书的我也不太清楚，不同类型的证书都有一定的格式和规范，我没有仔细去研究过这部分内容。 微软为我们提供了一个用来创建证书的工具makecert.exe，在安装Visual Studio的时候会安装上。如果没有安装也无所谓，可以上网去下一个，搜索**makecert**就可以了。可以直接从我的博客下载，这是链接。

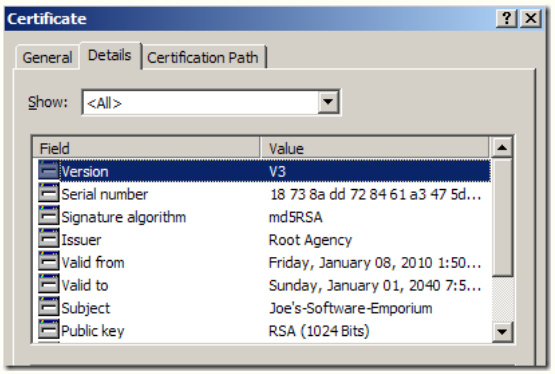
向一些正规的证书发布机构申请证书一般是要收费的(因为别人要花时间检查你的身份，确认有没有同名的证书等等)，这里我们看下如何自己创建一个证书，为后面在IIS中配置Https做准备。

我们用到的是**makecert**这个工具，微软有很详细的使用帮助，我这里只做一个简单的解释，详细的各种参数和使用方法请查看MSDN的makecert的帮助。但是里面有些参数说得不够清楚，而且还有遗漏的，可以参看我后面的解释作为一个补充。

先看下**makecert**最简单的使用方式：

**makecert.exe test.cer**

上面的命令会在makecert.exe所在的目录生成一个证书文件test.cer的数字证书文件。可以双击证书打开，看看证书的内容如下：




证书的发布机构是"Root Agency"，证书的主题(证书发布给谁)是"Joe's-Software-Emporium"，因为我们没有指定把证书发布给谁，makecert自己给我们随便生成了一个公司的名字。另外还指定了公钥、签名算法(用来解密签名)、指纹和指纹算法等。

注意，因为这个证书是由微软的工具生成的，严格来说它没什么发布机构，所以微软虚拟了一个叫做"Root Agency"的发布机构，默认情况下，windows里面安装了这个所谓的证书发布机构的证书，但是这证书默认情况下不是受信任的，原因很简单，这样做大家都可以用makecert来制作合法的数字证书了。如果我们自己硬是要，也可以把它设置为受信任的。

下面我们看下其它的参数，比如我们要给网站 www.jefferysun.com 生成一个证书MyCA.cer，假设我们把makecert.exe放在C：盘下，命令行如下：

makecert -r -pe -n "CN=10.30.146.206" -b 01/01/2000 -e 01/01/2036 -eku 1.3.6.1.5.5.7.3.1 -ss my -sr localMachine -sky exchange -sp "Microsoft RSA SChannel Cryptographic Provider" -sy 12

C:\> makecert.exe –pe -r –n "CN=www.jefferysun.com" -ss my -sr LocalMachine -a sha1 -len 2048 MyCA.cer

解释一下makecert的常用参数的意思：

- **-n** 指定主题的名字，这个是有固定的格式的， CN=主题名字 ，CN应该是Certificate Name的缩写。我这里的主题的名字就是我们的IIS所在机器的IP。这里可以指定一些主题的其它附加信息，例如 O= *** 表示组织信息等等。
- **-r** 创建自签署证书，意思就是说在生成证书时，将证书的发布机构设置为自己。

- **-pe** 将所生成的私钥标记为可导出。注意，服务器发送证书给客户端的时候，客户端只能从证书里面获取公钥，私钥是无法获取的。如果我们指定了这个参数，证书在安装在机器上后，我们还可以从证书中导出私钥，默认情况下是不能导出私钥的。正规的途径发布的证书，是不可能让你导出私钥的。
- **-b –e** 证书的有效期
- **-ss** 证书的存储名称，就是windows证书存储区的目录名，如果不存在在的话就创建一个。
- **-sr** 证书的存储位置，只有currentuser（默认值）或 localmachine两个值。
- **-sv** 指定保存私钥的文件，文件里面除了包含私钥外，其实也包含了证书。这个文件是需要保密的，这个文件在服务端配置时是需要用到的。
- 这个CN=10.30.146.206要与自己的服务器相对应，要不然在配置HTTPS的时候会出现错误
- **-a** 指定签名算法，必须是md5或rsa1。(还记得签名算法的作用不？可以看一下3章的第1节中关于签名算法的介绍)
- **-in** 指定证书发布机构的名称
- **-len** 这个参数在中文的帮助文档中好像没有提到，但是这个其实很重要，用于指定公钥的位数，越大越安全，默认值是1024，推荐2048。我试了下，这个不为1024的倍数也是可以的。

来源： http://www.cnblogs.com/franson-2016/p/5530671.html