被修改过的软件。马上我们会问，用户如何知道软件的确是来自它自己所声称的厂商，并且用户又如何知道软件从它被生产之后没有被修改过呢。当我们从一个名声未知的在线商店中下载软件或者从站点下载ActiveX控件的时候，这个问题就显得格外重要。例如，如果ActiveX控件来自一个著名的软件公司，那么它几乎不可能包含一个木马程序，但是，用户如何确信这一点呢？

一种被广泛应用的解决办法是数字签名，这部分内容在9.2.4节中已经讲解过。如果用户只运行那些由可信的地方制造并签名的程序、插件、驱动、ActiveX控件以及其他软件，那么陷入麻烦的机会就会少得多。但是这样做导致的后果就是，那些来自于Snarky Software的新的、免费的、好玩的、花哨的游戏可能非常不错但是不会通过数字签名的检查，因为你不知道谁制造了他们。

代码签名法是基于公钥密码体系。如某个软件厂商产生了一对密钥（公钥和私钥），将公钥公开，私钥妥善保存。为了完成对一个软件签名，供应商首先将代码进行散列函数运算，得到128位（采用MD5算法）、160位（采用SHA-1算法）或256位（采用SHA-256算法）的值。然后通过私钥加密取得散列值的数字签名（实际上，在使用时如图9-3所示进行了解密）。这个数字签名则始终伴随着这个软件。

当用户得到这个软件后，计算出散列函数并保存结果，然后将附带的数字签名用公钥进行解密。接着，核对解密后的散列函数值同自己运算出的值是否相等。如果相等，这个软件就被接受，否则就作为伪造版本被拒绝。这里所用到的数学方法使得任何想要篡改软件的人十分难以得手，因为这个散列函数要同从真正的数字签名中解密出来的散列函数匹配。在没有私钥的情况下通过产生匹配的假数字签名是十分困难的。签名和校验的过程如图9-34所示。

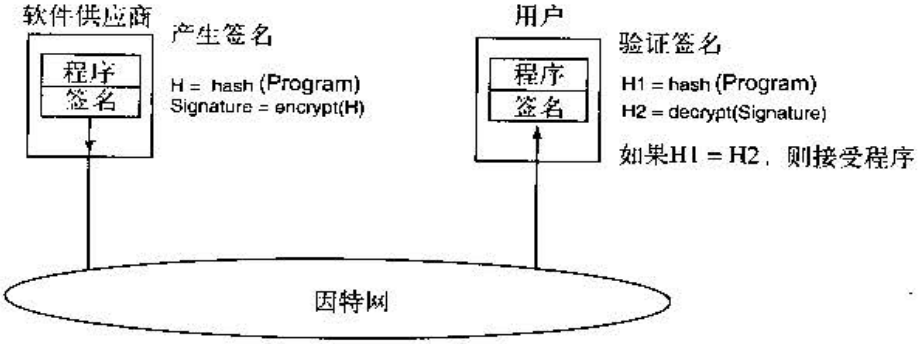

图9-34 代码签名的工作原理

网页能够包含代码，比如AcitiveX控件，以及各种脚本语言写出的代码。通常这些代码会被签名，而浏览器会自动地检查这些签名。当然，为了验证签名，浏览器需要软件厂商的公钥，它们通常和代码在一起。和公钥一起的还有被某个CA签名过的证书。如果浏览器已经保存了这个CA的公钥的话，它可以自己验证这个证书。如果这个证书是被浏览器所不知道的某个CA签名的话，那么它会弹出一个对话框询问是否接受这个证书。

9.8.4 囚禁

一个古老的俄国谚语说："相信但需要验证。"很明显地，古代的俄国人在头脑中就已经清楚地有了软件的概念。即使一个软件已经被签名了，一个好的态度是去核实它是否都能正常运行。做这件事情的一种技术是囚禁（jailing），如图9-35所示。

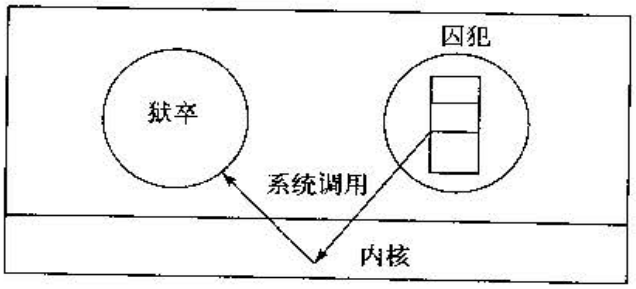

图9-35 囚禁的操作过程

如图9-35，一个新被接受的程序会作为一个标有"囚犯"的标签的进程来运行。这个"狱卒"是一个可信任的（系统的）进程，可以监管囚犯进程的行为。当一个被监禁的进程作出一个系统调用的时候，系统调用不会被执行，而是把控制移交给狱卒进程（通过一个内核陷阱）并把系统调用号和参数传递给它。这个狱卒进程会判断是否这个系统调用被允许。例如，如果被监禁的进程试图和一个狱卒进程不知道的远程主机建立一个网络连接，这个系统调用会被拒绝然后该囚犯进程被结束。如果这个