**第** **7** **章**

**数字签名技术** 



 相**概还**

互联网作为一种当今社会普遍使用的信息交换平台，越来越成为人们生活与工作中不可或 缺的一部分，但是各种各样具有创新性、高复杂度的网络攻击方法也层出不穷。2016年，国 家互联网应急中心(CNCERT/CC) 监测发现，我国境内约有1.7万个网站被篡改，针对境内 网站的仿冒页约17.8万个。网络信息的分散性广、来源隐蔽、边界模糊等特点都使得信息的 安全性与防御性十分脆弱，不法分子利用网络系统的漏洞肆意进行攻击、传播病毒、伪造信息 和盗取机密等，给政府、企业造成了巨大的经济损失。随着网络信息与共享资源的不断扩大， 网络环境的日益复杂，计算机网络信息安全的重要性也日益体现出来。因此，如何在计算机网 络中实现信息的安全传输已成为近些年人们研究的热点课题之一。

为了保障网络中数据传输的保密性、完整性，传输服务的可用性，以及传输实体的真实性、 可追溯性、不可否认性，通常情况下所采用的方法除了被动防卫型的安全技术(如防火墙技术) 外，很大程度上依赖于基于密码学的网络信息加密技术。公钥体制是目前研究与应用非常深入 的密码体制之一，其中RSA 公钥算法是其中很典型、很具影响力的代表。从最早的 E-Mail、 电子信用卡系统、网络安全协议的认证，到现在的车辆管理和云计算等各种安全或认证领域， 这种密码算法可以抵御大多数的恶意攻击方式，因此被推荐为公钥加密的行业标准。与此同时， 各种新的、针对性的攻击方式也在不断出现，512位的RSA 公钥算法早在1999年就已被超级 计算机因式分解破译，768位的RSA 公钥算法也在10年后的2009年12月被攻破。2010年， 美国密歇根大学的三位科学家已经研究出了在100小时内破译1024位RSA公钥算法的方法。 2013年8月，Google 公司宣布使用2048位的RSA 密钥加密和验证Gmail等服务。为了保证 RSA 公钥算法的安全性，有效的方法是不断地提高算法中密钥的位数。这样的做法虽然确保 了算法的安全性，但是也增加了密钥选取的困难性与算法中的基本运算——大整数模幂乘运算 的复杂性，算法的效率随之急剧地降低，也成为影响其发展的主要瓶颈之一。

数字签名是公钥密码学发展过程中衍生出的一种安全认证技术，主要用于提供实体认证、 认证密钥传输和认证密钥协商等服务。简单来说，发送消息的一方可以通过添加一个起签名作 用的编码来代表消息文件的特征，如果文件发生了改变，那么对应的数字签名也要发生改变， 即不同的文件经过编码所得到的数字签名是不同的。使用基于RSA 公钥算法的数字签名技术，



**Windows C/C++加密解密实战**

如果密码攻击者试图对原始消息进行篡改和冒充等非法操作，由于无法获取消息发送方的私钥， 因此也就无法得到正确的数字签名，若消息的接收方试图否认和伪造数字签名，则公证方可以 用正确的数字签名对消息进行验证，判断消息是否来源于发送方，这样的方式很好地保证了消 息文件的完整性，鉴定发送者身份的真实性与不可否认性。针对RSA 公钥算法攻击方式的深 入研究，制约了其在数字签名中的应用。并且，当下许多领域对数字签名技术提出了各种新的 应用需求，在未来的网络信息安全和身份认证中，基于RSA 公钥算法的数字签名技术在理论 和实际应用方面仍具有重要的研究价值与研究意义。

20世纪70年代，公钥密码体制被提出之后，应无纸化办公、数字化和信息化发展的要求， 数字签名技术也随之应运而生。不同于一般的手写签名，数字签名是一种电子形式的签名方式， 但是作用与一般的手写签名或者印章类似。数字签名技术在实现消息的保密传输、数据完整性、 身份认证和不可否认性方面的功能，对于物联网、电子商务、医疗系统等领域的发展都起着非 常重要的作用。

数字签名的方案主要依赖于密码技术，比较主流的签名方案主要有基于公开密钥的方案、 基于EIGamal的方案、基于椭圆曲线ECC 算法的方案等。RSA 公钥算法由于可以同时用于信 息加密和数字签名，并且具有较好的安全性，是目前应用普遍的签名方案之一。随着数字签名 技术研究的愈加深入，签名方案日益增多，有盲签名、群签名、代理签名等。同时，关于数字 签名的安全性分析和攻击研究也在不断深入。

自从数字签名技术被提出以来，不少密码学者和相关的机构(如麻省理工学院、IBM 研 究中心等)都开始致力于数字签名的研究，并高度关注该项技术的发展。1984年9月，在国 际标准化组织的建议下，SC20 率先开始为数字签名技术制订标准，将其分为带影子、带印章 以及使用Hash 函数的三种数字签名。WG2 在1988年5月起草了使用Hash 函数的签名方案 DP9796,并于1989年10月将其升级为DIS9796。与此同时，日本、英国等国的相关组织也 迅速展开了对数字签名标准的制订工作。1991年8月，美国NIST 公布了DSA 签名算法，并 将其纳入 DSS 数字签名标准之中。2000年6月，美国通过了有关数字签名的法案。目前，影 响较大的制订数字签名相关标准的组织包括：国际电子技术委员会 (International

Electrotechnical Commission,IEC)、国 际 标 准 化 组 织 (International Organization for

Standardization,ISO) 、 美国国家标准与技术委员会 (National Institute of Standards and Technology,NIST) 等。在国际上，数字签名技术的相关标准和法案已日趋完善，我国因为在 这一领域起步较晚的关系，在数字签名的操作中主要是吸收国外的经验，并且还存在很多不规 范之处。鉴于此，2005年4月正式颁布并施行了《中华人民共和国电子签名法》,这部法律 不仅确立了数字签名的法律效力和地位，还规范了数字签名的行为，维护了有关方的合法权益， 促进了我国电子商务的发展，我国数字签名研究工作的顺利进行也因此得到了有力的支持和保 障。



第7章数字签名技术



**什么是数字签名技术**

**7.2.1** **签** **名**

在文件上手写签名和盖章长期以来被用作证明作者的身份，或至少同意文件的内容。一个 签名至少具有5个特性：(1)签名是可信的；(2)签名不可伪造；(3)签名不可重用；(4) 签名的文件是不可改变的；(5)签名是不可抵赖的。我们使用数字签名技术来保证对电子文 档的签名，也同样具有这5个特性，从而使数字签名跟手写签名和盖章一样，甚至具有更高的 安全性。

**7.2.2** **数** **字** **签** **名** **的** **基** **本** **概** **念**

数字签名是对手写签名的模拟，通过分析手写签名的特点，一个数字签名方案至少要满足 以下三个条件：

(1)不可抵赖性。签名者事后不能否认他签署的签名。

(2)不可伪造性。其他任何人不能伪造签名者的签名。

(3)可仲裁性。当签名双方对一个签名的真实性发生争执时，第三方仲裁机构可以帮助 解决争执。

数字签名算法与公钥加密算法类似，是私钥或公钥控制下的数学变换，而且通常可以从公 钥加密算法派生而来。签名是利用签名者的私钥对消息进行计算、变换，而验证则是利用公钥 检验该签名是不是由签名者签署。只有掌握了签名者的私钥，才能得到签名者的签名，从而实 现不可否认性、不可伪造性和可仲裁性。

数字签名是公钥密码体制衍生出的重要的技术之一。相较于在纸上书写的物理签名，数字 签名首先对信息进行处理，再将其绑附一个私钥上进行传输，形成一个比特串的表示形式，可 以实现用户对电子信息来源的认证，并对信息进行签名，确认并保证信息的完整性与有效性。 在数字签名出现之前，除了传统的物理签名外，还有一种“数字化”签名技术，即在电子板上 进行签名，接着将签名传输到电子文件中，虽然相比物理签名，这种方式更加便捷，然而仍然 可以被非法地复制并粘贴到其他文件上，因此签名的安全性无法得到保障。不同于“数字化” 签名技术，数字签名与手写签名的形式毫无关系，它实际上是使用密码学的技术将发送方的信 息明文转换成不可识别的密文进行传输，在信息的不可伪造性、不可抵赖性等方面有着其他签 名方式无法替代的作用。

数字签名技术应用十分广泛，数字签名在包括身份认证识别、数据完整性保护、信息不可 否认及匿名性等许多信息安全领域中都有重要的用途。甚至可以说有信息安全的地方，就有数 字签名。特别是在网络安全通信、电子商务等系统中，数字签名具有重要作用。



Windows C/C++加密解密实战

**7.2.3** **数字签名的原理**

原则上，所有的公开密钥算法都能用于数字签名。事实上，工业标准就是RSA算法，许 多保密产品都使用它。数字签名发展至今，常见的签名算法除了EIGamal 签名算法、RSA签 名算法外，还有Schnorr算法、DSA 算法等。公钥密码体制并非都可以同时用作加解密系统和 进行数字签名，一般只能实现其中的一种功能，而我们学习的RSA 公钥密码体制可以同时实 现二者。下面给出数字签名的原理。

(1)系统的初始化，生成数字签名中所需的参数。

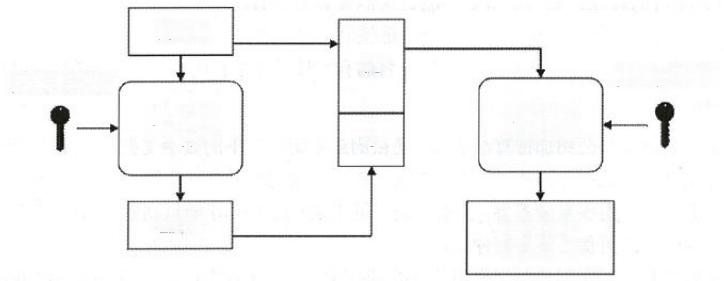
(2)发送方利用自己的私钥对消息进行签名。

(3)发送方将消息原文和作为原文附件的数字签名同时传给消息接收方。

(4)接收方利用发送方的公钥对签名进行解密。

(5)接收方将解密后获得的消息与消息原文进行对比，如果二者一致，那么表示消息在 传输中没有受到过破坏或者篡改，反之不然。

消息签名和验证的基本原理如图7-1所示。



消息M

发送方的公钥

数字签名 验证算法

返回签名 有效或无效

数字签名 生成算法

**发送方的私钥**

签名S

M

S

图7-1

**7.2.4** **数字签名的一般性定义**

一般来说， 一个数字签名方案由三个集合和三个算法组成，这三个集合分别是消息空间， 签名空间和密钥空间(包括私钥和公钥)。三个算法如下：

(1)密钥生成算法：这是一个概率多项式时间算法，输入为安全参数，输出为私钥 SK 和公钥PK。

(2)签名生成算法：这是一个(概率)多项式时间算法，输入为私钥SK 和待签消息m, 输出对应于私钥SK 的关于消息m 的签名 sign(m)。

(3)签名验证算法：这是一个确定性多项式时间算法，输入为签名 sign(m) 、消 息m 和 对应于私钥SK 的公钥PK, 输出为“正确”或“错误”。正确就是验证通过，错误就是验证



**第7章数字签名技术**

没通过。被验证为正确的签名称为有效签名。一个数字签名至少应满足正确性、不可伪造性和 可仲裁性三个性质。

**7.2.5** **数字签名的分类**

(1)基于数学难题的分类

根据数字签名方案所基于的数学难题，数字签名方案可分为基于离散对数问题的签名方案、 基于素因子分解问题(包括二次剩余问题)的签名方案、基于椭圆曲线的数字签名方案、基于 有限自动机理论的数字签名方案等。比如，ELGamal 数字签名方案和DSA 数字签名方案都是 基于离散对数问题的数字签名方案。而众所周知的RSA 数字签名方案是基于素因子分解问题 的数字签名方案。将离散对数问题和因子分解问题结合起来，又可以产生同时基于离散对数和 素因子分解问题的数字签名方案。

(2)基于密码体制的分类

根据密码体制可以将数字签名分为对称密钥密码体制的数字签名和非对称密钥密码体制 的数字签名两种。

对称密钥密码体制的数字签名：这种签名体制引入公证机关这个第三方，其文件的安全性 和可靠性都取决于公证机关。具体来讲是先由发送方把明文用自己的密钥加密后传送给公证机 关，公证机关用发送方的密钥对报文解密，再构造一个新报文，包括发送方的名字、地址、时 间及原报文，用只有公证机关知道的密钥加密后送回发送方，发送方把此加密后的新报文发送 给接收方，接收方因为不知道密码而无法恢复原报文，只能将此报文发至公证机关，公证机关 解密后再用接收方的密钥进行加密，发回接收方，接收方即可解密还原报文。从以上过程不难 看出，这种体制的数字签名由于两次通过公证机关，增加了报文的传输时间，降低了报文的传 输效率。

非对称密钥密码体制的数字签名方案：这种体制的独特优点是具有公开密钥和秘密密钥两 个密钥，以至于特别适合实现数字签名。其过程为：发送方先用其秘密密钥对报文进行解密运 算，然后用接收方的公开密钥进行加密运算后，传送至接收方，接收方收到该报文后先用自己 的秘密密钥进行解密运算，再用发送方的公开密钥进行加密运算，即可恢复出明文。在此过程 中，因为除了发送方自己外，没有别人具有其解密密钥，所以除了发送方自己外，没有别人能 产生密文，报文就被签名了。如果发送方抵赖曾发送报文给接收方，接收方可将发送方产生的 密文出示给公证机关，公证机关用发送方的公开密钥去证实其发文的确实性，反之，如接收方 对报文进行伪造篡改，则其不能用发送方的公开密钥进行加密后发送给第三方，证明其伪造了 报文。以上过程表明，用非对称密钥密码体制实现数字签名既简便又安全。

(3)基于特殊用途的分类

当一般数字签名方案不能满足某些特别的签名需要时，便需要借助特殊数字签名方案。下 面对几种常用的特殊数字签名进行简要的介绍。

盲签名(Blind Signature):有时候消息的拥有者想让签名者对消息进行签名，而又不希 望签名者知道消息的具体内容，同时签名者也不想了解所签消息的具体内容，他只是想让人们



**Windows C/C++加密解密实战**

知道他曾经签署过这个消息，这时就需要使用盲签名。由于盲签名具有匿名的性质，因此在电 子货币和电子选举系统中得到了广泛的应用。

双重签名(Dual Signature): 例如在安全电子交易(SET) 协议中，交易时，客户需要发 送订购信息(Order Information,OI) 给商家，发送支付信息(Payment Information,PI) 给银 行。而客户发往银行的支付指令是通过商家转发的，为了避免在交易的过程中商家窃取客户的 信用卡信息，以及避免银行跟踪客户的行为，侵犯消费者的隐私，商家不需要知道客户的信用 卡信息，银行也不需要知道客户订单的细节。但同时又不能影响商家和银行对客户所发信息的 合理验证，这时SET 协议可以采用双重签名来解决这一问题。在双重签名中，签名者希望验 证者只知道报价单，中间人只知道授权指令，能够让中间人在签名者和验证者报价相同的情况 下进行授权操作。

群签名 (Group Signature):允许一个群体中的成员以整个群体的名义进行数字签名，并 且验证者能够确认签名者的身份。群签名中重要的是群密钥的分配，以能够高效处理群成员的 动态加入和退出。一般的群密钥管理可以分为两大类别：集中式密钥管理和分布式密钥管理。

代理签名 (Proxy Signature): 现实生活中人们常常会将自己的签名权力委托给可信的代 理人，让代理人代表他们在文件上盖章或签名。在数字化的信息社会中，人们使用数字签名的 过程中仍然会遇到需要将签名权力委托或转移给他人的情况。这时候就需要用到代理签名，代 理签名允许密钥持有者授权第三方，获得授权的第三方能够代表密钥持有者进行数字签名。

(4)其他分类

根据接收者验证签名的方式可将数字签名分为真数字签名和仲裁数字签名两类。从签名者 在一个数字签名方案中所能签的消息的个数来分，可将数字签名分为一次数字签名和非一次数 字签名。根据数字签名方案中的验证方程是否为隐式或显式，可将数字签名分为隐式数字签名 和显式数字签名。根据数字签名的功能，可将数字签名分为普通的数字签名和具有特殊性质的 数字签名。

**7.2.6** **数字签名的安全性**

针对数字签名的安全性的研究，即鉴定签名方案是否满足信息完整性、抗修改性和抗抵赖 性等安全性质，一直是促进该项技术发展的重要方面之一。安全性的主要研究方法是从理论上 进行分析，分析的技术分为三种：

(1)安全性评估。在数字签名方案设计的过程中，设计者对所设计的方案进行相关的密 码分析，尽可能保证其在所能考虑到的范围内的安全性。由于一般情况下，方案设计者无法穷 举所有可能出现的情况，因此这种分析不是证明，只是一种对签名方案安全性的评估，可以在 一定程度上使用户对方案拥有信心。

(2)安全性证明。这种方式主要应用在数字签名方案设计的过程中，证明数字签名方案 被密码攻击者破译的难度与破解一个公认的难题相当。目前主要的技术方案有两类 基于随 机应答模型与基于标准模型。前者就是假设 Hash 函数是一个对所有请求都会做出随机应答的 随机黑盒应答器，相同的请求得到的应答完全相同。然而实际上，Hash 函数并不满足这个假



**第7章数字签名技术**

设，所以基于随机应答模型的证明不完全可靠。后者是指在不需要不合理假设的情况下，证明 方案是安全的，所谓的标准模型本质上并不是一个具体的模型，仅是针对随机应答模型而言的。

(3)攻击。此类技术是指对方案的安全缺陷进行研究，主要针对的是目前已知的数字签 名方案。分析人员从攻击者的角度进行演绎与推理，尝试通过不同的手段获取在方案安全性规 定下不应该或者不能够获取的信息。如果成功地获取到了这样的信息，就找出了已知方案所存 在的安全性缺陷，然后再次分析整个推理过程。针对攻击的分析中，恶意分析者被称为攻击者， 方案分析人员演绎与推理的过程被称为一个攻击方法。攻击分析促使人们对存在安全性缺陷的 方案进行改进或者直接淘汰，从客观上提升了数字签名的安全性。

**7.2.7** **数字签名的特征与应用**

一个优秀的数字签名应当具备下列特征：

(1)消息发送方一旦给接收方发出签名之后，不可以再对他所签发的消息进行否认。

(2)消息接收方可以确认并证实发送方的签名，但是不可以伪造签名。

(3)如果签名是复制其他的签名获得的，那么消息接收方可以拒绝签名的消息，即不可 以通过复制的方式将一个消息的签名变成其他消息的签名。

(4)如果消息接收方已收到签名的消息，就不能再否认。

(5)可以存在第三方确认通信双方之间的消息传输的过程，但是不可以伪造这个过程。

在互联网和通信技术迅猛发展的潮流下，电子商务一跃成为商务活动的新模式，越来越多 的信息都以数字化的形式在互联网上流动。在传统的商务系统中，一般都是利用纸质文件的签 名或印章等物来规定某些具有契约性质的责任，而在电子商务系统中，当事人身份与数据信息 的真实性是利用传送的文件的电子签名进行证实的。电子商务包括电子数据交换、管理信息系 统、商业增值网、电子订货系统等，其中电子数据交换既是电子商务这些部分的核心，又是一 项涉及多个环节的复杂的人机工程，同时对信息安全性的要求也相当高。互联网环境的开放性、 共享性等特点使得网络信息安全变得异常脆弱，如何保证数据信息在互联网上传输的安全性以 及交易双方的身份确认，不仅是电子商务的必然要求，也是电子商务能否得到长足发展的关键。 作为电子商务安全性的重要保障之一，RSA 公钥签名方案相对成功地解决了上述问题，并在 各种互联网行为中都有着广泛的应用。



**RSA** **公钥算法在数字签名中的应用**

基于公钥体制的 RSA 算法为信息安全传输的问题提供了新的解决思路和技术，也被用作 数字签名方案，在实现消息认证方面得到了深入的应用。总的来说，RSA 公钥签名方案包括 消息空间、参数生成算法、签名算法和验证算法等部分。签名和验证的过程具体包括消息摘要 的生成、大素数和密钥的生成以及消息签名、消息验证等几个步骤。

传统的签名方案使用 RSA 公钥算法进行数字签名，签名的过程如下：



**Windows C/C++加密解密实战**

(1)参数的选择和密钥的生成。

(2)签名过程：用户A 对消息M 进行签名，则计算： S=Sig(M)=M⁴mod n

d 是私钥，相当于用私钥进行加密运算(但最好不要习惯说私钥加密，说私钥签名比较好， 否则容易和公钥加密混淆，加密就是加密，签名就是签名，两者的目的不同，私钥是为了签名， 不是加密。有些算法库提供了私钥加密函数，大家应该清楚实际上是私钥签名),然后将签名 结 果S 作为用户A 对消息M 的数字签名附在消息M 后面，发送给用户B。

(3)验证过程：用户B 验证用户A 对 M 的数字签名S, 则计算：M'=S mod n,然后判 断 M 和 M 是否相等，若二者相等，则可以证明签名S 的确来源于用户A, 否则签名S 有可能 为伪造的签名。例如，假设用户 A 选取 p=823 、q=953,那么模数 n=784319、 φ(n)=(p-1)(q-1)=782544, 然后选取e=313 并计算出d=160009, 则公钥为(313,784319),私钥 为(160009,784319)。如果现在用户A 拟发送消息M=19070给用户B, 用私钥对消息进行签名： M:19070->S=(19070)160009 mod 784319=210625 mod 78319,用 户A 将消息和签名同时发送 给用户B, 用户B 接收消息和签名，并计算出M':

M'=2106253¹³3mod 784319=19070 mod 794319->M=M'mod n

因此，用户B 验证了用户A的签名，并接收了消息。

在实际的应用过程中，待签名的消息一般都比较长，因此需要对消息明文先进行分组，然 后对不同的分组明文分别进行签名。这样会致使算法对于长文件签名的效率十分低下。为了解 决这个问题，可以利用单项摘要函数(Hash 函数),即在对消息进行签名之前，事先使用Hash 函数对需要签名的消息做Hash变换，对变换后的摘要消息再进行数字签名。

增加了Hash 运算，则发送者要发送三样东西：原文、原文的摘要值和摘要的数字签名值。 这样，接收者收到三样东西：原文、原文的摘要以及摘要的数字签名。接收者先对原文做摘要， 然后和收到的摘要值做比较，如果一致就说明原文没有被篡改过，接着用发送者的公钥对数字 签名做验签，如果通过就说明一定是发送者本人发来的(因为只有拥有私钥的发送者才能签出 这样一份数字签名)。

**7.4 使用OpenSSL命令进行签名和验签**

本节以RSA 私钥签名为例进行介绍。首先生成RSA密钥对。打开操作系统的命令行窗口， 输入cd 进入D:lopenss-1.1.1b\win32-debug\bin\,然后输入openssl.exe,并按回车键开始运行。

我们输入生成私钥的命令：genrsa -out prikey.pem,如图7-2所示。



**第7章数字签名技术**



**回管理员：C:\Tiadows\systo32\cd exo-penss1e** 区

D:openssl-1.1.1bwin32-debug bin>openssl.exe OpenSSL>genrsa -out prikey.pen

Generating RSA priuate key.2048 bit long nodulus<2 prines)

..+++++

e is 65537(BxB1BB81>

0penSSI

D



图7-2

默认生成了一个长度为2048位的私钥。执行后，将在 D:\openssl-1.1.1b\win32-debug\bin\ 下生成一个名为prikey.pem的私钥文件，它是PEM格式的。

接着从私钥中导出公钥，输入命令：rsa -in prikey.pem-pubout -out pubkey.pem,如图7-3 所示。

回管理员：C:NYindos\systo32\cd

OpenSSL>rsa -in prikey.pem

oriting RSA key OpenSSL>

eoens1

-puhout -out

pubkey.pen

凶

图7-3

执行后，将在同目录下生成公钥文件 pubkey.pem。

公私钥准备好后，就可以开始签名了。我们用RSA私钥对 SHA1计算得到的摘要值进行 签名。首先，准备原文件，可以在D:\openssl-1.1.1b\win32-debug\bin\ 下新建一个文本文件 file.txt, 输入一行内容：helloworld, 然后保存退出。接着，在OpenSSL提示符下，输入命令：

dgst -sign prikey.pem -shal -out shal\_rsa\_file.sign file.txt

执行后，将在同目录下生成一个签名文件 shal\_rsa\_file.sign,其内容就是摘要的RSA数字 签名。

至此，签名工作完成。下面开始用相应的公钥和相同的摘要算法进行验签，在 OpenSSL 提示符下输入命令：

dgst -verify pubkey.pem-shal -signature shal\_rsa\_file.sign file.txt

执行后，如果验签通过，就提示Verified OK,如图7-4所示。



|  |  |  |
| --- | --- | --- |
| 国管理员：C:\indorr\systo32\ead.exe-openssl.exe | 回 | × |
| OpensSL>dgst -verify pubkey.pen -hat -ignature | cha1\_rsa\_file.cign file.txt | 日 |
| Uerified OK 0pensS |  |  |

图7-4

至此，RSA 验签成功。



Windows C/C++加密解密实战



**基于OpenSSL** **的签名验签编程**

和 RSA 加解密一样，签名验签的编程方式也有两种，一种是直接调用RSA 加解密函数进 行签名和验签，这种方式简称直接方式；另一种是使用EVP 系列函数。

**7.5.1 直接使用** **RSA 函数进行签名验签**

对于RSA 签名，OpenSSL 提供了RSA\_sign 和RSA\_verify 这两个函数来完成签名和验签。 其中，函数RSA\_sign 用于对摘要进行签名，它使用私钥对指定摘要算法的摘要结果进行签名， 该函数声明如下：



int RSA\_sign(int type,const unsigned char \*m,unsigned int m\_length,unsigned char \*sigret,unsigned int \*siglen,RSA\*rsa);

其中，参数type 表示摘要值所采用的摘要算法，常用取值如下：

● NID\_md5:表 示MD5摘要算法。

● NID\_sha:表示 SHA 摘要算法。

● NID\_shal: 表示 SHA1 摘要算法。

● NID\_md5\_shal:表示同时做MD5和SHA1摘要。

参 数m 表示要签的摘要值；m\_length 表示摘要m 的长度；sigret 表示输出的签名结果；siglen 表示签名结果的长度；rsa 指 向RSA 结构体。如果函数执行成功就返回1,否则返回0。

RSA\_sign函数使用PKCS#1 v2.0中指定的私钥RSA, 对大小为m\_len 的消息摘要m 进行 签名。它将签名结果存储在sigret 中，签名大小存储在siglen 中 。sigret 必须指向RSA\_size(RSA) 字节大小的内存缓冲区。

通常在函数RSA\_sign 调用前，要调用摘要函数对原文进行摘要计算，然后把摘要结果传 到RSA\_sign 函数中。

函数RSA\_verify 验证siglen 大小的签名sigbuf 是否与m\_len 大小的给定消息摘要m 匹配。 类型表示用于生成签名的消息摘要算法。rsa 是签名者的公钥。该函数声明如下：



int RSA\_verify(int type,const unsigned char \*m,unsigned int m\_length,const unsigned char \*sigbuf,unsigned int siglen,RSA\*rsa);

其中，参数type 表示摘要值所采用的摘要算法，常用取值如下：

● NID\_md5:表示MD5摘要算法。

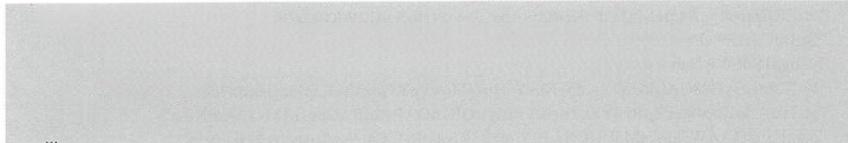
● NID\_sha:表示SHA摘要算法。

● NID\_shal: 表示SHA1摘要算法。

● NID\_md5\_shal: 表示同时做MD5和SHA1摘要，此时m\_length应该是36,md5是 16字节，shal 是20字节，一共36字节。在RSA\_sign源码(rsa\_sign.c) 中，如果 m\_length 不是36,将返回0,代码片段如下：



**第7章数字签名技术**



if(type==NID\_md5\_shal){

if(m\_len!=SSL\_SIG\_LENGTH){//SSL\_SIG\_LENGTH 是一个宏，值是36 RSAerr(RSA F RSA\_SIGN,RSA\_R\_INVALID\_MESSAGE\_LENGTH);

return (0);

}

参数m 表示摘要值；m\_length 表示摘要m 的长度；sigbuf 表示输出的签名结果；siglen 表示签名结果的长度；rsa 指向RSA 结构体，用于存放公钥。如果函数验签成功就返回1,否 则返回0。

【例7.1】直接方式签名验签

(1)打开VC 2017,新建一个控制面板工程，工程名是test。 ( 2 ) 在 工 程 中 打 开 test.cpp, 并 输 入 代 码 如 下 ：

#include "pch.h"

#include<iostream>

#include<openssl/pem.h> #include<openssl/ssl.h> #include<openssl/rsa.h>

#include<openssl/evp.h> #include<openssl/bio.h> #include<opensslVer.h> #include <stdio.h>

#include<iostream> #include<fstream>

using namespace std;

#ifdef WIN32

#pragma comment(lib,"libcrypto.lib") #pragma comment(lib,"ws2\_32.lib") #pragma comment(lib,"crypt32.lib") #endif

int padding=RSA\_PKCS1\_PADDING;

char publicKey[]=----BEGIN PUBLIC KEY----n"\

"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAy8Dbv8prpJ/0kKhlGeJY\n"\ "ozo2t60EG8L0561g13R29LvMR5hyvGZIGJpmn65+A4xHXInJYiPuKzrKUnApeLZ+\n"\

"vwlHocOAZtWK0z3r26uA8kQYOKX9Qt/DbCdvsF9wF8gRKOptx9M6R13NvBxvVQApln"\ "fc9jB9nTzphOgM4JiEYvIV8FLhg9yZovMYd6Wwf3aoXK891VQxTr/kQYoqlYp+68\n"\

"i6T4nNq7NWC+UNVjQHxNQMQMzU6IWCX8zyg3yH88OAQkUXIXKfQ+NkvYQ1cxaMoV\n"\

"PpY72+eVthKzpMeyHkBn7ciumk5qgLTEJAfWZpe4f4eFZj/Rc8Y8Jj2IS5kVPjUy\n"\ "wQIDAQAB\n"\

"-----END PUBLIC KEY-----\n";

char privateKey[]=----BEGIN RSA PRIVATE KEY-----"

"MIIEowIBAAKCAQEAy8Dbv8prpJ/0kKhIGeJYozo2t60EG8L0561g13R29LvMR5hyìn"\

**Windows** **C/C++加密解密实战**

"vGZIGJpmn65+A4xHXInJYiPuKzrKUnApeLZ+vwlHocOAZtWK0z3r26uA8kQYOKX9\n"\

"QtDbCdvsF9wF8gRKOptx9M6R13NvBxvVQApfc9jB9nTzphOgM4JiEYvIV8FLhg9\n"\

"yZovMYd6Wwf3aoXK891VQxTr/kQYoqlYp+68i6T4nNq7NWC+UNVjQHxNQMQMzU61\n"\ "WCX8zyg3yH88OAQkUXIXKfQ+NkvYQ1cxaMoVPpY72+eVthKzpMeyHkBn7ciumk5q\n"\

"gLTEJAfWZpe4f4eFZj/Rc8Y8Jj2IS5kVPjUywQIDAQABAoIBADhglu1MvlhAAIX8\n"\ "omzlGn2f4AAW2aos2cM5UDCNwlSYmj+9SRIkaxjRsE/C409swloxrg1/z6kajVOe\n"\

"N/t008FdlVKHXAIYWF93JMoVvIpMmT8jft6AN/y3NMpivgt2inmmEJZYNioFJKZG\n"

"X+/vKYvsVISZm2fw8NfnKvAQK55yu+GRWBZGOeS9K+LbYvOwcrjKhHz66m4bedKd\n"\

"gVAix6NE5iwmjNXktSQIJMCjbtdNXg/xol/G4kG2p/MO1HLcKfelN5FgBiXj3Qjln"\ "vgvjJZkhlas2KTgaPOBqZaP03738VnYg23ISyvfT/teArVGtxrmFP7939EvJFKpF\n"

"IwTxuDkCgYEA7tODR37zt+dEJy+5vm7zSmN97VenwQJFWMiulkHGaOyU3ILasxxuln"\ "mOoUtndljenlvSx6t3Y+agK2F3EPb0AZ5wZ1plIXs4vktgeQwSSBdqcM8LZFDvZ\n"\

"uPboQnJoRdIkd62XnP5ekIEIBAfOp8v2wFpSfE7nNH2u4CpAXNSF9HsCgYEA218D\n"\

"JrDE5m9Kkn+J4l+AdGfeBLligPF3DnuPoV67BpgiaAgI4h25UJzXiDKKoa706SOD\n"\

"4XB74zOLXI1MaGPMIdhlG+SgeQfNoC5IE4ZWXNyESJHISVgRGT9nBC2vtL6bxCVV\n"I "WBkTeC5D6c/QXcai6yw6OYyNNdp0uznKURelxvMCgYBVYYcEjWqMuAvyferFGV+5\n"\ "nWqr5gM+yJMFM2bEqupD/HHSLoeiMm2O8KIKvwSeRYzNohKTdZ7FwgZYxr8fGMoG\n" "PxQ1VK9DxCvZL4tRpVaU5Rmknud9hg9DQG6xIbgIDR+f79sb8QjYWmcFGc1SyWOA\n"\ "SkjlykZ2yt4xnqi3BfD9QKBgGqLgRYXmXplQoVIBRaWUi55nzHg1XbkWZqPXvzI\n"\

"I3uMLvljLjJIHk3euKqTPmC05HoApKwSHeA0/gOBmg404xyAYJTDcCidTg6hlF96\n"\

"ZBja3xApZuxqM62F6dV4FQqzFXOWWhWp5n301N33r0qR6FumMKJzmVJITA8tmzEF\n"\ "yINRAoGBAJqioYs8rK6eXzA8ywYLjqTLu/yQSLBn/4ta36K8DyCoLNINxSuox+A5\n"\

"w6z2vEfRVQDq4Hm4vBzjdi3QfYLNkTiTqLcvgWZ+eX44ogXtdTDO7c+GeMKWz4XX\n"\

"uJSUVL5+CVjKLjZEJ6Qc2WZL194xSwL71E41H4YciVnSCQxVc4Jwìn"\ "-----END RSA PRIVATE KEY-----n";

//把字符串写成 public.pem 文件

int createPublicFile(char\*file,const string &pubstr)

{

if(pubstr.emptyO) {

printf("public key read error\n"); return(-1);

}

int len=pubstr.length(); string tmp=pubstr

for (inti=64;i<len;i+=64) {

if(tmp[i]!='n') {

tmp.insert(i,"\n");

一

f

*i++;*

}

tmp.insert(0,"----BEGIN PUBLIC KEY-----n"); tmp.append("\n-----END PUBLIC KEY-----\n");

//写文件

ofstream fout(file) fout<<tmp.c\_str(



**第7章数字签名技术**

return (0); }

//把字符串写成 private.pem 文件

int createPrivateFile(char\*file,const string &pristr) {

if(pristr.emptyO)

{

printf("public key read error\n") return(-1);



j

int len=pristr.length(); string tmp=pristr

for (inti=64;i<len;i+=64) {

if(tmp[i]!='n) {

tmp.insert(i,"\n"); }

i++; }

tmp.insert(0,----BEGIN RSA PRIVATE KEY----n"); tmp.append(----END RSA PRIVATE KEY----n")

//写文件

ofstream fout(file); fout<<tmp.c\_str()

retum(0); }

//读取密钥

RSA\*createRSA(unsigned char\*key,int publi) {

RSA \*rsa=NULL; BIO\*keybio

keybio=BIO\_new\_mem\_buf(key,-1); if(keybio=NULL)

{

printf("Failed to create key BIO\n"); retum 0;

}

if (publi) {

rsa=PEM\_read\_bio\_RSA\_PUBKEY(keybio,&rsa,NULL,NULL);

}

else (

rsa=PEM\_read\_bio\_RSAPrivateKey(keybio,&rsa,NULL,NULL);

**Windows** **C/C++加密解密实战**

}

if(rsa ==NULL)

—

printf("Failed to create RSA\n"); }

return rsa; }

//公钥加密

int public\_encrypt(unsigned char\*data,int data\_len,unsigned char\*key,unsignedchar\*encrypted) {

RSA\*rsa=createRSA(key,1);

int result=RSA\_public\_encrypt(data\_len,data,encrypted,rsa,padding) return result;

}

//私钥解密

int private\_decrypt(unsigned char\*enc\_data,int data\_len,unsigned char\*key,unsigned char\*decrypted) {

RSA\*rsa=createRSA(key,0);

int result=RSA\_private\_decrypt(data\_len,enc\_data,decrypted,rsa,padding); return result;

}

int public\_decrypt(unsigned char\*enc\_data,int data\_len,unsigned char\*key,unsigned char\*decrypted) {

RSA\*rsa=createRSA(key,1);

int result=RSA\_public\_decrypt(data\_len,enc\_data,decrypted,rsa,padding); returm result;

}

//私钥签名

int private sign(const unsigned char \*in str,unsigned int in str len,unsigned char \*outret,unsigned int\*outlen, unsigned char\*key)

{

RSA\*rsa=createRSA(key,0);

unsigned char md[20]="";

SHA1(in\_str,in\_str\_len,md);

int result=RSA\_sign(NID\_shal,md,20,outret,outlen,rsa) if (result !=1)

{

printf("sign error\n"); return -1;

}

return result; }

//公钥验签

int public\_verify(const unsigned char \*in\_str,unsigned int in\_len,unsigned char \*outret,unsigned int outlen, unsigned char\*key)

**第7章数字签名技术**

{

RSA\*rsa=createRSA(key,1); unsigned char md[20]=""; SHAl(in\_str,in\_len,md);

int result=RSA\_verify(NID\_shal,md,20,outret,outlen,rsa); if(result !=1)

{

printf("verify error\n"); return -1;

}

return result; }

int main()//主函数 {

char plainText[2048/8]="hello";/key length:2048

printf("create pem file\n"); //自己预定义公钥

string strPublicKey =

"MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQChNrOTmflORv9C62+SAYhyj4DwB6fyOHqttddq8Y+ R+8cIGT7EKuqSRuUUuLVBN6IIjd14UkxxtjHqrDxPWZz9WfXOLB2ITmnSdkg9Q10IfP9ZrVCW8Pe5vJ7gt5iQ41 OebdqR47+ef9E7oE+eJFQhxSYGGy/FnKjBkadJQtwPQIDAQAB";

int file\_ret=createPublicFile(char\*)"public\_test.pem",strPublicKey);//自己建立公钥文件

unsigned char encrypted[4098]={}; unsigned char decrypted[4098]={); unsigned char signret[4098]={};

unsigned int siglen;

printf("source data=[%s]\n",plainText)

printf("public encrytpt ---private decrypt nhn");

intencrypted\_length=public\_encrypt(unsigned char\*)plainText,strlen(plainText),(unsigned char\*)publicKey,encrypted);

if(encrypted\_length =-1)

{

printf("encrypted error \n"); exit(0)

}

printf("Encrypted length =%d\n",encrypted\_length);

int decrypted\_length=private\_decrypt((unsigned char\*)encrypted,encrypted\_length,(unsigned char\*)privateKey,decrypted);

if(decrypted\_length ==-1)

{

printf("decrypted error \n"); exit(0);

}

printf("DecryptedText =%s\n",decrypted);

printf("DecryptedLength=%d\n",decrypted\_length);



**Windows** **C/C++加密解密实战**

printf("\nprivate sign ----public verify:%d\n\n");

nt ret=private\_sign((const unsigned char\*)plainText,strlen(plainText),signret,&siglen,(unsigned char\*)privateKey);

printf("sign ret =[%d]n",ret)

ret=public\_verify((const unsigned char\*)plainText,strlen(plainText),signret,siglen,(unsigned char\*)publicKey);

if(ret=1)

printf("verify OK,ret =[%d]n",ret); else

printf("verify failed,ret =[%d]\n",ret); return(0);

在代码中，我们对明文hello 进行了加解密及签名和验签，签名和验签所使用的摘要算法 是 SHA1, 它的结果是20字节。

本例所使用的公私钥是预定义的，首先预定义了一段公钥存于strPublicKey中，并建立了 一个公钥文件public\_test.pem, 这给大家演示了如何手工构造一个PEM 文件，其实并不神秘。 然后我们显式地进行了加解密，解密时的私钥privateKey是一个全局变量，也是预置的私钥， 效果其实和从PEM 文件中读取是一样的，最终都是构造出RSA 结构体来，构造RSA 结构体 的操作是在自定义函数createRSA 中进行的，该函数的第二个参数publi 用于标记是构造公钥 还是私钥，如果是构造公钥就调用 PEM\_read\_bio\_RSA\_PUBKEY 函数，否则调用 PEM\_read\_bio\_RSAPrivateKey 函数，这两个函数都是从内存中读取的公私钥内容(比如是已 经准备的PEM 格式，这很明显，因为这两个函数名的开头都是PEM\_)。

最后，在工程属性中添加附加包含目录：D:\openssl-1.1.1b\win32-debug\include, 再添加附 加库目录：D:\openssl-1.1.1b\win32-debuglib。

(3)保存工程并运行，运行结果如图7-5所示。

人日



Lierosoft Visual Stadio谓试控制台口回区

create pen fil

source data-[hello]

public encrytpt - private Encrypted length =256

DecryptedText =hello

**DecryptedLength -5**

at g[1 ----public verify

verify OK,ret =[1J

口

decrypt



图7-5

**7.5.2 使** **用EVP 系列函数进行签名验签**

使用EVP 系列函数进行签名，流程和摘要编程一样，也是三部曲，即签名初始化、签名 更新(Update, 一次或多次)、签名结束 (Final) 出结果。验签也是如此，即验签初始化、验



**第7章数字签名技术**

签(一次或多次)、验签结束。

OpenSSL 在 Evp.h 中对签名和验证函数进行了封装。对数据签名其实就是对数据的摘要进 行私钥加密。验证签名就是解密签名数据，和原始的摘要对比是否一样。在OpenSSL 中签名 是先对原始数据计算摘要，再对摘要进行私钥加密。在OpenSSL 验证签名的过程是对原始消 息计算摘要，解密签名值，和摘要对比是否一致。如果一致，就说明签名有效；否则认为原名 或签名值被篡改。数字签名结合数字证书可以实现身份认证、防篡改、防抵赖的功能。签名的 数据格式为PKCS#1。

OpenSSL 处理签名的函数主要有 EVP\_SignInit\_ex 、EVP\_SignUpdate 和 EVP\_SignFinal 。 验证签名的函数主要有 EVP\_VerifyInit\_ex 、EVP\_Verify\_Update 和 EVP\_VerifyFinal。其 中 ， EVP\_SignInit\_ex 和 EVP\_VerifyInit\_ex 是 摘 要 函 数 EVP\_DigestInit\_ex 的 宏 定 义 ， EVP\_SignUpdate 和 EVP\_VerifyUpdate 是摘要函数 EVP\_Digest\_Update 的宏定义。

**1.签名初始化函数** **EVP\_Signlnit\_ex**

函 数EVP\_SignInit\_ex 用于签名初始化，这是签名工作的第一步，主要功能是设置摘要算 法和摘要算法引擎等。该函数是一个宏定义函数，其实际定义如下：

#define EVP\_SignInit\_ex(a,b,c)EVP\_DigestInit\_ex(a,b,c) 函数EVP\_DigestInit\_ex 声明如下：

nt EVP\_DigestInit\_ex(EVP\_MD\_CTX\*ctx,const EVP\_MD \*type,ENGINE\*impl);

其中，参数ctx[in]指向EVP\_MD\_CTX 结构体的指针；type[in] 表示要使用的摘要算法， 其值可以由表7-1所示的算法取得。

**表7-1** **摘要算法**

|  |  |
| --- | --- |
| **摘要算法** | **说** **明** |
| const EVP\_MD\*EVP\_md2(; | MD2摘要算法 |
| const EVP\_MD\*EVP\_md40; | MD4摘要算法 |
| const EVP\_MD\*EVP\_md5(); | MD5摘要算法 |
| const EVP MD\*EVP sha(); | SHA摘要算法 |
| const EVP\_MD\*EVP\_shal(; | SHA1摘要算法 |
| const EVP\_MD\*EVP\_dss(); | DSS摘要算法 |
| const EVP\_MD\*EVP\_mdc20; | MDC2摘要算法 |

参数impl[in] 表示摘要算法使用的引擎。应用程序可以使用自定义的算法引擎，如硬件摘 要算法等。如果此参数为NUL, 就使用默认引擎。如果函数执行成功就返回1,否则返回0。

**2** **.签名函数EVP\_SignUpdate**

这是签名三部曲中第二步要调用的函数，可以被多次调用，这样就可以处理大数据了。该 函数也是宏定义函数，在 openssl/evp.h 文件中声明如下：

#define EVP\_SignUpdate(a,b,c) EVP\_DigestUpdate(a,b,c)



**Windows C/C++加密解密实战**

EVP\_DigestUpdate 函数声明如下：

int EVP DigestUpdate(EVP MD CTX \*ctx,const void\*d,size t cnt);

其中，参数ctx[in] 指向摘要上下文结构体，该结构体必须是已经初始化过的了；d[in]指向 要进行摘要计算的源数据的缓冲区；cnt[in] 表示要进行摘要计算的源数据的长度，单位是字节。 如果函数执行成功就返回1,否则返回0。

**3.签名结束函数** **EVP\_SignFinal**

这是签名三部曲中的第三步要调用的函数，也是最后一步调用的函数，该函数只能调用一 次，而且是最后调用，调用完该函数，签名结果也就出来了。该函数声明如下：

int EVP SignFinal(EVP MD CTX\*ctx,unsigned char\*md,unsigned int \*s,EVP PKEY\*pkey)

其中，参数 ctx[in] 指向摘要上下文结构体，该结构体必须是已经初始化过的了；md[out] 存放输出的签名结果，对于不同的哈希算法，哈希结果长度不同，因此 md 所指向的缓冲区长 度要注意不要开辟小了；s[out] 指向整型变量的地址，该变量存放输出其签名结果的长度； pkey[in]表示签名的私钥。如果函数执行成功就返回1,否则返回0。

**4.验签初始化函数EVP\_Verifylnit\_ex**

前面理论部分讲到，因为验签的时候也要计算原文的摘要，所以这里初始化的时候肯定要 设置和签名时一样的摘要算法。

函数EVP\_VerifyInit\_ex 用于验签初始化，设置摘要算法、引擎等。该函数也是一个宏定 义函数，它和消息摘要函数是一样的，函数声明如下：

#define EVP VerifyInit ex(a,b,c)EVP DigestInit ex(a,b,c) 函数EVP\_DigestInit\_ex 声明如下：

int EVP DigestInit ex(EVP MD CTX\*ctx,constEVP MD \*type,ENGINE \*impl)

其中，参数 ctx[in]指向EVP\_MD\_CTX 结构体的指针；type[in] 表示要使用的摘要算法，

其值可以由表7-1所示的算法取得；参数 impl[in]表示摘要算法使用的引擎。应用程序可以使 用自定义的算法引擎，如硬件摘要算法等。如果此参数为NULL, 就使用默认引擎。如果函数 执行成功就返回1,否则返回0。

**5.验签函数EVP\_VerifyUpdate**

这是验签三部曲中第二步调用的函数，可以被多次调用，这样就可以处理大数据了。该函 数也是宏定义函数，在openssl/evp.h 文件中声明如下：

#define EVP VerifyUpdate(a,b,c) EVP DigestUpdate(a,b,c) EVP\_DigestUpdate 函数声明如下：

int EVP DigestUpdate(EVP MD CTX\*ctx,const void\*d,size t cnt);

其中，参数ctx[in]指向摘要上下文结构体，该结构体必须是已经初始化过了的；d[in] 指向



**第7章数字签名技术**

要进行摘要计算的源数据的缓冲区；cnt[in]表示要进行摘要计算的源数据的长度，单位是字节。 如果函数执行成功就返回1,否则返回0。

**6.验签结束函数EVP\_VerifyFinal**

该函数是验签三部曲的最后一步，只需要调用一次，该函数调用后就可以得到验签结果了， 即通过还是没通过。该函数声明如下：

int EVP VerifyFinal(EVP MD CTX\*ctx,const unsigned char \*sigbuf,unsigned int siglen,EVP PKEY \*pkey);

其中，参数ctx[in] 指向摘要上下文结构体，该结构体必须是已经初始化过了的；sigbuf[in]

存放签名值；siglen[in]指向签名者的长度；pkey[in]表示验签要使用的公钥。如果验签成功就 返 回 1 , 否 则 返 回 0 。

在 使 用 完 之 后 ，ctx 必须使用EVP\_MD\_CTX\_cleanup 函数释放内存，否则会导致内存泄

漏 。

【例7.2】EVP方式签名和验签

( 1 ) 打 开VC 2017,新建一个控制面板工程，工程名是 test。

(2)在工程中打开 test.cpp, 并输入代码如下：

#include "pch.h"

#include <stdio.h> #include <string.h>

#include<windows.h>

#include <openssl/evp.h> #include <openss/x509.h>

#ifdef WIN32

#pragma comment(lib,"libeay32.lib") #endif

|  |  |
| --- | --- |
| void tSign( { |  |
| unsigned char sign\_value[1024];  unsigned int sign\_len; EVP\_MD\_CTX mdctx;  char mess1[]="I love China!"; RSA \*rsa=NULL;  EVP\_PKEY\*evpKey=NULL; int i; | //保存签名值的数组 //签名值长度  //摘要算法上下文变量  //待签名的消息  //RSA结构体变量  //EVP KEY结构体变量 |

printf("正在产生RSA 密钥..")

rsa=RSA\_generate\_key(1024,RSA\_F4,NULL,NULL);// 产生一个1024位的RSA 密钥

if(rsa=NULL) {

printf("gen rsa err\n"); return;

}

printf(" 成功.\n")

evpKey=EVP\_PKEY\_new(); if(evpKey==NULL)

//新建一个 EVP\_PKEY 变量

**Windows** **C/C++加密解密实战**

{

printf("EVP\_PKEY\_new err\n"); RSA\_free(rsa);

return; }

if (EVP\_PKEY\_setl\_RSA(evpKey,rsa)!=1) {

printf("EVP\_PKEY\_setl\_RSA err\n"); RSA\_free(rsa);

EVP\_PKEY\_free(evpKey) return;

}

//以下计算签名代码

EVP\_MD\_CTX\_init(&mdctx)

if(!EVP\_Signlnit\_ex(&mdctx,EVP\_md5(),NULL) {

printf("err\n")

EVP\_PKEY\_free(evpKey) RSA free(rsa)

return; }

if(IEVP\_SignUpdate(&mdctx,messl,strlen(messl))) (

printf("err\n");

EVP\_PKEY\_free(evpKey); RSA\_free(rsa);

return;

//保存RSA 结构体到EVP\_PKEY 结构体

//初始化摘要上下文

//签名初始化，设置摘要算法，本例为MD5

//计算签名(摘要)更新

}

if(!EVP\_SignFinal(&mdctx,sign\_value,&sign\_len,evpKey)// 签名输出

{

printf("err\n");

EVP\_PKEY\_free(evpKey); RSA free(rsa)

return;



printf("消息\"%s\"的签名值是：\n",mess1); for (i=0;i<sign\_len;i++)

{

if(i%16==0)

printf("\n%08xH:",i)

printf("%02x",sign\_value[i]);

}

printf("\n");

//EVP\_MD\_CTX\_cleanup(&mdctx);

printf("\n正在验证签名...n"); /以下验证签名代码

EVP\_MD\_CTX\_init(&mdctx);

if(IEVP\_Verifylnit\_ex(&mdctx,EVP\_md5(0,NULL)) {

printf("EVP\_VerifyInit\_ex err\n");

EVP\_PKEY\_free(evpKey); RSA free(rsa)

return

//初始化摘要上下文

验证初始化，设置摘要算法，一定要和签名一致



**第7章数字签名技术**

if(!EVP\_VerifyUpdate(&mdctx,messl,strlen(mess1)) {

printf("err\n");

EVP\_PKEY\_free(evpKey); RSA\_free(rsa);

return }

if(!EVP\_VerifyFinal(&mdctx,sign\_yalue,sign\_len,evpKey)) {

printf("verify err\n");

EVP\_PKEY\_free(evpKey); RSA\_free(rsa);

return;

}

else {

printf(" 验证签名正确.\n");



//释放内存

EVP\_PKEY\_free(evpKey); RSA free(rsa)

//EVP\_MD\_CTX\_cleanup(&mdctx); return;

l

一

in— t main()

OpenSSL\_add\_all\_algorithms();/ 所有算法初始化函数 SignAndVerify();

return 0;

//验证签名(摘要)更新

//验证签名

一

在代码中，我们在自定义函数SignAndVerify中对原文“Ilove China!”这一段字符串先进 行签名再验签，签名要用到RSA 私钥，验签要用到RSA 公钥，所以显示生成了1024位的RSA 公私钥对，然后开始EVP 三部曲签名和验签，它们所使用的哈希函数是EVP\_md5。

最后在工程属性的附加包含目录旁添加C:\openssl-1.0.2m\inc32, 在工程属性的附加库目 录旁添加 C:\openssl-1.0.2m\out32dll。

**(3)保存工程并运行，运行结果如图7-6所示。**



**Hierosoft** **Visml** **Stadio** **调试控初白** i 问×

**在产生RsA密钥...成功.**

**隋息1** **love** **China!"的签名值是：**

:: 3 9i b3BheBb?2es6 6f3f2711coht?9e? 65

B0:Oleec8  a7d e c 0 5 c7B7bd8 i

00000040H:baGo a⁸b2 b9 h22n 7h d 424447 da23 62 c8 9009005OH:95 h⁴ht 165309881007456915 d681 oh 9003886H:f84h f1 b248 bf 4536 i dr 2faB 3c 736h d8 3806987PH:15781f 173e=a797 Bc at e?fh e68F 18 59 52

正验名.

☑





8 的

图7