（续表）

参　数	说　明
-validity	指定创建的证书有效期为多少天，缺省为 90
-keysize	指定密钥长度，缺省为 1024
-storepass	指定密钥库的密码（获取 keystore 信息所需的密码）
-keypass	指定别名条目的密码（私钥的密码）
-dname	指定证书拥有者信息，例如： "CN=名字与姓氏,OU=组织单位名称,O=组织名称,L=城市或区域名称,ST=州或省份名称,C=两字母国家代码"
-list	显示密钥库中的证书信息
-v	显示密钥库中的证书详细信息
-export	将别名指定的证书导出到文件，如“keytool -export -alias 别名 -keystore 密钥库 -file 证书文件 -storepass 密码”
-file	参数指定导出到文件的文件名，若不指定，读时为标准输入，写时为标准输出
-delete	删除密钥库中的某条目，如“keytool -delete -alias 别名 -keystore 密钥库 -storepass 密码”
-printcert	查看导出的证书信息，如“keytool -printcert -file me.crt”
-keypasswd	修改密钥库中指定条目口令。如“keytool -keypasswd -alias 别名-keypass 旧密码-new 新密码-storepass 密码-keystore 密钥库”
-storepasswd	修改 keystore 口令。如“keytool -storepasswd -keystore 密钥库-storepass 原始密码-new 新密码”
-import	将已签名数字证书导入密钥库。如“keytool -import -alias 别名-keystore 密钥库-file 需导入的证书”

2．生成 keystore

在当前目录产生密钥库 mytest.keystore，执行如下命令：

```
keytool -genkey -alias mytest -keypass test123 -keyalg RSA -keysize 1024 -validity 365 -keystore mytest.
keystore -storepass pass123 -dname "CN=mytest, OU=dev, O=bjca, L=haidian, ST=beijing, C=CN"
```

显示已经产生的密钥库内容，执行如下命令：

```
keytool -list -v -keystore mytest.keystore -storepass pass123
```

显示内容为：

```
Keystore 类型：JKS
Keystore 提供者：SUN
别名名称：mytest
创建日期：2014-2-12
项类型：PrivateKeyEntry
认证链长度：1
认证[1]：
所有者：CN=mytest, OU=dev, O=bjca, L=haidian, ST=beijing, C=CN
签发人：CN=mytest, OU=dev, O=bjca, L=haidian, ST=beijing, C=CN
序列号：52fb3b75
有效期：Wed Feb 12 17:14:29 CST 2014 至 Thu Feb 12 17:14:29 CST 2015
证书指纹：
    MD5:2B:83:F8:44:96:C4:5B:75:47:C0:E4:D0:47:96:88:23
    SHA1:BD:94:1A:59:0B:A3:F2:33:52:E1:4C:FA:41:0F:D6:56:30:F7:42:84
签名算法名称：SHA1withRSA
版本：3
```

在产生密钥库的过程中，产生了一个 RSA 算法的自签名证书，其密钥长度为 1024 位，使用 SHA1withRSA 签名算法。

当然，可以不使用自签名证书，而是产生 RSA 密钥，然后产生证书请求，把证书请求提交到 CA 发证机构签发证书，然后把签发后的证书导入密钥库。

12.3.2.2　读取密钥库中的证书数据代码

在密钥库建立完成后，就可以用代码读取其中的证书了。

```
//需要包含的包
import java.security.*;
import java.io.*;
import java.util.*;
import java.security.*;
import java.security.cert.*;
import sun.security.x509.*
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
//从密钥库中直接读取证书
String pass="pass123";
String alias= "mytest";
FileInputStream in=new FileInputStream("mytest.keystore");
KeyStore ks=KeyStore.getInstance("JKS");
ks.load(in,pass.toCharArray());
java.security.cert.Certificate c=ks.getCertificate(alias);
//显示证书指定信息
System.out.println("输出证书信息:\n"+c.toString());
System.out.println("版本号:"+t.getVersion());
System.out.println("序列号:"+t.getSerialNumber().toString(16));
System.out.println("使用者：  "+t.getSubjectDN());
System.out.println("签发者：  "+t.getIssuerDN());
System.out.println("有效期：  "+t.getNotBefore());
System.out.println("签名算法：  "+t.getSigAlgName());
```

12.3.3　使用私钥

使用私钥进行签名的示例代码如下：

```
//需要包含的包
import java.security.*;
import java.io.*;
import java.util.*;
import java.security.*;
import java.security.cert.*;
import sun.security.x509.*
```

```
import java.security.cert.Certificate;
import java.security.cert.CertificateFactory;
import java.security.PrivateKey;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignatureException;
//从密钥库中读取私钥
String pass="pass123";
String alias= "mytest";
String keypass= "test123";
FileInputStream in=new FileInputStream("mytest.keystore");
KeyStore ks=KeyStore.getInstance("JKS");
ks.load(in,pass.toCharArray());
PrivateKey prk=(PrivateKey)ks.getKey(alias, keypass.toCharArray());
Signature rsa = Signature.getInstance("SHA1withRSA");
rsa.initSign(prk);
// Update and sign the data
Byte[] data="this is to be signed text".getBytes();
rsa.update(data);
byte[] sig = rsa.sign();
```

12.4 CNG

12.4.1 CNG 简介

Windows Vista 引入了新的加密 API 以替代旧的 CryptoAPI，旧的 CryptoAPI 存在于早期版本的 Windows NT 系列和 Windows 95。下一代加密技术（CNG）旨在长期替代 CryptoAPI，取代 CryptoAPI 提供的所有加密基元或服务。CNG 支持 CryptoAPI 提供的所有算法，而且应用更广泛并且包括许多新算法和更灵活的设计，从而为开发人员提供了对如何执行加密操作，以及算法如何协同工作以执行各种操作的更强的控制能力。

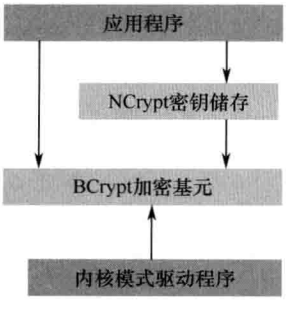

图 12-5　CNG 体系结构

图 12-5 说明了 CNG 的总体设计。BCrypt 是 CNG 的子集，提供加密基元，如随机数字生成、哈希函数、签名和加密密钥。NCrypt 也是 CNG 的子集，但它提供密钥存储工具，以支持永久保存非对称密钥和硬件（如智能卡）。从 BCrypt 和 NCrypt 的命名并不能了解太多内容。BCrypt 仅仅是头文件和为 CNG 提供基本服务的 DLL 的名称。在这种情况下，“B”代表“base”。同样，NCrypt 仅仅是头文件和提供高级别密钥存储功能的 DLL 的名称。“N”代表“new”。

BCrypt 提供的加密基元可在内核模式下直接使用，第一次为用户模式和内核模式应用程序提供公用加密框架，而 NCrypt 提供的密钥存储工具只能用于用户模式应用程序。

有两种主要方法可查看 CNG 提供的加密基元。第一种方法是作为一组逻辑对象，这些对象提供可以调用的方法和可以查询（有时可以修改）的属性。这些对象包括算法提供程序、哈希函数、密钥和密码协议。随机数生成、签名和不同类型的密钥来自哪里？实际上，随机数生成由算法提供程序直接处理，而密钥提供几乎所有其他项。它们可以表示对称或非对称密钥，并用于签发和验证哈希签名。哈希对象和密钥对象从算法提供程序派生而来，而密码协议从一对密钥对象（希望安全地进行通信的一个主体的公钥和另一个主体的私钥）派生而来。

查看 CNG 的另一种方法是作为软件路由器或加密操作的中介。CNG API 基于一组逻辑加密接口构建。例如，可以使用哈希接口，而无需对特定哈希算法实现进行硬编码。大多数加密 API 采用以算法为中心的方法，而 CNG 采用以接口为中心的方法。这为开发人员和应用程序管理员提供了更大的灵活性，在发布的应用程序中使用的算法有缺陷时，他们可以替换该算法。图 12-6 说明了这种以接口为中心的 CNG 视图。

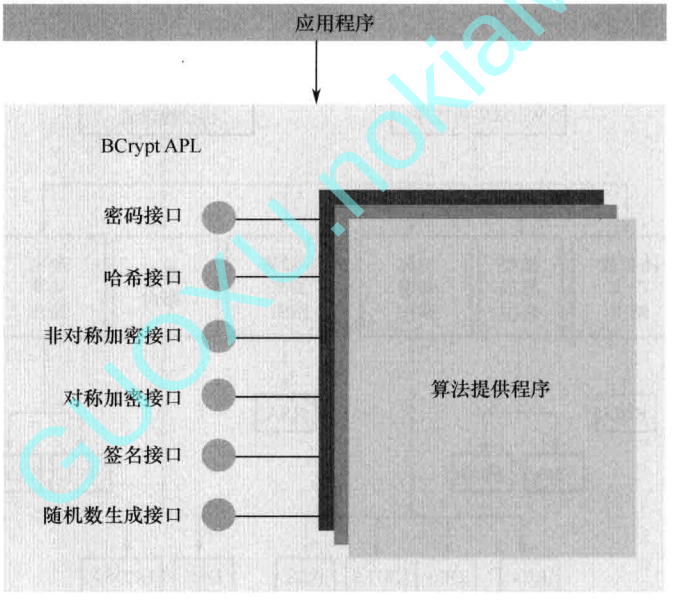

图 12-6　CNG 接口

CNG 允许开发人员不必指定算法提供程序即可请求算法，提供程序负责算法实现，而正是 CNG 的这一方面允许管理员重新配置应用程序以使用不同的实现，无论在应用程序中以声明的方式使用，还是在整个系统策略中使用。

1．CNG 功能特点

① CNG 允许客户使用各自的加密算法或执行标准加密算法。客户还可以添加新的算法。

② CNG 支持采用内核模式加密。为了完全支持加密功能，同时在内核模式和用户模式下使用相同的 API。除了使用 CNG 的启动过程以外，SSL/TLS 与 IPsec 也在内核模式下操作。

③ CNG 已通过 FIPS 140-2 2 级认证。

④ CNG 在安全过程中使用并存储长效密钥。

⑤ CNG 支持当前的 CryptoAPI 1.0 算法。

⑥ CNG 支持椭圆曲线加密（ECC）算法。

⑦ 任何具备受信任的平台模块（TPM）的计算机均可在 TPM 中提供密钥隔离和密钥存储。

2. CNG 加密基元结构

CNG 封装多种加密算法。每一种算法或算法类输出基元 API。一个给定的算法的多种实现可以同时共存，但是一个算法只有一个默认实现。

CNG 中每个算法类可以表示为基元路由。在用户模式下使用基元函数的应用程序链接到 BCrypt.dll 二进制路由文件，在内核模式的应用程序链接到 Ksecdd.sys 路由文件。各种路由器组件管理所有的算法基元。这些路由跟踪系统安装的每个算法实现，以及跟踪算法基元实现者的函数调用。

图 12-7 说明了 CNG 加密基元的结构和功能。

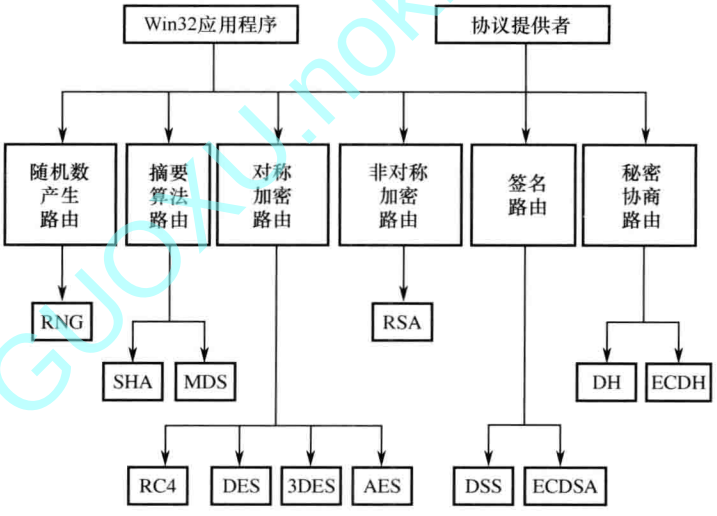

图 12-7　CNG 加密基元的结构和功能

3. CNG 密钥存储架构

CNG 提供的密钥存储模型能够满足现在和将来使用公钥或私钥进行加密的应用需求。密钥存储路由是这个模型的核心部件并在 Ncrypt.dll 实现。应用程序通过密钥存储路由访问安装在系统上的密钥存储提供者（Key Storage Providers，KSP），密钥存储路由封装了实现细节。图 12-8 显示了 CNG 密钥隔离架构的设计和功能。

依据 CC（Common Criteria）通用标准的要求，长时间存在的密钥必须被隔离，使它们永远不会出现在应用程序中。目前，CNG 使用了 Windows Server 2008 和 Windows Vista 及后续 Windows 版本中默认安装的微软软件 KSP，以支持非对称私钥的存储。

默认情况下，在 Windows Server 2008 和 Windows Vista 及后续版本中启用了密钥隔离。此外，密钥隔离服务（LSA 进程中）不会加载第三方 KSP，只会加载微软的 KSP。