# 第 3 章　密码学编程

密码学是信息安全的基础，可应用于数据加解密、数字签名、安全认证、电子投票等领域。在理解密码学基本概念的基础上，可应用经典密码算法解决实际系统中的安全问题。

## 3.1　密码学基本概念

经典密码学是指秘密书写的科学。密码(cipher)是一种秘密书写的方法。把明文(plaintext)变换为密文(ciphertext)或密报(cryptography)，这种变换叫加密(encipherment 或 encryption)。而将密文变换为明文的过程称为解密(decipherment 或 decryption)。

加密和解密都要通过密钥(Key)的控制。加密/解密示意图如图 3-1 所示。

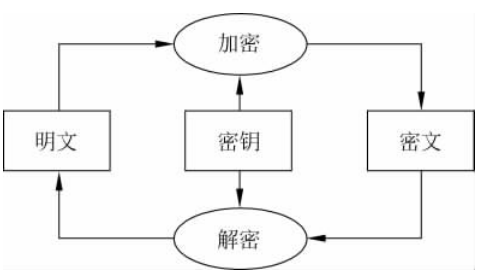


图 3-1　加密/解密示意图

密码学的研究领域可分为密码编码学(Cryptography)和密码分析学(Cryptanalysis)两个分支，分别研究密码的编制和破译问题。密码编码学和密码分析学是密码学的两个方面，两者既相互对立，又互相促进和发展。

密码编码学研究密码编码(也称为加密)、译码(也称为解密)的理论和算法。密码分析也叫做破译，密码分析学的主要任务是研究加密信息的破译或认证信息的伪造。它主要是对密码信息的解析方法进行研究。只有密码分析者才能评判密码体制的安全性。

### 3.1.1　对称密码

对称密码算法又称为传统密码算法，其主要特征是加密算法与解密算法所使用的密钥是相同的，或者从一个容易推出另一个。对称密码算法可用于保护数据的机密性和完整性，还可以扩展到身份识别等。对称密码算法在最近半个多世纪的研究中得到了迅猛发展，有很多成熟的算法可供选择。具有代表性意义的算法有两类：一类是分组密码算法；另一类是序列密码算法。分组密码算法是把明文、密文分成等长的组，然后对这些等长的组进行变换，把明文变为密文，把密文变为明文。而序列密码算法则是通过算法把密钥 $k$ 扩展为与明文或密文相一致的子密钥序列，然后明文与密文通过与该子密钥序列按位模 2 相加，把明文变为密文，把密文变为明文。代表性分组密码算法有 DES、AES。其他的对称分组密码算法包括 IDEA，RC5，CAST-128 等。

### 3.1.2　公钥密码

公钥密码算法又称为非对称密钥密码算法，其主要特征是加密密钥可以公开，而不会影

响到解密密钥的机密性。它可用于保护数据的机密性、完整性和身份识别等。

公钥密码体制也称为双密钥密码体制或非对称密码体制，与此相对应，将序列密码和分组密码等称为单密钥密码体制或对称密钥密码体制。表 3-1 总结了单钥加密和公开密钥加密的重要特征。

**表 3-1　单钥加密与公钥加密**

| 选项 | 单钥加密 | 公开密钥加密 |
|---|---|---|
| 运行条件 | ① 加密和解密使用同一密钥和同一算法<br>② 发送方和接收方必须共享密钥和算法 | ① 用同一算法进行加密和解密，而密钥有一对，其中一个用于加密，而另一个用于解密<br>② 发送方和接收方每个拥有一个相互匹配的密钥中的一个(不是另一个) |
| 安全条件 | ① 密钥必须保密<br>② 如果不掌握其他信息，要想解密报文是不可能或者至少是不现实的<br>③ 知道所用的算法加上密文的样本必须是不足以确定密钥的 | ① 两个密钥中的一个必须保密<br>② 如果不掌握其他信息，要想解密报文是不可能或者至少是不现实的<br>③ 知道所用的算法加上一个密钥和密文的样本必须不足以确定密钥 |

为了区分这两个体制，一般将单钥加密中使用的密钥称为秘密密钥(Secret Key)，公开密钥加密中使用的两个密钥分别称为公开密钥(Public Key)和私有密钥(Private Key)。在任何时候私有密钥都是保密的，但把它称为私有密钥而不是秘密密钥，以免同单钥加密中的秘密密钥混淆。

单钥密码安全的核心是通信双方秘密密钥的建立，当用户数增加时，其密钥分发就越来越困难，而且单钥密码不能满足日益膨胀的数字签名的需要。公开密钥密码编码学是在试图解决单钥加密面临的这个难题的过程中发展起来的。公共密钥密码的优点是不需要经安全渠道传递密钥，大大简化了密钥管理。它的算法有时也称为公开密钥算法或简称为公钥算法。公开密钥的应用主要有以下三方面。

(1) 加密和解密。发送方用接收方的公开密钥加密报文。

(2) 数字签名。发送方用自己的私有密钥“签署”报文。签署功能是通过对报文或者作为报文的一个函数的一小块数据应用发送者私有密钥加密完成的。

(3) 密钥交换。两方合作以便交换会话密钥。

典型的公钥密码算法包括：RSA 算法，有限域乘法群密码与椭圆曲线密码。

### 3.1.3　哈希函数

#### 1. 哈希函数的概念及应用

哈希函数是为了实现数字签名或计算消息的鉴别码而设计的。哈希函数以任意长度的消息作为输入，输入一个固定长度的二进制值，称为哈希值、杂凑值或消息摘要。从数学上看，哈希函数 $H$ 是一个映射：

$$H: Z_2^* \rightarrow Z_2^n$$

$$x \rightarrow H(x)$$

这里，$n$ 是一个给定的自然数，称为杂凑长度。用 $Z_2^m$ 表示长度为 $m$bit 的全体二进制数的集合，而 $Z_2^* = \bigcup_{m \in N} Z_2^m$。

单向哈希函数或者安全哈希函数对于消息鉴别和数字签名都是很重要的。

**2. 哈希函数的要求**

(1) $H$ 可应用于任意大小的数据块。

(2) $H$ 产生一个固定长度的输出。

(3) 对于任意给定的 $x$,计算 $H(x)$比较容易,用硬件和软件均可实现。

(4) 对于任意给定的散列码 $h$,找到满足 $H(x)=h$ 的 $x$ 在计算上是不可行的。具有这种性质的散列函数称为单向或抗原像的。

(5) 对于任意给定的分组 $x$,找到满足 $y<>x$ 且 $H(y)=H(x)$的 $y$ 在计算上是不可行的。具有这种性质的散列函数称为抗第二原像的,有时也被称为弱抗碰撞的。

(6) 找到任何满足 $H(y)=H(x)$的偶对$(x,y)$在计算上是不可行的。具有这种性质的散列函数称为抗碰撞的,有时也称为强抗碰撞的。

**3. 哈希函数的安全性**

与对称加密一样,对安全哈希函数的攻击也有两种方法:密码分析和强力攻击。与对称加密算法一样,哈希函数的密码分析设计利用算法的逻辑弱点。

**4. 哈希函数的应用**

消息认证:消息认证是用来验证消息完整性的一种机制或服务。消息认证确保收到的数据确实和发送时一样(即没有修改、插入、删除或重放)。此外,通常还要求消息认证机制确保发送方声称的身份是真实有效的。消息认证中使用哈希函数的本质如下:发送者根据待发送的消息使用该函数计算一组哈希值,然后将哈希值和消息一起发送出去。接收者收到后对于消息执行同样的哈希计算,并将结果与收到的哈希值进行对比。如果不匹配,则接收者推断出消息遭受了篡改。

口令:在基于口令的身份认证机制中,操作系统存储的是口令的哈希值,而不是口令本身,因此,获得口令文件访问的黑客并不能获取实际的口令。简单来说,当用户输入口令时,该口令的哈希值与存储在系统中的哈希值进行比较验证。这个应用要求哈希函数具有抗原像性,或许还要求抗第二原像。

入侵检测和病毒检测:在系统中为每个文件存储 $H(F)$并保护好该哈希值(例如,存储在一个受保护的 CD-R 上)。可以通过重新计算 $H(F)$确定文件是否已被修改。入侵者可能会改变 $F$,但不可能改变 $H(F)$。这个应用要求哈希函数抗第二原像。

密码学哈希函数也能够用于构建随机函数(PRF)或用作伪随机数发生器(PRNG)。

### 3.1.4 数字签名

对文件进行加密只解决了传送信息的保密问题,而防止他人对传输的文件进行破坏以及如何确定发信人的身份还需要采取其他的手段,这一手段就是数字签名。在信息安全保密系统中,数字签名技术有着特别重要的地位,信息安全服务中的源鉴别、完整性服务、不可否认服务中,都要用到数字签名技术。

数字签名算法属于公钥密码范畴,它的签名密钥是私钥,验证密钥是公钥。主要用途是完成数字签名,从而实现抵抗赖、消息鉴别和身份识别。它与公钥加密算法的公私钥生成算法是一样的,区别只是在于:公钥加密算法使用公钥进行加密,用私钥进行解密;而签名算

法中公钥和私钥的角色是对换了的，它使用私钥进行加密，公钥进行解密也即验证。

数字签名是一种类似写在纸上的普通的物理签名，但是使用了私钥加密领域的技术实现，用于鉴别数字信息的方法。一套数字签名通常定义两种互补的运算，一个用于签名，另一个用于验证。数字签字由公钥密码发展而来，它在网络安全，包括身份认证、数据完整性、不可否认性以及匿名性等方面有着重要应用。特别是在大型网络安全通信中的密钥分配、认证以及电子商务系统中都有重要的作用，数字签名的安全性日益受到高度重视。

为了保证信息的完整性与真实性，数字签名技术必须具有以下三个基本功能：发送者不能抵赖对消息的签名、接收者能够核实发送者对消息的签名，接收者不能伪造对方的签名。换句话说，数字签名技术必须具有普通签名的特点。

而数字签名的特点是它代表了消息的特征，消息如果发生改变，数字签名的值也将发生改变，不同的消息将得到不同的数字签名。安全的数字签名使接收方可以得到保证：消息确实来自发送方。因为签名的私钥只有发送方自己保存，他人无法做一样的数字签名，如果第三方冒充发送方发出一个消息，而接收方在对数字签名进行解密时使用的是发送方的公开密钥，只要第三方不知道发送方的私有密钥，加密出来的数字签名和经过计算的数字签名必然是不相同的，这就提供了一个安全的确认发送方身份的方法，即数字签名的真实性得到了保证。

假定 A 给 B 发送了一个带签名的消息 M，则 A 的数字签名必须满足下述条件。

(1) B 能够证实 A 对消息 M 的签名；

(2) 任何人，包括 B，都不能伪造 A 对 M 的签名；

(3) 如果 A 否认了 B 对 M 的签名，可以通过仲裁机构解决 A 和 B 的争议。

数字签名类似手书签名，它具有以下的性质。

(1) 能够验证签名产生者的身份，以及产生签名的日期和时间；

(2) 能用于证实被签名消息内容；

(3) 数字签名可由第三方验证，从而能够解决通信双方的争议。

为了实现数字签名的以上性质，它就应满足下列要求。

(1) 签名是可信的：任何人都可以验证签名的有效性。

(2) 签名是不可伪造的：除了合法的签名者外，任何人伪造其签名是困难的。

(3) 签名是不可复制的：对一个消息的签名不能通过复制变为另一个消息的签名，如果一个消息的签名是从别处复制得到的，则任何人都可以发现消息与签名之间的不一致性，从而可以拒绝签名的消息。

(4) 签名的消息是不可改变的：经签名的消息不能篡改，一旦签名的消息被篡改，任何人都可以发现消息与签名之间的不一致性。

(5) 签名是不可抵赖的：签名者事后不能否认自己的签名。可以由第三方或仲裁方来确认双方的信息，以做出仲裁。

为了满足数字签名的这些要求，例如，通信双方在发送消息时，既要防止接收方或其他第三方伪造，又要防止发送方因对自己的不利而否认，也就是说，要保证数字签名的真实性。

### 3.1.5　随机数与伪随机数

#### 1. 随机数的使用

许多基于密码学的网络安全算法使用了随机数。例如：

(1) RSA 公钥加密算法和其他公钥算法中的密钥生成;

(2) 对称流密码的密钥流生成;

(3) 用作临时会话密钥或创建数字信封的对称密钥的生成;

(4) 在许多密钥分配方案中,例如 KERBEROS,使用随机数进行握手以防止重放攻击;

(5) 会话密钥的生成,无论是通过密钥分配还是由主体之一完成。

这些应用对随机数提出了两种截然不同的且不能互相兼容的要求:随机性和不可预测性。

随机性:传统上,对所谓随机数字序列的生成的关注一直在与数字序列是否具有某些明确定义的统计意义上的随机性。

验证数字序列随机性的准则如下。

均匀分布性:数字序列的分布应是均匀的,即每个数的出现频率大致相等。

独立性:序列中的任何数不能由其他数推导出来。

**2. 随机与伪随机**

密码应用通常使用算法生成随机数。这些算法是确定性的,因此无法产生统计随机的数字序列。但是,如果该算法是良好的,所得到的序列将能经受住许多合理的随机性测试,这样的数字被称为伪随机数。

## 3.2 基于 SHA-1 算法的文件完整性校验

该算法实现如下功能。

(1) 编写应用程序,正确实现 SHA-1 算法。

(2) 程序不仅能够为任意长度的字符串生成 SHA-1 摘要,而且可以为任意大小的文件生成 SHA-1 摘要。

(3) 程序还可以利用 SHA-1 摘要验证文件的完整性。验证文件的完整性有两种方式:一种是手动输入 SHA-1 摘要的条件下,计算出当前被测文件的 SHA-1 摘要,再将两者进行比对;另一种是先利用系统工具 SHA-1sum 为被测文件生成一个. SHA-1 的同名文件,然后让程序计算出被测文件的 SHA-1 摘要,将其余与. SHA-1 文件中的 SHA-1 摘要进行比较,最后得出检测结果。具体要求有以下几点。

**1. 程序的输入格式**

程序为命令行程序,可执行文件名为 SHA-1. exe,命令行格式如下。

```
./SHA-1 [选项][被测文件路径][. SHA-1 文件路径]
```

其中,[选项]是程序为用户提供的各种功能。在本程序中,[选项]包括{-h,-t,-c,-v,-f}5 个基本功能。[被测文件路径]为应用程序指明被测文件所在文件系统中的路径。[. SHA-1 文件路径]为应用程序指明由被测文件生成的。其中第一个参数为必选项,后两个参数可以根据功能进行选择。

**2. 程序的执行过程**

(1) 打印帮助信息;

(2) 在控制台命令行中输入./ -h 打印程序的帮助信息；

(3) 打印测试信息；

(4) 在控制台命令行中输入./SHA-1 -t 打印程序的测试信息；

(5) 为指定文件生成 SHA-1 摘要；

(6) 在控制台命令行中输入./SHA-1 -c[被测文件路径]，计算出被测文件的 SHA-1 摘要并打印出来。

**3. 验证文件完整性方法 1**

在控制台命令行中输入./SHA-1 -c[被测文件路径]，程序会先让用户输入被测文件的 SHA-1 摘要，然后再重新计算被测文件的 SHA-1 摘要，最后将两个摘要逐位比较。若一致，则说明文件是完整的，否则，说明文件遭到了破坏。

**4. 验证文件完整性方法 2**

在控制台命令行中输入./SHA-1 -f [被测文件路径][.SHA-1 文件路径]，程序会自动读取.SHA-1 文件中的摘要，然后再重新计算出被测文件的 SHA-1 摘要，最后将两者逐位比较。若一致，则说明文件是完整的，否则，说明文件遭到了破坏。

### 3.2.1　SHA-1 算法

**1. 安全散列函数算法(SHA)**

近年来，一直使用广泛的散列函数是安全散列算法(Secure Hash Algorithm，SHA)。SHA 由美国国家标准与技术研究院(National Institute of Standards and Technology，NIST)设计，并于 1993 年作为美国联邦信息处理标准(FIPS180)发布。当 SHA 的缺点被发现后，1995 年发布了修订版 FIPS180-1，通常称之为 SHA-1。SHA-1 产生 160 位的散列值。2002 年，NIST 提出了该标准的修订版 FIPS180-2，它定义了 3 种新的 SHA 版本，散列长度分别为 256 位、384 位和 512 位，分别称为 SHA-256、SHA-384 及 SHA-512。这些新版本具有与 SHA-1 相同的基础结构，并使用相同类型的模算术运算和逻辑二进制运算。

研究人员已经证明 SHA-1 的安全性弱于它的 160 位散列长度，有必要转向使用较新版本的 SHA。

**2. SHA-1 算法原理**

SHA-1 算法由 NIST 与美国国家安全局设计，并且被美国政府采纳，成为美国国家标准。事实上，SHA-1 目前是全世界使用最为广泛的哈希算法，已经成为业界的事实标准。可以对长度不超过 $2^{64}$b 的消息进行计算，输入以 512 位数据块为单位处理，产生 160b 长的消息摘要作为输出。

1) 数据填充与分拆

在 SAH-1 中，对于输入的任意长度的消息 $X$，先把它扩充成长度(位数)为 512 的整倍数的数据：

$$\underset{\text{(原消息)}}{X \rightarrow X} \qquad \underset{\text{(填充)}}{\| 1 \| 0 \cdots 0 \|} \qquad \underset{\text{(64b)}}{(X \text{ 的长度})} \qquad L$$

再把所得数据分成 $s$ 个 512b 长的数组：

$$X = x_1 \| x_2 \| \cdots \| x_s$$

2) SHA 算法描述

(1) SHA 的初始化和主循环

SHA-1 有 5 个 32b 的链接变量 $A,B,C,D,E$。算法执行时对 $A,B,C,D,E$ 初始化为(十六进制表示):

$$A=0\text{x}67452301$$
$$B=0\text{x}efcdab89$$
$$C=0\text{x}98badcfe$$
$$D=0\text{x}10325476$$
$$E=0\text{x}c3d2e1f0$$

如图 3-2 所示给出了 SHA-1 的主循环结构图。它执行 $s$ 次循环,把链接变量的初始值,在逐次循环中变换,产生最终的哈希值。每个主循环都由 4 个轮循环组成,每轮 20 次操作,每次操作对 $a$、$b$、$c$、$d$、$e$ 中的三个进行一次非线性运算,后进行移位和加运算。$a$、$b$、$c$、$d$ 和 $e$ 分别加上 $A$、$B$、$C$、$D$ 和 $E$,然后用下一数据分组继续运行算法。最后的输出由 $A$、$B$、$C$、$D$ 和 $E$ 级联而成。

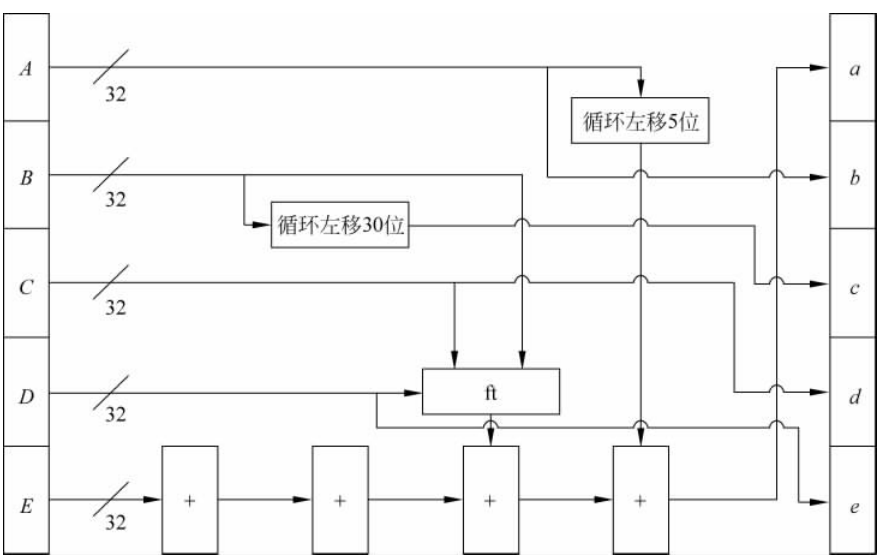


**图 3-2 SHA-1 的主循环结构**

(2) 轮函数

SHA-1 的 4 个轮函数中的每一轮都由 20 次的操作组成,4 轮共完成 80 次操作。SHA-1 中定义了三个基本逻辑函数,它们合并为一个带参数 $i$(表示操作序号)的逻辑函数,用在 4 轮的 80 个操作中。设 $X,Y,Z$ 表示 32b 的字,定义如下:

$$(X,Y,Z)=\begin{cases}(X\wedge Y)\vee(X\wedge Z) & 0\leqslant i\leqslant 19\\ X\oplus Y\oplus Z & 20\leqslant i\leqslant 39,\quad 60\leqslant i\leqslant 79\\ (X\wedge Y)\vee(X\wedge Z)\vee(X\wedge Z) & 40\leqslant i\leqslant 59\end{cases}$$

各个轮函数的输入除了链接变量外,另一个输入是 512b 的字分组的扩展。若把这 16 个 32b 字分组表示,先把它扩展为 80 次操作中的所需要的 80 个 32b 如下:

$$\begin{cases} M_i & 0 \leqslant i \leqslant 15 \\ (W_{i-3} \oplus W_{i-8} \oplus W_{i-14} \oplus W_{i-16}) <<< 1 & 16 \ll i \ll 79 \end{cases}$$

轮函数中还有 4 个常数。按照 80 次操作，它们记为：

$$K_i = \begin{cases} 0\text{x}5a827999 & 0 \leqslant i \leqslant 19 \\ 0\text{x}6ed9eba1 & 20 \leqslant i \leqslant 39 \\ 0\text{x}8f1bbcdc & 40 \leqslant i \leqslant 59 \\ 0\text{x}ca62c1d6 & 60 \leqslant i \leqslant 79 \end{cases}$$

现在已为每个操作准备了逻辑函数、32b 消息字和轮常量。这里 $i$ 对应操作序号(79)，表示循环左移 $s$b 运算，⊕表示模加法。

这时，主循环可以表示如下：

$$a = A,\quad b = B,\quad c = C,\quad d = D,\quad e = E$$

对 $i$=0 to 79 执行

$$\text{TEMP} = (a <<< 5) + f_i(b,c,d) + e + W_i + K_i$$
$$e = d$$
$$d = c$$
$$c = b <<< 30$$
$$b = a$$
$$a = \text{TEMP}$$

80 次循环后，计算 $A=a+A$，$B=b+B$，$C=c+C$，$D=d+D$，$E=e+E$。

然后，利用下一次 512b 分组进行计算，直至用完最后一个 512b 分组为止。这时变量 $A,B,C,D,E$ 的当前值的毗连：$A \parallel B \parallel C \parallel D \parallel E$，即是所要的 Hash 值。

SHA-1 算法实现流程如图 3-3 所示。

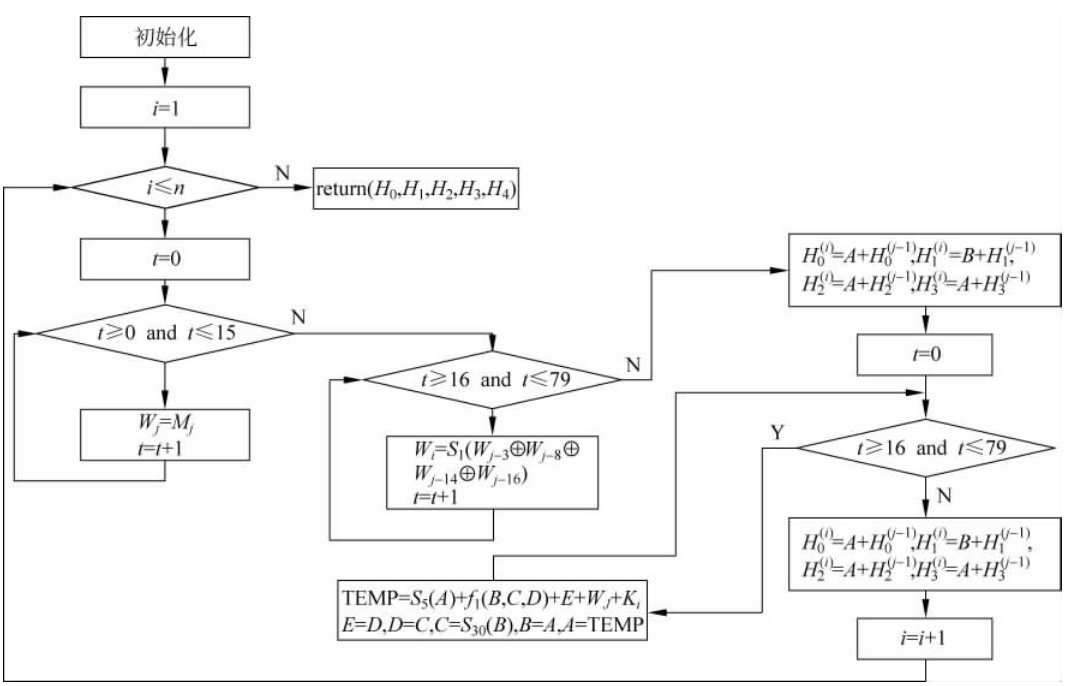


**图 3-3　SHA-1 算法的整体流程图**

3）SHA-1 算法实现

```
//SHA-1.cpp : 定义控制面板应用程序的入口点
#include "stdafx.h"
#include<stdio.h>
#include<string.h>
#include<conio.h>
#include<wtypes.h>
void creat_w(unsigned char input[64], unsigned long w[80])
{
int i, j; unsigned long temp, temp1;
for (i = 0; i<16; i++)
{
    j = 4 * i;
    w[i] = ((long)input[j]) << 24 | ((long)input[1 + j]) << 16 | ((long)input[2 + j]) << 8 |
((long)input[3 + j]) << 0;
}
for (i = 16; i<80; i++)
{
    w[i] = w[i - 16] ^ w[i - 14] ^ w[i - 8] ^ w[i - 3];
    temp = w[i] << 1;
    temp1 = w[i] >> 31;
    w[i] = temp | temp1;
}
}
char ms_len(long a, char intput[64])
{
unsigned long temp3, p1; int i, j;
temp3 = 0;
p1 = ~(~temp3 << 8);
for (i = 0; i<4; i++)
{
    j = 8 * i;
    intput[63 - i] = (char)((a&(p1 << j)) >> j);
}
return '0';
}
int _tmain(int argc, _TCHAR* argv[])
{
unsigned long H0 = 0x67452301, H1 = 0xefcdab89, H2 = 0x98badcfe, H3 = 0x10325476,
H4 = 0xc3d2e1f0;
unsigned long A, B, C, D, E, temp, temp1, temp2, temp3, k, f; int i, flag; unsigned long w[80];
unsigned char input[64]; long x; int n;
printf("input message:\n");
scanf("%s", input);
n = strlen((LPSTR)input);
if (n<57)
{
    x = n * 8;
    ms_len(x, (char*)input);
```

```
    if (n == 56)
          for (i = n; i<60; i++)
              input[i] = 0;
    else
    {
          input[n] = 128;
          for (i = n + 1; i<60; i++)
              input[i] = 0;
    }
}
creat_w(input, w);
/*for(i=0;i<80;i++)
printf("%lx,",w[i]);*/
printf("\n");
A = H0; B = H1; C = H2; D = H3; E = H4;
for (i = 0; i<80; i++)
{
    flag = i / 20;
    switch (flag)
    {
    case 0: k = 0x5a827999; f = (B&C) | (~B&D); break;
    case 1: k = 0x6ed9eba1; f = B^C^D; break;
    case 2: k = 0x8f1bbcdc; f = (B&C) | (B&D) | (C&D); break;
    case 3: k = 0xca62c1d6; f = B^C^D; break;
    }
    /*printf("%lx,%lx\n",k,f);*/
    temp1 = A<<5;
    temp2 = A>>27;
    temp3 = temp1 | temp2;
    temp = temp3 + f + E + w[i] + k;
    E = D;
    D = C;
    temp1 = B<<30;
    temp2 = B>>2;
    C = temp1 | temp2;
    B = A;
    A = temp;
    //printf("%lx,%lx,%lx,%lx,%lx\n",A,B,C,D,E);    //输出编码过程
}
H0 = H0 + A;
H1 = H1 + B;
H2 = H2 + C;
H3 = H3 + D;
H4 = H4 + E;
printf("\noutput hash value:\n");
printf("%lx%lx%lx%lx%lx", H0, H1, H2, H3, H4);
getch();
}
```

运行结果如图 3-4 所示。


**图 3-4 基于 SHA 计算消息摘要运行结果图**

### 3.2.2 基于 SHA-1 的文件完整性检验

**1. 基本步骤**

文件完整性检验在 main()函数中实现。应用程序为用户提供了多个选项,不但可以在命令行下计算文件的 SHA-1 摘要,验证文件的完整性,还可以显示程序的帮助信息和 SHA-1 算法的测试信息。

在 main 函数中,程序通过区分参数 argv[1]的不同值来启动不同的工作流程。如果 argv[1]等于"-h",表示显示帮助信息;如果 argv[1]等于"-t",表示显示测试信息;如果 argv[1]等于"-c",表示计算被测文件的 SHA-1 摘要;如果 argv[1]等于"-v",表示根据手工输入的 SHA-1 摘要验证文件的完整性;如果 argv[1]等于"-h",表示根据.SHA-1 文件中的摘要验证文件的完整性。

帮助信息可以协助用户快速地了解命令行输入格式。测试信息可以让用户验证 SHA-1 算法的正确性。用于测试的消息字符串都是 SHA-1 算法官方文档(RFC1321)中给出的例子,如果计算的结果相同,则说明程序的 SHA-1 运算过程正确无误。

程序提供了两种验证文件完整性的方式:一种是让用户手工输入被测文件的 SHA-1 摘要,然后调用 SHA-1 类的运算函数重新计算被测文件的 SHA-1 摘要,最后将两个摘要逐位进行比较,进而验证文件的完整性;另一种是从与被测文件对应的 SHA-1 文件中读取 SHA-1 摘要,然后调用 SHA-1 类的运算函数重新计算被测文件的 SHA-1 摘要,最后将两个摘要逐位进行比较,进而验证文件的完整性。

手工输入验证分为以下 6 个步骤。

(1) 首先比较参数 argv[1],判断是否通过手工输入进行验证。若是,则继续下面的步骤;否则,退出。

(2) 检测被测文件的路径是否存在,若存在,则继续下面的步骤;否则,退出。

(3) 打开被测文件的 SHA-1 摘要并保存在数组 InputSHA-1 中。

(4) 打开被测文件,读取被测文件的内容,并调用 Update 函数重新计算被测文件的 SHA-1 摘要。

(5) 调用 Tostring 函数将 SHA-1 摘要表示成十六进制字符串的形式。

(6) 最后调用 strcmp 函数判断两个摘要是否相同,若相同,则说明被测文件是完整的;否则,说明文件受到了破坏。

**2. 实现代码**

```
//头文件
#include "SHA-1.h"
```

```
#include <iostream>
using namespace std;
//功能函数
//Main 函数
//
int main(int argc,char * argv[])
{
   char * pFilePath;                    //需要进行 SHA-1 计算的文件路径
   char * pSHA-1FilePath;               //存放 SHA-1 摘要的 SHA-1 文件路径
   char SHA-1Digest[33];                //SHA-1 摘要,用于存放手动输入的 SHA-1 摘要信息
   char SHA-1Record[50];                //SHA-1 文件中的一行记录
   string strTmp;                       //字符串定义
       char * pHelpMsg = {"-h"};        //帮助信息
       char * pTestMsg = {"-t"};        //SHA-1 测试程序的应用信息
       char * pCompute = {"-c"};        //计算指定文件的 SHA-1 摘要
       char * pMValidate = {"-mv"};     //手动对文件进行 SHA-1 认证
       char * pfValidate = {"-fv"};     //通过比较对文件的 SHA-1 摘要进行认证
       char * pSpace = {" "};           //定义空格

       //参数检测
       if(argc<2 || argc>4)
       {
            cout<<"Parameter Error!"<<endl;
            return -1;
       }
     //显示帮助信息
        if((argc == 2)&&(!strcmp(pHelpMsg,argv[1])))
        {
              cout<<"SHA-1 usage: [-h] --help information"<<endl;
              cout<<" [-t] --test SHA-1 application"<<endl;
              cout<<" [-c] [file path of the file computed]"<<endl;
              cout<<" --compute SHA-1 of the given file"<<endl;
              cout<<" [-mv] [file path of the file validated]"<<endl;
               cout<<" --validate the integrality of a given file by manual input SHA-1
value"<<endl;
cout<<" [-fv] [file path of the file validated] [file path of the .SHA-1 file]"<<endl;
              cout<<" --validate the integrality of a given file by read SHA-1 value from
.SHA-1 file"<<endl;
        }
        //显示 SHA-1 应用程序的测试信息
  if((argc == 2)&&(!strcmp(pTestMsg,argv[1])))
        {
              cout<<"SHA-1(\"\") = "<<SHA-1("").toString()<<endl;
              cout<<"SHA-1(\"a\") = "<<SHA-1("a").toString()<<endl;
              cout<<"SHA-1(\"abc\") = "<<SHA-1("abc").toString()<<endl;
              cout<<"SHA-1(\"message digest\") = "<<SHA-1("message digest").toString()<<
endl;
              cout<<"SHA-1(\"abcdefghijklmnopqrstuvwxyz\") = "<<SHA-1("abcdefghijklmn
opqrstuvwxyz").toString()<<endl;
```

```
            cout <<"SHA - 1(\"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345
6789\")"<< endl;
            cout <<" = "<< SHA - 1("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0
123456789").toString()<< endl;
            cout <<"SHA - 1(\"12345678901234567890123456789012345678901234567890123456
78901234567890123456789O\")"<< endl;
cout <<" = "<< SHA - 1("12345678901234567890123456789012345678901234567890123456789012345
5678901234567890").toString()<< endl;
}
//计算指定文件的 SHA - 1 摘要,并显示出来
        if((argc == 3)&&(!strcmp(pCompute,argv[1])))
        {
            //如果没有文件路径,则参数出错
            if(argv[2] == NULL)
            {
                cout <<"Parameter Error ! Please input file path !"<< endl;
                return - 1;
            }
            else
            {
                pFilePath = argv[2];
            }
            //打开指定的文件
            ifstream File_1(pFilePath);
            //声明 SHA - 1 对象,并进行计算
            SHA - 1 SHA - 1_obj1(File_1);
            //输出计算结果
            cout <<"SHA - 1(\""<< argv[2]<<"\") = "<< SHA - 1_obj1.toString()<< endl;
        }
            //手动进行文件完整性检测
        if((argc == 3)&&(!strcmp(pMValidate,argv[1])))
        {
            //如果没有文件路径,则参数出错
            if(argv[2] == NULL)
            {
                cout <<"Parameter Error ! Please input file path !"<< endl;
                return - 1;
            }
            else
            {
                pFilePath = argv[2];
            }
        //手动输入了被测文件的 SHA - 1 摘要
            cout <<"Please input the SHA - 1 value of file(\""<< pFilePath <<"\")..."<< endl;
            cin >> SHA - 1Digest;
            //在摘要的字符串末尾加上结束符
            SHA - 1Digest[32] = '\0';
            //打开指定的文件
            ifstream File_2(pFilePath);
            //声明 SHA - 1 对象,并进行计算
```

```
            //SHA-1 SHA-1_obj2(File_2);
            SHA-1 SHA-1_obj2;
            SHA-1_obj2.reset();
            SHA-1_obj2.update(File_2);
            //读取文件内容并计算 SHA-1 摘要
strTmp = SHA-1_obj2.toString( );
const char* pSHA-1Digest = strTmp.c_str( );
//输出两个摘要
cout<<"The SHA-1 digest of file(\""<<pFilePath<<"\") which you input is: "<<endl;
cout<<SHA-1Digest<<endl;
cout<<"The SHA-1 digest of file(\""<<pFilePath<<"\") which calculate by program is: "<<endl;
            cout<<strTmp<<endl;
                        //比较摘要的结果是否相同
            if (strcmp(pSHA-1Digest,SHA-1Digest))
            {
                  cout<<"Match Error! The file is not integrated!"<<endl;
            }
            else
            {
                  cout<<"Match Successfully! The file is integrated!"<<endl;
            }
        }
        //通过 SHA-1 文件进行文件完整性检测
        if((argc == 4)&&(!strcmp(pfValidate,argv[1])))
        {
            //如果没有文件路径,则参数出错
            if((argv[2] == NULL)||(argv[3] == NULL))
            {
                  cout<<"Parameter Error ! Please input file path !"<<endl;
                  return -1;
            }
            else
            {
                  pFilePath = argv[2];
                  pSHA-1FilePath = argv[3];
            }
        //打开 SHA-1 文件
            ifstream File_3(pSHA-1FilePath);
            //读取 SHA-1 文件中的记录
            File_3.getline(SHA-1Record,50);
        //以空格为标记,获得 SHA-1 文件中的 SHA-1 值与对应文件名
        char* pSHA-1Digest_f = strtok(SHA-1Record,pSpace);
        char* pFileName_f = strtok(NULL,pSpace);
              //打开被测文件
          ifstream File_4(pFilePath);
            //声明 SHA-1 对象,并进行计算
            //SHA-1 SHA-1_obj3(File_4);
            SHA-1 SHA-1_obj3;
            SHA-1_obj3.reset();
            SHA-1_obj3.update(File_4);
```

```
            //读取文件内容并计算 SHA-1 摘要
                strTmp = SHA-1_obj3.toString();
                const char* pSHA-1Digest_c = strTmp.c_str();
                //输出两个摘要
                cout<<"The SHA-1 digest of file(\""<<pFileName_f<<"\") which is in file(\""<<
    pSHA-1FilePath<<"\") is: "<<endl;
                cout<<pSHA-1Digest_f<<endl;
                cout<<" The SHA-1 digest of file(\""<<pFilePath<<"\") which calculate by
    programme is: "<<endl;
                cout<<strTmp<<endl;
                    //比较摘要,进行验证
        if (strcmp(pSHA-1Digest_c,pSHA-1Digest_f))
            {
                    cout<<"Match Error! The file is not integrated!"<<endl;
            }
            else
            {
                    cout<<"Match Successfully! The file is integrated!"<<endl;
            }
        }
        //函数返回
        return 0;
    }
```

通过 SHA-1 文件进行验证,与手工输入验证类似,也可以分为以下 6 个步骤。

(1) 首先比较参数 argv[1],判断是否通过 SHA-1 文件进行验证。若是,则继续下面的步骤;否则,退出。

(2) 检测被测文件的路径和 SHA-1 文件的路径是否存在,若存在,则继续下面的步骤;否则,退出。

(3) 打开 SHA-1 文件,读取文件中的记录,调用 strtok 函数获得被测文件的 SHA-1 摘要。

(4) 打开被测文件,读取被测文件的内容,并调用 Update 函数重新计算被测文件的 SHA-1 摘要。

(5) 调用 Tostring 函数将 SHA-1 摘要表示成十六进制字符串的形式。

(6) 最后调用 strcmp 函数判断两个摘要是否相同,若相同,则说明被测文件是完整的;否则,说明文件受到了破坏。

## 3.3 基于 RSA 算法实现数据加解密

该算法要求实现以下目的。

(1) 理解 RSA 算法的基本工作原理。

(2) 掌握实现 RSA 算法的编程方法。

(3) 掌握基于 RSA 算法实现数据加解密的工作原理和实现方法。

RSA 算法是一种非对称加密算法，它大概是世界上使用最为广泛的公钥密码体制了，当然，它也是最著名的，因为它能够同时提供数字签名方案和公钥加密方案，从而使其成为一个非常通用的算法工具。

### 3.3.1　RSA 算法原理

此部分主要讲述了 RSA 算法的三部分内容，分别是 RSA 算法数学理论基础，RSA 算法的原理实现以及基于 RSA 算法实现数据加解密。

**1. RSA 算法的数学基础**

RSA 算法具有非常严谨的数学理论基础，现描述如下。

(1) **定理 1**　算术基本定理。任何一个不等于 0 的正整数 $n$ 都可以写成唯一的表达式，即

$$n = p_1^{k_1} \cdots p_r^{k_r} \tag{3-1}$$

这里 $p_r > p_2 > p_3 > \cdots > p_r$ 是素数，其中，$k_i > 0$。

(2) **定义 1**　欧拉函数 $\varphi(n)$。$\varphi(n)$是少于或等于 $n$ 的数中与 $n$ 互质的数的数目，将 $n$ 分解为素数互乘的形式 $n = p_1^{k_1} \cdots p_r^{k_r}$，每个 $p_i$ 都是素数，则

$$\varphi(n) = p_1^{k_1}\left(1-\frac{1}{p_1}\right)p_2^{k_2}\left(1-\frac{1}{p_2}\right)\cdots p_r^{k_r}\left(1-\frac{1}{p_r}\right) \tag{3-2}$$

由此可以得出以下两条结论。

① 若 $n$ 为质数，则 $\varphi(n) = n-1$；

② 若 $m$ 与 $n$ 互质，则 $\varphi(mn) = \varphi(m)\varphi(n)$。

(3) **定理 2**　欧拉定理。若整数 $a$ 与整数 $n$ 互素，则 $a^{\varphi(n)} \equiv 1 (\mathrm{mod}\ n)$。该定理有以下三个推论。

① 当 $p$ 为素数时，且 $n = p$ 时，有 $a^{p-1} \equiv 1 (\mathrm{mod}\ p)$，即为 Fermat 定理：

$$a^{\varphi(n+1)} \equiv a (\mathrm{mod\ n}) \tag{3-3}$$

② 若 $n = pq$，且 $p$ 与 $q$ 为互异素数，取 $0 < m < n$，若 $(m, n) = 1$，有 $m^{\varphi(n)} \equiv m (\mathrm{mod}\ n)$，即为

$$m^{(p-1)(q-1)+1} \equiv m (\mathrm{mod}\ n) \tag{3-4}$$

**2. RSA 算法的原理**

RSA 算法主要包括三个部分：公私密钥的生成、加密过程及解密过程，具体算法过程如下。

1) 公钥和私钥的产生

假设 Alice 想要通过一个不可靠的媒体接收 Bob 的一条私人讯息，她可以用以下的方式来产生一个公钥和一个私钥。

(1) 随意选择两个大的质数 $p$ 和 $q$，$p$ 不等于 $q$，计算 $N = pq$；

(2) 根据欧拉函数，不大于 $N$ 且与 $N$ 互质的整数个数为 $(p-1)(q-1)$；

(3) 选择一个整数 $e$ 与 $(p-1)(q-1)$ 互质，并且 $e$ 小于 $(p-1)(q-1)$；

(4) 用以下公式计算 $d$：

$$d * e \equiv 1 (\mathrm{mod}(p-1)(q-1)) \tag{3-5}$$

(5) 将 $p$ 和 $q$ 的记录销毁。

$(N,e)$是公钥,$(N,d)$是私钥。$(N,d)$是秘密的。Alice 将她的公钥$(N,e)$传给 Bob,而将她的私钥$(N,d)$藏起来。

2) 加密消息

假设 Bob 想给 Alice 发送一个消息 $m$,他知道 Alice 产生的 $N$ 和 $e$。他使用起先与 Alice 约好的格式将消息 $m$ 转换为一个小于 $N$ 的整数 $n$,比如他可以将 $m$ 中的每一个字转换为相应的 Unicode 码,然后将这些 Unicode 码连在一起组成一个数字。假如他的信息非常长的话,他可以将这个信息分为几段,然后将每一段信息字符加密。用下面这个公式他可以将 $n$ 加密为 $c$:

$$n^e \equiv c(\text{mod } N) \tag{3-6}$$

计算 $c$ 并不复杂,Bob 算出 $c$ 后就可以将它传递给 Alice。

3) 解密消息

Alice 得到 Bob 的消息 $c$ 后就可以利用她的密钥 $d$ 来解码。她可以用以下这个公式来将 $c$ 转换为 $n$。

$$c^d \equiv n(\text{mod } N) \tag{3-7}$$

得到 $n$ 后,可以将原来的信息 $m$ 重新复原。

解码的原理是:$c^d \equiv n^{e-d}(\text{mod } N)$和 $ed \equiv 1(\text{mod}(q-1))$以及 $ed \equiv 1(\text{mod}(p-1))$。由费马小定理可证明(因为 $p$ 和 $q$ 是质数):

$$n^{e-d} \equiv n(\text{mod } p) \quad \text{和} \quad n^{e-d} \equiv n(\text{mod } q) \quad \text{以及} \quad n^{e-d} \equiv n(\text{mod } pq)$$

**3. RSA 算法的安全性**

理论上,RSA 的安全性取决于因式分解模 $n$ 的困难性。从严格的技术角度上来说这是不正确的,在数学上至今还未证明分解模数就是攻击 RSA 的最佳方法。事实情况是,大整数因子分解问题过去数百年来一直是令数学家头疼而未能有效解决的世界性难题。人们设想了一些非因子分解的途径攻击 RSA 体制,但这些方法都不比分解 $n$ 来得容易。因此,严格地说,RSA 的安全性基于求解其单向函数的逆的困难性。RSA 单向函数求逆的安全性没有真正因式分解模数 $n$ 的安全性高,而且目前人们也无法证明这两者等价。许多研究人员都试图改进 RSA 体制使它的安全性等价于因式分解模数 $n$。

**4. 针对 RSA 算法的攻击手段**

下面列出了常见的针对 RSA 算法的攻击方法。

1) 对 RSA 分解模数 $n$ 攻击

分解模数 $n$ 是最直接的攻击方法,也是最困难的方法。攻击者可以获得公开密钥 $e$ 和模数 $n$;如果模数 $n=pq$ 被因式分解,则攻击者通过 $p$、$q$ 便可计算出 $f(n)=(p-1)(q-1)$,进而 $de \equiv 1(\text{mod } f(n))$而得到解密密钥 $d$。如果密码分析者能够不分解 $n$ 而直接求得 $f(n)$,则可根据 $de \equiv 1(\text{mod } f(n))$求得解密密钥 $d$,从而破译 RSA。又因为:

$$p+q = n-f(n)+1$$

$$p-q = \sqrt{(p+q)^2-4n}$$

所以知道 $f(n)$和 $n$ 就可以容易地求得 $p$ 和 $q$,从而成功地分解 $n$。目前已经出现了不对 $n$ 进行因子分解而直接估算 $f(n)$的攻击方法,但还没证明,直接计算 $f(n)$比对 $n$ 进行因子分

解更容易。大整数分解研究一直是数论与密码理论研究的重要课题，随着计算能力的增强和因子分解算法的不断完善，为保证 RSA 的安全性，在实际应用中对 $p$ 和 $q$ 的选取要求也越来越高。

2）RSA 的小指数攻击

这类攻击专门针对 RSA 算法的实现细节。采用小的 $e$、$d$ 可以加快加密和验证签名的速度，而且所需的存储空间小，但是如果 $e$、$d$ 太小，则容易受到小指数攻击（Low Encryption/Decryption Exponent Attack），包括低加密指数攻击和低解密指数攻击。通过独立随机数字对明文消息 $x$ 进行填充，可以有效地抗击小指数攻击。

3）耗时攻击

这种攻击是通过监视一台计算机解密消息所花费的时间来确定解密指数。RSA 的基本运算是乘方取模，这种运算的特点是耗费时间精确，这样如果破译者能够监视到 RSA 解密的过程，并对它计时，他就能算出 $d$。关于如何防御这种攻击，最简单的方法莫过于使 RSA 解密时花费均等的时间，而与解密指数 $d$ 无关。其次在加密前对数据做一个变换（花费恒定时间），在解密时做逆变换，这样总时间也不再依赖于解密指数 $d$。另外，耗时攻击对攻击者资源的要求太高，目前还不实用，但从理论上说是一个崭新的思路。

### 3.3.2　基于 RSA 算法实现数据加解密

根据上述 RSA 算法原理，RSA 算法实现过程以及基于 RSA 算法实现数据加解密的实现代码如下。

```
//RSA.CPP:定义控制面板应用程序的入口点
#include "stdafx.h"
#include<stdio.h>
#include<string.h>
#include<stdlib.h>
#include<time.h>
#include<math.h>
#include<malloc.h>
#define MAX 100
#define LEN sizeof(struct slink)
void sub(int a[MAX], int b[MAX], int c[MAX]);
struct slink
{
int bignum[MAX];
//bignum[98]用来标记正负号,1 正,0 负 bignum[99]来标记实际长
struct slink *next;
};
//大数运算
void print(int a[MAX])
{
int i;
for (i = 0; i<a[99]; i++)
      printf("%d", a[a[99] - i - 1]);
printf("\n\n");
return;
```

```
}
int cmp(int a1[MAX], int a2[MAX])
{
int l1, l2;
int i;
l1 = a1[99];
l2 = a2[99];
if (l1 > l2)
      return 1;
if (l1 < l2)
      return -1;
for (i = (l1 - 1); i >= 0; i-- )
{
      if (a1[i]> a2[i])
            return 1;
      if (a1[i]< a2[i])
            return -1;
}
return 0;
}
void mov(int a[MAX], int *b)
{
int j;
for (j = 0; j < MAX; j++)
      b[j] = a[j];
return;
}
//大数相乘(向左移)
void mul(int a1[MAX], int a2[MAX], int *c)
{
int i, j, y, x, z, w, l1, l2;
l1 = a1[MAX - 1];
l2 = a2[MAX - 1];
if (a1[MAX - 2] == '-'&& a2[MAX - 2] == '-')
      c[MAX - 2] = 0;
else if (a1[MAX - 2] == '-')
      c[MAX - 2] = '-';
else if (a2[MAX - 2] == '-')
      c[MAX - 2] = '-';
for (i = 0; i < l1; i++)
{
    for (j = 0; j < l2; j++)
    {
          x = a1[i] * a2[j];
          y = x / 10;
          z = x % 10;
          w = i + j;
          c[w] = c[w] + z;
          c[w + 1] = c[w + 1] + y + c[w] / 10;
          c[w] = c[w] % 10;
```

```
    }
}
w = l1 + l2;
if (c[w - 1] == 0)w = w - 1;
c[MAX - 1] = w;
return;
}
//大数相加,注意进位
void add(int a1[MAX], int a2[MAX], int * c)
{
int i, l1, l2;
int len, temp[MAX];
int k = 0;
l1 = a1[MAX - 1];
l2 = a2[MAX - 1];
if ((a1[MAX - 2] == '-') && (a2[MAX - 2] == '-'))
{
      c[MAX - 2] = '-';
}
else if (a1[MAX - 2] == '-')
{
    mov(a1, temp);
    temp[MAX - 2] = 0;
    sub(a2, temp, c);
    return;
}
else if (a2[MAX - 2] == '-')
{
    mov(a2, temp);
    temp[98] = 0;
    sub(a1, temp, c);
    return;
}
if (l1 < l2)len = l1;
else len = l2;
for (i = 0; i < len; i++)
{
    c[i] = (a1[i] + a2[i] + k) % 10;
    k = (a1[i] + a2[i] + k) / 10;
}
if (l1 > len)
{
    for (i = len; i < l1; i++)
    {
          c[i] = (a1[i] + k) % 10;
          k = (a1[i] + k) / 10;
    }
    if (k != 0)
    {
        c[l1] = k;
```

```
        len = l1 + 1;
    }
    else len = l1;
}
else
{
    for (i = len; i<l2; i++)
    {
          c[i] = (a2[i] + k) % 10;
          k = (a2[i] + k) / 10;
    }
    if (k != 0)
    {
        c[l2] = k;
        len = l2 + 1;
    }
    else len = l2;
}
c[99] = len;
return;
}
//大数相减,注意借位
void sub(int a1[MAX], int a2[MAX], int *c)
{
int i, l1, l2;
int len, t1[MAX], t2[MAX];
int k = 0;
l1 = a1[MAX - 1];
l2 = a2[MAX - 1];
if ((a1[MAX - 2] == '-') && (a2[MAX - 2] == '-'))
{
    mov(a1, t1);
    mov(a2, t2);
    t1[MAX - 2] = 0;
    t2[MAX - 2] = 0;
    sub(t2, t1, c);
    return;
}
else if (a2[MAX - 2] == '-')
{
    mov(a2, t2);
    t2[MAX - 2] = 0;
    add(a1, t2, c);
    return;
}
else if (a1[MAX - 2] == '-')
{
    mov(a2, t2);
    t2[MAX - 2] = '-';
    add(a1, t2, c);
```

```
        return;
    }
    if (cmp(a1, a2) == 1)
    {
        len = l2;
        for (i = 0; i < len; i++)
        {
            if ((a1[i] - k - a2[i])< 0)
            {
                  c[i] = (a1[i] - a2[i] - k + 10) % 10;
                  k = 1;
            }
            else
            {
                  c[i] = (a1[i] - a2[i] - k) % 10;
                  k = 0;
            }
        }
        for (i = len; i < l1; i++)
        {
            if ((a1[i] - k)< 0)
            {
                  c[i] = (a1[i] - k + 10) % 10;
                  k = 1;
            }
            else
            {
                  c[i] = (a1[i] - k) % 10;
                  k = 0;
            }
        }
    if (c[l1 - 1] == 0)     //使得数组 C 中的前面所有 0 字符不显示了,如 1000 - 20 = 0980 -- ->显
                            //示为 980 了
        {
            len = l1 - 1;
            i = 2;
            while (c[l1 - i] == 0)//111456 - 111450 = 00006,消除 0 后变成了 6
            {
                len = l1 - i;
                i++;
            }
        }
        else
        {
            len = l1;
        }
    }
    else
        if (cmp(a1, a2) == ( - 1))
        {
```

```
            c[MAX - 2] = '-';
            len = l1;
            for (i = 0; i<len; i++)
            {
                if ((a2[i] - k - a1[i])<0)
                {
                      c[i] = (a2[i] - a1[i] - k + 10) % 10;
                      k = 1;
                }
                else
                {
                      c[i] = (a2[i] - a1[i] - k) % 10;
                      k = 0;
                }
            }
            for (i = len; i<l2; i++)
            {
                if ((a2[i] - k)<0)
                {
                      c[i] = (a2[i] - k + 10) % 10;
                      k = 1;
                }
                else
                {
                      c[i] = (a2[i] - k) % 10;
                      k = 0;
                }
            }
            if (c[l2 - 1] == 0)
            {
                len = l2 - 1;
                i = 2;
                while (c[l1 - i] == 0)
                {
                    len = l1 - i;
                    i++;
                }
            }
            else len = l2;
        }
        else if (cmp(a1, a2) == 0)
        {
            len = 1;
            c[len - 1] = 0;
        }
    c[MAX - 1] = len;
    return;
    }
    //取模数
```

```
void mod(int a[MAX], int b[MAX], int *c)     //c = a mod b,注意：根据检验知道此处 A 和 C 的
                                             //数组都改变了
{
int d[MAX];
mov(a, d);
while (cmp(d, b) != (-1))                    //c = a-b-b-b-b-b…until(c<b)
{
    sub(d, b, c);
    mov(c, d);                               //c 复制给 a
}
return;
}
//大数相除(向右移)
//试商法,调用以后 w 为 a mod b, C 为 a div b
void divt(int t[MAX], int b[MAX], int *c, int *w)
{
int a1, b1, i, j, m;                         //w 用于暂时保存数据
int d[MAX], e[MAX], f[MAX], g[MAX], a[MAX];
mov(t, a);
for (i = 0; i<MAX; i++)
    e[i] = 0;
for (i = 0; i<MAX; i++)
    d[i] = 0;
for (i = 0; i<MAX; i++) g[i] = 0;
a1 = a[MAX - 1];
b1 = b[MAX - 1];
if (cmp(a, b) == (-1))
{
    c[0] = 0;
    c[MAX - 1] = 1;
    mov(t, w);
    return;
}
else if (cmp(a, b) == 0)
{
    c[0] = 1;
    c[MAX - 1] = 1;
    w[0] = 0;
    w[MAX - 1] = 1;
    return;
}
m = (a1 - b1);
for (i = m; i>= 0; i--)//例如//341245/3 = 341245 - 300000 * 1 --->41245 - 30000 * 1 --->
11245 - 3000 * 3 --->2245 - 300 * 7 --->145 - 30 * 4 = 25 --->//25 - 3 * 8 = 1
{
    for (j = 0; j<MAX; j++)
        d[j] = 0;
    d[i] = 1;
    d[MAX - 1] = i + 1;
    mov(b, g);
```

```
    mul(g, d, e);
    while (cmp(a, e) != (-1))
    {
        c[i]++;
        sub(a, e, f);
        mov(f, a);                          //f 复制给 g
    }
    for (j = i; j<MAX; j++)                 //高位清零
        e[j] = 0;
}
mov(a, w);
if (c[m] == 0) c[MAX - 1] = m;
else c[MAX - 1] = m + 1;
return;
}
//解决了 m = a * b mod n
void mulmod(int a[MAX], int b[MAX], int n[MAX], int *m)
{
int c[MAX], d[MAX];
int i;
for (i = 0; i<MAX; i++)
    d[i] = c[i] = 0;
mul(a, b, c);
divt(c, n, d, m);
for (i = 0; i<m[MAX - 1]; i++)
    printf("%d", m[m[MAX - 1] - i - 1]);
printf("\nm length is : %d \n", m[MAX - 1]);
}
//解决了 m = a^p mod n 的函数问题
void expmod(int a[MAX], int p[MAX], int n[MAX], int *m)
{
int t[MAX], l[MAX], temp[MAX];              //t 放入 2,l 放入 1
int w[MAX], s[MAX], c[MAX], b[MAX], i;
for (i = 0; i<MAX - 1; i++)
    b[i] = l[i] = t[i] = w[i] = 0;
t[0] = 2; t[MAX - 1] = 1;
l[0] = 1; l[MAX - 1] = 1;
mov(l, temp);
mov(a, m);
mov(p, b);
while (cmp(b, l) != 0)
{
    for (i = 0; i<MAX; i++)
        w[i] = c[i] = 0;
    divt(b, t, w, c);                       //c = p mod 2 w = p /2
    mov(w, b);                              //p = p/2
    if (cmp(c, l) == 0)                     //余数 c == 1
    {
        for (i = 0; i<MAX; i++)
            w[i] = 0;
```

```
            mul(temp, m, w);
            mov(w, temp);
            for (i = 0; i < MAX; i++)
                w[i] = c[i] = 0;
            divt(temp, n, w, c);        //c 为余 c = temp % n,w 为商 w = temp/n
            mov(c, temp);
        }
        for (i = 0; i < MAX; i++)
            s[i] = 0;
        mul(m, m, s);                   //s = a * a
        for (i = 0; i < MAX; i++)
            c[i] = 0;
        divt(s, n, w, c);               //w = s/n;c = s mod n
        mov(c, m);
    }
    for (i = 0; i < MAX; i++)
        s[i] = 0;
    mul(m, temp, s);
    for (i = 0; i < MAX; i++)
        c[i] = 0;
    divt(s, n, w, c);
    mov(c, m);                          //余数 s 给 m
    m[MAX - 2] = a[MAX - 2];            //为后面的汉字显示需要,用第 99 位作为标记
    return;
    //k = temp * k % n
    }
    int   is_prime_san(int p[MAX])
    {
    int i, a[MAX], t[MAX], s[MAX], o[MAX];
    for (i = 0; i < MAX; i++)
        s[i] = o[i] = a[i] = t[i] = 0;
    t[0] = 1;
    t[MAX - 1] = 1;
    a[0] = 2;                           //{ 2,3,5,7 }
    a[MAX - 1] = 1;
    sub(p, t, s);
    expmod(a, s, p, o);
    if (cmp(o, t) != 0)
    {
        return 0;
    }
    a[0] = 3;
    for (i = 0; i < MAX; i++) o[i] = 0;
    expmod(a, s, p, o);
    if (cmp(o, t) != 0)
    {
        return 0;
    }
    a[0] = 5;
    for (i = 0; i < MAX; i++) o[i] = 0;
    expmod(a, s, p, o);
```

```
if (cmp(o, t) != 0)
{
    return 0;
}
a[0] = 7;
for (i = 0; i < MAX; i++) o[i] = 0;
expmod(a, s, p, o);
if (cmp(o, t) != 0)
{
    return 0;
}
return 1;
}
//检测两个大数之间是否互质
int coprime(int e[MAX], int s[MAX])
{
int a[MAX], b[MAX], c[MAX], d[MAX], o[MAX], l[MAX];
int i;
for (i = 0; i < MAX; i++)
    l[i] = o[i] = c[i] = d[i] = 0;
o[0] = 0; o[MAX - 1] = 1;
l[0] = 1; l[MAX - 1] = 1;
mov(e, b);
mov(s, a);
do
{
    if (cmp(b, l) == 0)
    {
        return 1;
    }
    for (i = 0; i < MAX; i++)
        c[i] = 0;
    divt(a, b, d, c);
    mov(b, a);                  //b --->a
    mov(c, b);                  //c --->b
} while (cmp(c, o) != 0);
//printf("They are not coprime!\n")
return 0;
}
//产生随机素数 p 和 q
void prime_random(int *p, int *q)
{
int i, k;
time_t t;
p[0] = 1;
q[0] = 3;
//p[19] = 1;
//q[18] = 2;
p[MAX - 1] = 10;
q[MAX - 1] = 11;
```

```
do
{
    t = time(NULL);
    srand((unsigned long)t);
    for (i = 1; i<p[MAX - 1] - 1; i++)
    {
        k = rand() % 10;
        p[i] = k;
    }
    k = rand() % 10;
    while (k == 0)
    {
        k = rand() % 10;
    }
    p[p[MAX - 1] - 1] = k;
} while ((is_prime_san(p)) != 1);
printf("素数 p 为 : ");
for (i = 0; i<p[MAX - 1]; i++)
{
    printf(" %d", p[p[MAX - 1] - i - 1]);
}
printf("\n\n");
do
{
    t = time(NULL);
    srand((unsigned long)t);
    for (i = 1; i<q[MAX - 1]; i++)
    {
        k = rand() % 10;
        q[i] = k;
    }
} while ((is_prime_san(q)) != 1);
printf("素数 q 为 : ");
for (i = 0; i<q[MAX - 1]; i++)
{
    printf(" %d", q[q[MAX - 1] - i - 1]);
}
printf("\n\n");
return;
}
//产生与(p-1)*(q-1)互素的随机数
void erand(int e[MAX], int m[MAX])
{
int i, k;
time_t t;
e[MAX - 1] = 5;
printf("随机产生一个与(p-1)*(q-1)互素的 e :");
do
{
    t = time(NULL);
```

```
    srand((unsigned long)t);
    for (i = 0; i<e[MAX - 1] - 1; i++)
    {
        k = rand() % 10;
        e[i] = k;
    }
    while ((k = rand() % 10) == 0)
        k = rand() % 10;
    e[e[MAX - 1] - 1] = k;
} while (coprime(e, m) != 1);
for (i = 0; i<e[MAX - 1]; i++)
{
    printf(" %d", e[e[MAX - 1] - i - 1]);
}
printf("\n\n");
return;
}
//根据上面的 p、q 和 e 计算密钥 d
void rsad(int e[MAX], int g[MAX], int *d)
{
int  r[MAX], n1[MAX], n2[MAX], k[MAX], w[MAX];
int    i, t[MAX], b1[MAX], b2[MAX], temp[MAX];
mov(g, n1);
mov(e, n2);
for (i = 0; i<MAX; i++)
    k[i] = w[i] = r[i] = temp[i] = b1[i] = b2[i] = t[i] = 0;
b1[MAX - 1] = 0; b1[0] = 0;                 //b1 = 0;
b2[MAX - 1] = 1; b2[0] = 1;                 //b2 = 1;
while (1)
{
    for (i = 0; i<MAX; i++)
        k[i] = w[i] = 0;
    divt(n1, n2, k, w);                     //k = n1/n2;
    for (i = 0; i<MAX; i++)
        temp[i] = 0;
    mul(k, n2, temp);                       //temp = k * n2;
    for (i = 0; i<MAX; i++)
        r[i] = 0;
    sub(n1, temp, r);
    if ((r[MAX - 1] == 1) && (r[0] == 0))   //r = 0
    {
        break;
    }
    else
    {
        mov(n2, n1);                        //n1 = n2;
        mov(r, n2);                         //n2 = r;
        mov(b2, t);                         //t = b2;
        for (i = 0; i<MAX; i++)
            temp[i] = 0;
```

```
            mul(k, b2, temp);                //b2 = b1 - k * b2;
            for (i = 0; i < MAX; i++)
                b2[i] = 0;
            sub(b1, temp, b2);
            mov(t, b1);
        }
    }
    for (i = 0; i < MAX; i++)
        t[i] = 0;
    add(b2, g, t);
    for (i = 0; i < MAX; i++)
        temp[i] = d[i] = 0;
    divt(t, g, temp, d);
    printf("由以上的(p - 1) * (q - 1)和 e 计算得出的 d : ");
    for (i = 0; i < d[MAX - 1]; i++)
        printf(" %d", d[d[MAX - 1] - i - 1]);
    printf("\n\n");
    }
    //求解密密钥 d 的函数(根据欧几里得算法)
    unsigned long rsa(unsigned long p, unsigned long q, unsigned long e)   //求解密密钥 d 的函数
                                                                           //(根据欧几里得算法)
    {
    unsigned long g, k, r, n1, n2, t;
    unsigned long b1 = 0, b2 = 1;
    g = (p - 1) * (q - 1);
    n1 = g;
    n2 = e;
    while (1)
    {
        k = n1 / n2;
        r = n1 - k * n2;
        if (r != 0)
        {
            n1 = n2;
            n2 = r;
            t = b2;
            b2 = b1 - k * b2;
            b1 = t;
        }
        else
        {
            break;
        }
    }
    return (g + b2) % g;
    }
    //加密和解密
    void printbig(struct slink * h)
    {
    struct slink * p;
```

```
int i;
p = (struct slink *)malloc(LEN);
p = h;
if (h != NULL)
    do
    {
        for (i = 0; i<p->bignum[MAX - 1]; i++)
            printf("%d", p->bignum[p->bignum[MAX - 1] - i - 1]);
        p = p->next;
    }
while (p != NULL);
    printf("\n\n");
}
struct slink *input(void)          //输入明文并且返回头指针,没有加密的时候转化的数字
{
struct slink *head;
struct slink *p1, *p2;
int i, n, c, temp;
char ch;
n = 0;
p1 = p2 = (struct slink *)malloc(LEN);
head = NULL;
printf("\n请输入你所要加密的内容:\n");
while ((ch = getchar()) != '\n')
{
    i = 0;
    c = ch;
    if (c<0)
    {
        c = abs(c);                //把负数取正并且做一个标记
        p1->bignum[MAX - 2] = '0';
    }
    else
    {
        p1->bignum[MAX - 2] = '1';
    }
    while (c / 10 != 0)
    {
        temp = c % 10;
        c = c / 10;
        p1->bignum[i] = temp;
        i++;
    }
    p1->bignum[i] = c;
    p1->bignum[MAX - 1] = i + 1;
    n = n + 1;
    if (n == 1)
        head = p1;
    else p2->next = p1;
    p2 = p1;
```

```
    p1 = (struct slink *)malloc(LEN);
}
p2->next = NULL;
return(head);
}
//加密模块,例如 C = P^e mod n
struct slink *jiami(int e[MAX], int n[MAX], struct slink *head)
{
struct slink *p;
struct slink *h;
struct slink *p1, *p2;
int m = 0, i;
printf("\n");
printf("加密后形成的密文内容: \n");
p1 = p2 = (struct slink*)malloc(LEN);
h = NULL;
p = head;
if (head != NULL)
    do
    {
        expmod(p->bignum, e, n, p1->bignum);
        for (i = 0; i<p1->bignum[MAX - 1]; i++)
        {
            printf("%d", p1->bignum[p1->bignum[MAX - 1] - 1 - i]);
        }
        m = m + 1;
        if (m == 1)
            h = p1;
        else p2->next = p1;
        p2 = p1;
        p1 = (struct slink *)malloc(LEN);
        p = p->next;
    } while (p != NULL);
    p2->next = NULL;
    p = h;
    printf("\n");
    return(h);
}
//解密模块,例如 P = C^d mod n
void jiemi(int d[MAX], int n[MAX], struct slink *h)
{
int i, j, temp;
struct slink *p, *p1;
char ch[65535];
p1 = (struct slink*)malloc(LEN);
p = h;
j = 0;
if (h != NULL)
    do
    {
```

```
        for (i = 0; i < MAX; i++)
            p1 -> bignum[i] = 0;
        expmod(p -> bignum, d, n, p1 -> bignum);
        temp = p1 -> bignum[0] + p1 -> bignum[1] * 10 + p1 -> bignum[2] * 100;
        if ((p1 -> bignum[MAX - 2]) == '0')
        {
            temp = 0 - temp;
        }
        ch[j] = temp;
        j++;
        p = p -> next;
    } while (p != NULL);
    printf("\n");
    printf("解密密文后所生成的明文:\n");
    for (i = 0; i < j; i++)
        printf("%c", ch[i]);
    printf("\n");
    return;
}
//选择一种操作
void menu()
{
printf("R--------产生 密钥对\n");
printf("T--------简单测试\n");
printf("Q--------退出\n");
printf("请选择一种操作:");
}
//主函数
void main()
{
int i;
char c;
int p[MAX], q[MAX], n[MAX], d[MAX], e[MAX], m[MAX], p1[MAX], q1[MAX];
struct slink *head, *h1, *h2;
for (i = 0; i < MAX; i++)
    m[i] = p[i] = q[i] = n[i] = d[i] = e[i] = 0;          //简单初始化一下
while (1)
{
    menu();
    c = getchar();
    getchar();                                            //接受回车符
    if ((c == 'R') || (c == 'r'))                         //操作 r 产生密钥对
    {
        for (i = 0; i < MAX; i++)
            m[i] = p[i] = q[i] = n[i] = d[i] = e[i] = 0;
        printf("\n\n 随机密钥对产生如下: \n\n");
        prime_random(p, q);                               //随机产生两个大素数
        mul(p, q, n);                                     //p * q
```

```
            printf("由 p、q 得出 n :");
            print(n);
            mov(p, p1);
            p1[0]--;                        //p-1
            mov(q, q1);
            q1[0]--;                        //q-1
            mul(p1, q1, m);                 //m=(p-1)*(q-1)
            erand(e, m);                    //产生一个与 m 互素的随机数
            rsad(e, m, d);                  //e * d = 1 mod m
            printf("密钥对产生完成,现在可以直接进行加解密!\n");
            printf("\n 按任意键回主菜单…");
            getchar();
        }
        else if ((c == 'T') || (c == 't'))   //加密解密测试
        {
            head = input();
            h1 = jiami(e, n, head);
            jiemi(d, n, h1);
            printf("\nRSA 测试工作完成!\n");
            printf("\n 按任意键回主菜单……");
            getchar();
        }
        else if ((c == 'Q') || (c == 'q'))                  //结束退出
        {
            break;
        }
    }
    }
```

运行结果如图 3-5 所示。


**图 3-5　RSA 算法及基于 RSA 实现数据加解密运行结果图**

# 小　结

本章首先介绍了密码学的基本概念，讲述了经典密码算法 SHA-1 算法以及 RSA 算法的原理及实现，在此基础上基于经典密码算法 SHA-1 实现文件完整性验证，利用 RSA 算法实现数据加解密。

# 思 考 题

1. 密码系统的基本组成有哪些？
2. 公钥密码与对称密码的主要区别是什么？
3. 哈希函数具备哪些性质？
4. 编程实现基于口令身份识别中的口令散列过程(基于 SHA-1 算法)。
5. 随机数的用途是什么？如何编程产生随机数？
6. 编程实现基于 RSA 的文件加解密。