电子科技大学
UNIVERSITY OF ELECTRONIC SCIENCE AND TECHNOLOGY OF CHINA

硕士学位论文

MASTER DISSERTATION

论文题目 基于PKCS#11协议的安全平台关键技术研究与实现

学科专业 计算机软件与理论
指导教师 佘堃 副教授
作者姓名 沈仟
班学号 01S06088

分类号_____ 密级_____

UDC_____

学 位 论 文

基于 PKCS#11 协议的安全平台关键技术研究与实现

（题名和副题名）

沈 仟

（作者姓名）

指导教师姓名 余 堃

副教授 电子科技大学计算机学院

（职务、职称、学位、单位名称及地址）

申请学位级别 硕士 专业名称 计算机软件与理论

论文提交日期 2003/12/20 论文答辩日期 2004/3/8

学位授予单位和日期_____

答辩委员会主席_____

评阅人_____

2003 年 12 月 25 日

注 1 注明《国际十进分类法 UDC》的类号

独 创 性 声 明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。据我所知，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得电子科技大学或其它教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

签名：_____ 日期： 年 月 日

关于论文使用授权的说明

本学位论文作者完全了解电子科技大学有关保留、使用学位论文的规定，有权保留并向国家有关部门或机构送交论文的复印件和磁盘，允许论文被查阅和借阅。本人授权电子科技大学可以将学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后应遵守此规定）

签名：_____ 导师签名：_____

日期： 年 月 日

摘要

随着网络的日益普及，人们渐渐习惯于从网上获取需要的信息。但是这也带来信息的安全性问题。密码学是安全的基石，任何应用安全都是建筑其上的。因此作为一个底层的服务，其通用性、可靠性及性能就显得尤为重要。我们知道目前国内的大多数安全设备公司都是在做着同样的一件事情－－生产硬件密码设备，为了商业的考虑，这些安全设备的接口互不统一，这就为上层的二次开发者带来了极大的不便。他们往往为了换设备不得不修改已有的程序代码，使得后期的维护比较困难。

PKCS 是美国 RSA 数据安全公司为公钥密码学提供的一个工业标准接口。PKCS 包含一系列标准，而 PKCS#11 是一个加密设备接口标准，它详细规定了一个称为 Cryptoki 的编程接口，可以用于各种可移植的安全设备。Cryptoki 给出了一个通用的逻辑模型，用户不需要知道详细的技术细节就可以在可移植的设备上完成加密操作。PKCS#11 规范的提出主要有两个目的：一个目的是屏蔽各种安全实现之间的差异，向用户提供统一的接口，使安全服务的具体实现方法对用户透明化；另一个目的是使用户可以在多个安全服务具体实现之间共享资源，使一个设备可以被多个应用程序访问，一个应用程序也可以访问多个设备。

本项目开发的基于 PKCS#11 协议的产品 WPKCS#11 正是基于上面的思想诞生的。它提供了一个与底层硬件设备、操作系统无关的通用接口，屏蔽了硬件之间的差异，将复杂的密码运算封装成简单易懂的高层服务，并根据 PKI 的其他标准解决了密码运算的互操作问题。它减少程序员的开发和维护的工作量，提高了软件产品的可重用性。同时为开发硬件设备 Token 提供了快速开发模板。

关键词： PKCS#11,插槽,令牌,对象,接口

Abstract

With the rapaid prevalence of internet, people are be accustomed to get more information from internet step by step, and then security problems are raised. Cryptanalysis is the base of information security, and all applications are constructed on it. So, as a rock-bottom service, its common、reliability、capability become so important. As we know that, native corporations of information security are all do the same thing: product devices of cipher, for some pursome, these devices' interface are not uniform, and it is hard for above user or developers. They have to modify codes to change devices, and it is also hard for upper maintenance.

PKCS is an interface for public key cryptanalysis rasied by RSA Co. PKCS includes a series of protocols, and PKCS#11 protocol is a standard for cipher devices. It describle a program interface called Cryptoki, and it can be used for all kinds of devices that can be transplanted. The cryptoki rasied a common logical model. Users can use device even if they don't know the technology details about device. We rasied PKCS#11 protocl for two aims: one is that it can mask the difference of different devices, and gives an uniform interface for users; Second is that it can make share security service for many users, and also an application can acess more than one device.

WPKCS#11 product we developed is based on PKCS#11 protocol and above ideas. It providers a common interface, it masks the difference of devices and OS, and encapsulates the complicate interface to simple service. This reduces the work of programs' development and maintenance, and makes it easy reusing software. It also provider a template for rapaid development of devices' logical Token..

Key Words: PKCS#11,Slot,Token,Object,Interface

目 录

第一章 背景技术介绍

1.1 安全技术简述

信息安全技术主要包括两个方面：信息的存储安全和信息的传输安全。信息的存储安全不言而喻就是指信息在静态存放状态下的安全，如是否会被非授权地调用等。

实现访问控制是解决信息存储安全的主要方式。现阶段技术主要是通过设置访问权限、身份识别、局部隔离等措施来保证这方面的安全。由于访问控制是针对“外部”的访问、调用而言，而网络世界中，无论是任何调用指令，还是任何信息反馈均是通过网络传输实现的，因而在网络世界中信息网络传输上的安全就显得尤为重要。

信息的传输安全是指信息在动态传输过程中的安全。主要防止的问题如：

1．对网络上信息的监听

由于现阶段数据的传输多是以明文方式在网上传输的，攻击者只需在网络的传输链路上通过物理或逻辑的手段，就能对数据进行非法的截获与监听，进而得到用户或服务方的敏感信息。

2．对用户身份的仿冒

用户身份仿冒是最常见的一种网络攻击方式，传统的对策一般是靠用户的登录密码来对用户身份进行认证；但用户的密码在登录时也是以明文的方式在网络上进行传输的，很容易就能被攻击者在网络上截获，进而可以对用户的身份进行仿冒，使身份认证机制被攻破。

3．对网络上信息的篡改

攻击者有可能对网络上的信息进行截获并且篡改其内容（增加、截去或改写），使用户无法获得准确、有用的信息或落入攻击者的陷阱。

4．对发出的信息予以否认

某些用户可能对自己发出的信息进行恶意的否认，例如否认自己发出的转帐信息等。

5．对信息进行重发

除了以上情况之外，还存在“信息重发"的攻击方式，即攻击者截获网络上的

密文信息后，并不将其破译，而是把这些数据包再次向有关服务器（如银行的交易服务器）发送，以实现恶意的目的。

总而言之，实现信息在网络传输上的保密性、系统身份认证的可靠性、数据传输的完整性、信息的不可抵赖性、信息的唯一性等等是保证信息安全的关键所在。

无疑，正象大家所预见的，互联网正在影响这个世界，世界也必将因它发生巨大的变革。而安全问题的解决无疑是这一变革的一块基石。

经过世界信息安全界的多年研究、探索，发现加密、数字签名和认证等“古老”的密码学方面的现代研究成果是现代信息安全的基础。

1.2 数据加密认证技术

1.2.1 加密

加密的目的是为了保证消息发送者(sender)能够讲消息安全的发送给消息接收者(receiver)，而第三方无法阅读发送的消息。加密过程为：通过函数转换 c=E(m,k1)发送者将消息明文 m 通过密钥 k1 变换成消息密文 c；解密过程为：通过另一函数转换 m=D(c,k2)，接受者将消息密文 c 通过密钥 k2 恢复成明文消息 m；根据 k1 和 k2 是否相同加密算法可以分为对称加密算法和非对称加密算法两类。非对称加密算法也称为公开密钥算法。

1.2.1.1 对称加密

对称算法(symmetric algorithm)有时又叫传统密码算法，就是加密密钥能够从解密密钥中推算出来，反过来也成立。在大多数对称算法中，加/解密密钥是相同的。过程如下：

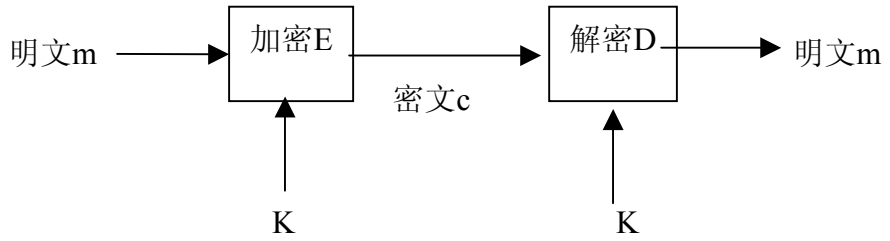

图 1－1　对称加密

目前常用的对称算法有：DES、Triple DES、IDEA、RC2、RC4、RC5 等。

1.2.1.2 非对称加密

公开密钥算法(pub-key alporithm，也叫非对称算法)，用作加密的密钥不同于用作解密的密钥，而且解密密钥不能根据加密密钥计算出来(至少在合理假定的长时间内)。所以叫公开密钥算法，是因为加密密钥能够公开，即陌生者能用加密密钥加密信息，但只有用相应的解密密钥才能解密信息。在这系统中加密密钥叫做公开密钥(pub-key，简称公钥)，解密密钥叫做私人密钥(pri-key，简称私钥)。过程如下：

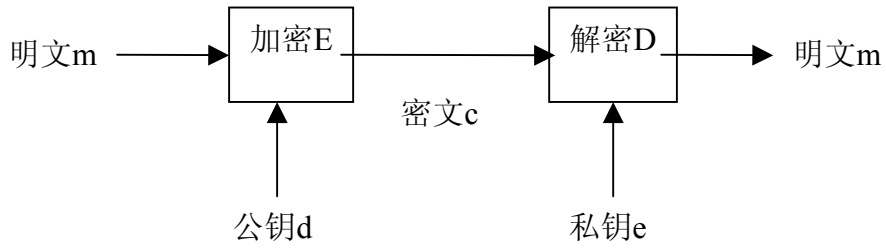

图 1－2　非对称加密

目前常用的非对称算法有 RSA、DSA、DH 等。

1.2.2 数字签名和身份认证

数字签名技术是实现交易安全的核心技术之一，它的实现基础就是加密技术。以往的书信或文件是根据亲笔签名或印章来证明其真实性的。但在计算机网络中传送的报文又如何盖章呢？这就是数字签名所要解决的问题。数字签名必须保证以下几点：接收者能够核实发送者对报文的签名；发送者事后不能抵赖对报文的签名；接收者不能伪造对报文的签名。现在已有多种实现各种数字签名的方法，但采用公开密钥算法要比常规算法更容易实现。

发送者 A 用其秘密解密密钥 SKA 对报文 X 进行运算，将结果 DSKA（X）传送给接收者 B。B 用已知的 A 的公开加密密钥得出 EPKA（DSKA（X））=X。因为除 A 外没有别人能具有 A 的解密密钥 SKA，所以除 A 外没有别人能产生密文 DSKA（X）。这样，报文 X 就被签名了。

假若 A 要抵赖曾发送报文给 B。B 可将 X 及 DSKA（X）出示给第三者。第

三者很容易用 PKA 去证实 A 确实发送消息 X 给 B。反之，如果是 B 将 X 伪造成 X'，则 B 不能在第三者面前出示 DSKA（X'）。这样就证明 B 伪造了报文。可以看出，实现数字签名也同时实现了对报文来源的鉴别。

但是上述过程只是对报文进行了签名。对传送的报文 X 本身却未保密。因为截到密文 DSKA（X）并知道发送者身份的任何人，通过查问手册即可获得发送者的公开密钥 PKA，因而能够理解报文内容。则可同时实现秘密通信和数字签名。SKA 和 SKB 分别为 A 和 B 的秘密密钥，而 PKA 和 PKB 分别为 A 和 B 的公开密钥。

由于非对称算法运算量比对称算法运算量大的多，如果对整个消息用私钥运算来做签名则显的效率太低，在真正的应用中采用的方式是：首先对消息使用某种消息摘要(Message Digest 或 Hash)算法进行处理，产生很短的消息摘要，再对消息摘要进行私钥加密作为数字签名，就大大提高了运算速度。接收者收到数据和签名后，用同样的算法算出数据的消息摘要，并用发送者的公钥对签名进行解密，最后把解密结果和上一步得到的消息摘要比较，就可以对签名进行校验。常用的消息摘要算法有 MD2、MD5、SHA1 等。数字签名过程如下：

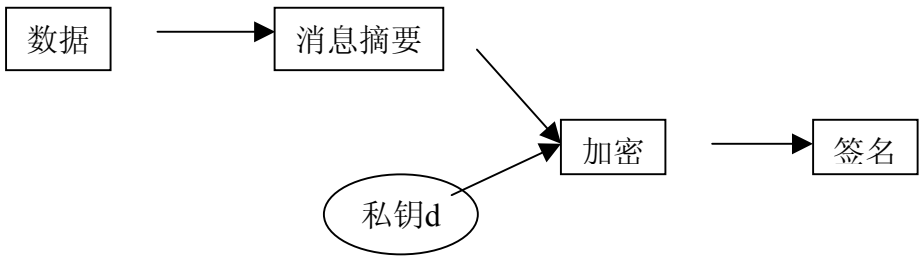

图 1－3　数字签名过程

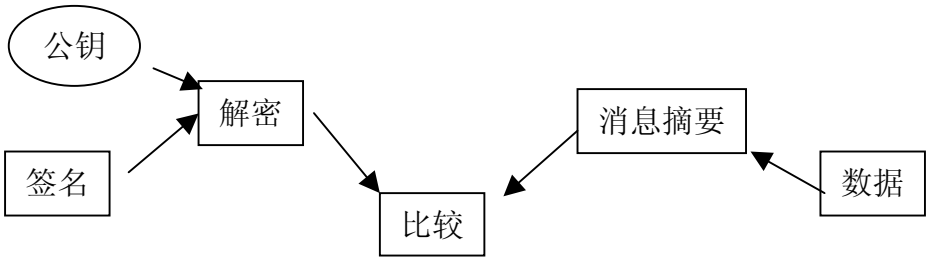

图 1－4　签名验证过程

1.2.3 数字证书及认证机构

数字证书是各类实体(持卡人/个人、商户/企业、网关/银行等)在网上进行信息交流及商务活动的身份证明，在电子交易的各个环节，交易的各方都需验证对方证书的有效性，从而解决相互间的信任问题。证书是一个经证书认证中心数字签名的包含公开密钥拥有者信息以及公开密钥的文件。从证书的用途来看，数字证书可分为签名证书和加密证书。签名证书主要用于对用户信息进行签名，以保证信息的不可否认性；加密证书主要用于对用户传送信息进行加密，以保证信息的真实性和完整性。

简单的说，数字证书是一段包含用户身份信息、用户公钥信息以及身份验证机构数字签名的数据。身份验证机构的数字签名可以确保证书信息的真实性。证书格式及证书内容遵循 X.509 标准。

数字证书认证中心（Certficate Authority,CA）是整个网上电子交易安全的关键环节。它主要负责产生、分配并管理所有参与网上交易的实体所需的身份认证数字证书。每一份数字证书都与上一级的数字签名证书相关联，最终通过安全链追溯到一个已知的并被广泛认为是安全、权威、足以信赖的机构-根认证中心（根 CA）。

1.2.4 PKI 和 PMI

PKI（Public Key Infrastructure）公钥基础设施是提供公钥加密和数字签名服务的系统或平台，目的是为了管理密钥和证书。一个机构通过采用 PKI 框架管理密钥和证书可以建立一个安全的网络环境。PKI 主要包括四个部分：X.509 格式的证书（X.509 V3）和证书废止列表 CRL（X.509 V2）；CA/RA 操作协议；CA 管理协议；CA 政策制定。一个典型、完整、有效的 PKI 应用系统至少应具有以下部分；

- 认证中心 CA
- X.500 目录服务器
- 具有高强度密码算法(SSL)的安全 WWW 服务器
- Web（安全通信平台）
- 自开发安全应用系统

完整的 PKI 包括认证政策的制定（包括遵循的技术标准、各 CA 之间的上下级或同级关系、安全策略、安全程度、服务对象、管理原则和框架等）、认证规则、运作制度的制定、所涉及的各方法律关系内容以及技术的实现。

PMI 是个生成、管理、存储及作废 X.509 属性证书的系统，PMI 实际上是 PKI 标准化过程中提出的一个新的概念，但是为了使 PKI 更迅速的发展，IETF 将它从 PKI 中分离出来单独制定标准。

与 PKI 不同 PMI 使用属性证书(AC attribute certificate)。属性证书将用户的一组属性和其它信息通过认证机构的私钥进行数字签名，使其不能伪造，用于证书的扩展使用。与公钥证书相关的某个实体可以同时拥有多个 AC。属性证书可以用于认证指定属性值，并且将这些属性值与 X.509 证书进行联编，或者直接与实体的名字进行联编。此外，属性证书的属性值可以实现组合匹配。AC 的上述特性使其能够用于构建基于角色的安全访问控制体系。

公钥证书与属性证书的格式定义于 X.509 标准中。

1.3 密码设备介绍

1．PCI 数据密码卡

数据密码卡是以 PCI 插卡的形式安装在用户计算机中的数据加/解密设备。它提供数据加/解密、数据完整性、数字签名、访问控制等功能，通过与应用程序的安全集成，在同一硬件平台上实现对各种安全应用的支持，为计算机信息系统提供安全保密服务，以防止来自内外的安全威胁和攻击，特别适合计算机和网络设备的数据加/解密。以下以卫士通数据密码卡为例进行介绍。

- 安全功能：数据加/解密；消息鉴别码（MAC）的产生/验证；支持单向散列；数字签名/数字验证；鉴别；访问控制。
- 技术特色：完善的系统保护措施；所有密码处理均在卡内由硬件实现，所有的密钥决不以明文的形式出现在卡外；口令保护；屏蔽罩保护，密钥、密码算法自毁；开发接口灵活，支持国际标准；
- 技术指标：

  实现一次一密；

接口标准：PCI 2.2 标准；

加密速率：低速 3M 比特/秒，高速 40-50M 比特/秒；

数字签名速度：低速 4 次/秒，高速 66 次/秒；

- 密码算法：

对称密码算法：专用密码算法、DES、3DES；

公开密钥算法：采用 RSA

散列算法：专用散列算法、MD5；

完整性算法（MAC）：专用 MAC 算法；

保护算法：专用保护算法；

- 开发接口：

提供标准 API 接口，支持多进程、多线程操作；

提供用户或系统集成商二次开发接口；

- 操作系统：

Windows 98、NT、2000、Sco Unix、Linux

平均无故障时间（MTBF）≥4000 小时；

- 产品特点：

PCI V2.2 标准接口；

高强度加/解密算法；

密码运算速度快；

完善的密码算法保护体制；

完善的密钥保护体制，密钥不以明文方式出现在密码卡外；

完善的密钥管理体制；

密钥通过智能 IC 卡注入；

随机密钥：采用物理噪声源，生成工作密钥，确保一次一密。

提供三层密钥管理体制；

提供标准 API，支持国际流行协议，产品互换性和升级性好；

便于用户作二次开发，可实现不同版本之间的平滑升级；

支持多进程、多线程操作；

即插即用（Plug and play），安装快速；

提供支持的各种操作系统下的驱动程序和测试程序；

外形小巧美观。

2．USB Key

USBKEY 系国家级项目，是以 USB 插卡的形式安装在用户计算机外 USB 口上的身份认证,数据加/解密设备.提供数据加/解密、数据完整性、数字签名、访问控制等功能，通过与应用程序的安全集成，在同一硬件平台上实现对各种安全应用的支持，对计算机信息提供安全保密措施，以防止来自内/外的安全威胁和攻击，特别适合网上身份认证和网络设备的数据加/解密。USB 小巧易用便携价格低廉。

3．金融数据密码机

SJL05 系列金融数据加密机是国内率先通过国家密码管 理办公室鉴定的基于金融业务主机的应用层数据加密机,主要用于数据加密、消息来源正确性验证、密钥 管理等,为计算机网络系统提供安全保密数据通信服务，防止网上的各种欺诈行为发生。

适用于各种类型的金融信息系统，尤其适用于对跨地区、 跨机构的金融交易系统提供数据加密机与安全保护，可广泛用于银行、证券、公交、社保、石化、城市一 卡通等计算机网络系统中。

1 密码体制

完整的安全保密体系结构

经国家主管部门审定批准的金融专用密码算法

完善的密钥管理系统

支持标准 DES、3DES

支持非对称密码算法

支持散列算法

2 兼容性

SJL05 金融数据加密机支持下列标准：

中国金融 集成电路（IC）卡规范 ANSI X.3.92 数据加密算法		
ANSI X.9.17 密钥管理	ANSI X.9.19 零售金融信息的鉴别	
物理锁防 拆设计	打开机盖 密钥自毁	人工毁钥
机外备份 密钥分段存放	访基于口 令认证的密钥管理,权限控制	

表 1－5 金融数据加密机支持标准

- 典型应用（有中心模式的应用）

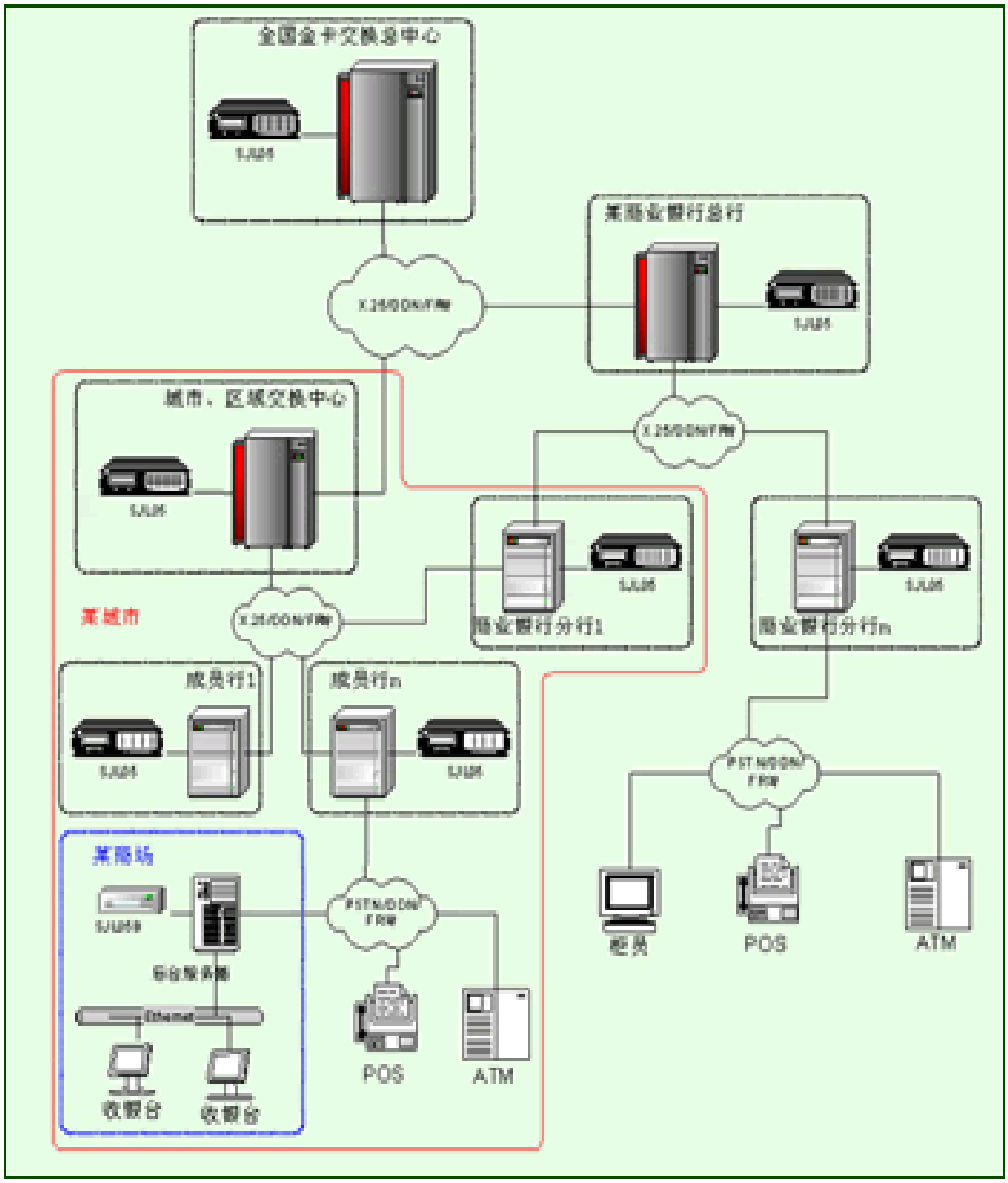

图 1－6 金融数据加密机典型应用模式

4．低端嵌入式平台

低端嵌入式平台以嵌入式操作系统为核心，提供包括数据加密、消息验证、密钥管理等低级密码服务的功能。

该平台主要应用于卫士通网络事业部的金融数据加密机作为其开发低速加密服务器市场。

第二章 基于 PKCS#11 协议安全平台的问题提出

2.1 课题研发背景

一般而言，安全一直仅占据着整个信息系统 10%的投资量。据 CCID 报道，国内安全市场 2002 年达到 20 亿人民币，这包括防火墙、加密软硬件模块、病毒软件、审计软件、系统恢复软件等。显然，加密软硬件仅仅是安全系统中很小的一部分。其中的深刻原因，就在于信息系统整体化水平不高，需求不高，使得信息系统的增值效应不明显，低水平建设多。当今国内的信息安全公司不下千家，但它们大部分都重复着同样的事情：生产加密设备，包括电话线上、DDN 上、帧中继、X.25、IP 等链路/路由级加密机，造成国家整体人力、物力资源的大量浪费。因此，公共、通用、易扩展的安全平台技术正受到国家相关部门的高度重视。作为电子政务、电子商务必须解决的问题之一，安全产品的互操作和可重用技术，将在未来 1－2 年内普及起来。而电子政务对安全的需求更加特殊，必须采用国内自主产权的产品。

目前安全技术中的高代价、易用性差、互操作能力弱等问题已经成为解决信息安全问题的瓶颈。

- 安全代价：安全的实现以牺牲系统的性能为前提。高速加密机是全世界所面临的一大难题，目前它的发展根本不能与网络本身基础设施的发展速度相提并论。
- 易用性：过去的加密设备没有考虑到它的商业需求，主要用于军事、政府等机密部门，有专门的人员操作、维护，易用性差。现在电子商务面向千家万

户，要每个人都有安全专家的水平是不可能的。

- 互操作：由于过去的加密设备覆盖范围小，一般一个部门只用一家的产品，产品各自为政，没有标准可循，不能互通互操作。例如：电子商务至少涉及用户、商户、开户行和收单行四者实体的复杂关系，但不可能让他们全部使用一家的产品。

2.2 国内外研究情况

为了有效的解决金融、保险、证券行业网络互联的安全保密问题，并且通过硬件实现对信息的加、解密等功能，各种加密机、加密卡等加密设备便应运而生。当前，国内许多科研单位都开发了高强度的加密算法，这些加密算法或者以纯软件的形式，或者以加密卡的形式提供给用户。然而，由于缺乏统一的技术标准，不同开发商的安全模块的体系结构和接口都不尽相同，这就给用户的使用造成了极大的不便，安全服务很难共用。因此，对提供加密互操作能力的安全平台的研究就显得尤为重要。事实上，对公共安全平台的研究最近几年来一直是方兴未艾。

国外对于安全产品的研究和开发已有较长的历史，企业/个人的安全意识相对较强，安全产品的应用较为广泛，除了传统的加密厂家还在从事这方面的工作外，象 IBM、Microsoft 这样的业界巨头已经在操作系统、中间件、电子政务/商务工具中引入安全增值服务，在更广的层面上普及安全事业。为此，国外许多厂家都提出了加密设备的互操作标准，如 RSA 公司的 PKCS#11 和 Microsoft 公司的相应竞争产品 CryptAPI 等。然而在国内，大部分安全企业都停留在加密设备的研制生产上，对公共安全平台的研究没有给予足够的重视。本课题的相关产品为 WPKCS#11（**以下提到的基于 PKCS#11 的安全平台均指的是 WPKCS#11 项目**）。下表展示了 WPKCS#11 与国外主要的 PKCS#11 实现产品的功能对比：

产品 / 功能	IBM Cryptoki	Encrypt PKCS#11 实现	TrustCenter GPKCS#11	WPKCS#11
生成密钥	√	√	√	√
数据加解密	√	√	√	√
数据摘要	√	√	√	√
签名验证	√	√	√	√
密钥导入导出	√	×	√	√
支持 Win32 平台	×	√	√	√
支持 Unix 平台	√	√	√	√
国内专有算法	×	×	×	√
支持多线程	√	√	×	√

表 2－1　WPKCS#11 与国外主要 PKCS#11 实现产品的功能对比

从上表可以看出， WPKCS#11 与主要的 PKCS#11 实现相比，支持的功能很全，并且支持国内自主研制的专有密码算法，充分体现了本课题的特色。

2.3 PKCS#11 协议解决的问题

公钥密码标准 PKCS（Public Key Cryptography Standards）是美国 RSA 数据安全公司为公钥密码学提供的一个工业标准接口。PKCS 包含一系列标准，其中的 PKCS#11 是加密设备接口标准，该标准详细规定了一个称为 Cryptoki（cryptographic Token interface 的缩写）的编程接口，它可以用于各种可移植的安全设备。Cryptoki 给出了一个通用的逻辑模型，用户不需要知道详细的技术细节就可以在可移植的设备上完成加密操作。Cryptoki（cryptographic Token interface）设计的目的是建立应用和加密设备（智能卡, PCMCIA 卡,加密卡，加密机，smart diskettes…… ）之间的接口。

PKCS#11 规范的提出主要有两个目的：一个目的是屏蔽各种安全实现之间的差异，向用户提供统一的接口，使安全服务的具体实现方法对用户透明化；另一个目的是使用户可以在多个安全服务具体实现之间共享资源，使一个设备可以被多个应用程序访问，一个应用程序也可以访问多个设备。PKCS#11 协议中的结构

描述正是以这两点为目标展开的。

2.4 基于 PKCS#11 协议安全平台拟解决的问题

- 基于 PKCS#11 协议安全平台的可行性论证。包括：对现有系统的分析、待研究系统的分析、投资及效益分析和社会条件方面的分析
- 协议安全逻辑的论证，提出更高级别的安全逻辑：敏感数据的“硬保护”、内存的安全访问、防后门程序设计
- PKCS#11 协议管理和调度层软件的设计实现
- PKCS#11 协议 Token 层软件的设计实现
- Token 开发工具包的设计实现
- 测试使用该平台后的性能和易用性
- 与第三方软件交互使用，测试该平台的通用性

2.5 本人完成的相关工作

参与完成本课题项目的系统设计和协议逻辑论证，完成代码约 35000 行（其中 C 程序 25000 余行，C++程序 10000 余行），文档约 50 页，RationalRose 作图数副，具体为：

- 安全平台协议层和令牌层软件的参与设计
- 多平台（Win32/Unix/Linux）设计
- 多线程模块设计实现，多进程模块设计
- 多卡（等速和非等速）阵列的调度设计
- 自定义的内存分配回收机制
- 实现对敏感信息的访问控制
- 完成金融加密机、加密卡、嵌入式平台 Token 的设计实现
- 完成 Token 开发包的实现

第三章 PKCS#11 协议的关键技术探讨

3.1 PKCS#11 系统设计模型

Cryptoki 的通用模型可以用下图来阐述。模型的最上层为一些需要执行密码操作的应用，最下层为执行具体运算操作的密码设备。

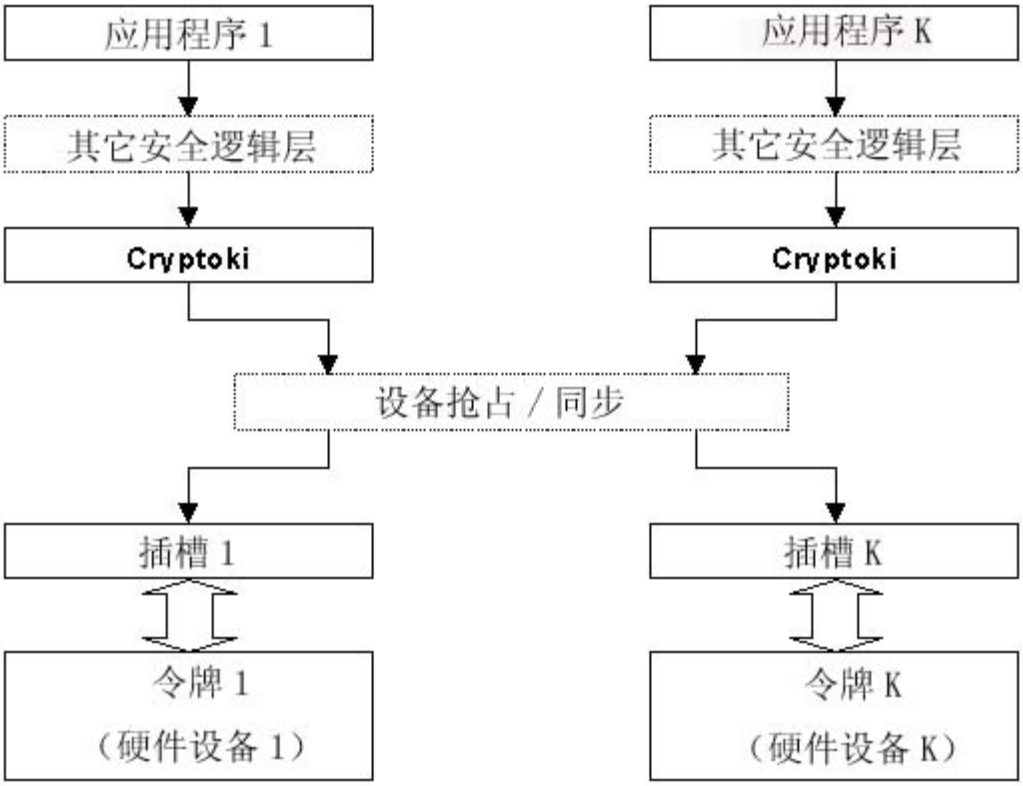

图 3－1 PKCS#11 通用设计模型

Cryptoki 通过一系列“slot”提供了对许多密码设备访问的接口，这些设备在 slot 中处于激活状态。每一个 slot 对应一个物理读卡器或其他设备接口，他们包括一个或多于一个的“Token”。Cryptoki 仅仅是提供了对 slot 和 Token 的一种逻辑视图，他们也可以代表其他相类似的物理设备，因此多个 slot 共享一个物理设备也是可行的。问题的关键是一个系统有许多 slot 和通过插在 slot 中的 Token 的进行连接的上层应用。

通常密码设备通过特定命令集执行一系列密码操作，典型是通过诸如 PCMCIA 卡服务驱动或 socket 服务。Cryptoki 使每个密码设备逻辑上看起与其他设备无异。因此应用不需要直接对设备驱动的接口，Cryptoki 把这些细节隐藏了。

实际上下面说的“设备”可以是纯软件的实现，没有任何硬的东西。

3.1.1 设定用户类型

Cryptoki 设定了两种类型的用户。一种是安全官角色(Security Office)，另一种类型是普通用户。只有普通用户在被授权认证后才可以访问 Token 中的对象。SO 的角色是：初始化一个 Token、设置修改普通用户的 PIN。

在通常情况下安全官和普通用户为一个人。

3.1.2 Application 和 Cryptoki 的作用关系

对 Cryptoki，一个应用程序包括一个单独的地址空间，本应用控制的线程都在其间执行。调用 C_Initialize 一个应用程序开始了一个”Cryptoki application”，其后本应用程序可以调用其他的 Cryptoki 函数。调用结束后，应用程序通过 Cryptoki 函数 C_Finalize 来中止 Cryptoki 应用。

3.1.2.1 Application 和 Process

一般来说一个应用程序包括一个单独的进程。

调用 C_Initialize 一个 UNIX 进程 P 成为一个 Cryptoki 应用，然后 fork()一个子进程 C。因为 P 和 C 有单独的地址空间，他们不是同一个应用程序的不同部分。因此如果 C 需要使用 Cryptoki 它需要再次调用 C_Initialize。而且如果 C 需要经由 Cryptoki 登录 Tokens，即使这个时候 P 已经登录，C 也必须做一次登录操作。

3.1.2.2 Application 和 Thread

一些应用程序会以多线程的方式访问 Cryptoki 函数库。Cryptoki 要求应用程序提供库需要的一些信息以保证对多线程的支持。特别的是，当一个应用程序通过调用 C_Initialize 初始化 Cryptoki 库时，他可以指定下面四种多线程模式：

1. 应用程序不会在多线程方式中使用 PKCS#11，这样库可以不使用同步机制。
2. 应用程序要在多线程方式中使用 PKCS#11，库必须使用本地操作系统的

同步机制来保证线程安全性。

3. 应用程序要在多线程方式中使用 PKCS#11，库使用应用程序提供的方法来保证线程安全性。
4. 应用程序要在多线程方式中使用 PKCS#11，要使用本地操作系统的同步机制或应用程序提供的方法来保证线程安全性。

在后两种情况中，应用程序要自己提供方法来实现同步机制。它需要提供四个方法来操作互斥对象（mutex objects），包括 CreateMutex、DestroyMutex、LockMutex、UnlockMutex。

PKCS#11 定义了四个函数指针来指向这四种方法，然后用这四中指针和一个 flags 来组成一个称为 CK_C_INITIALIZE_ARGS 的结构。调用 PKCS#11 的 C_Initialize 方法时，就是用一个指向这样的结构的指针作为参数。关于这几个函数和结构的具体情况参看 PKCS#11 文档。根据 flags 中的 CKF_OS_LOCKING_OK 位的值和四个函数指针的值可以指定应用程序的四种线程方式。

3.1.3 会话(Session)的建立和状态的变迁

应用程序在使用 Cryptoki 的时候，要先在 Token 上建立一个或多个会话（Session），然后才能访问 Token 上的 Object 和功能，会话就是应用程序和 Token 之间的逻辑连接。但是，仅仅建立了会话还不能执行 Token 提供的加密操作，而必须在之前登录为普通用户。在 Cryptoki 中定义了两种用户，一种是安全官员（Security Officer），一种是普通用户。只有普通用户可以访问 Private Object、执行 Token 提供的加密操作。安全官员则用来对 Token 进行初始化、设置普通用户的 PIN 码，同时它也可以操作 Public Object。普通用户也只有在安全官员设置完毕其 PIN 码后才能进行登录。可见，安全官员和普通用户可能是同一个人，也可能是不同的人。

会话可分为只读会话和读写会话两种类型。但是所谓的只读和读写都是针对 Token Object 而言，而不是 Session Object。这是因为两种会话都可以创建、读、写及销毁 Session Object 和读 Token Object，而只有使用读写会话的应用程序才能创建、修改和销毁 Token Object。

会话的状态决定了会话对对象的访问权限及在其上进行的加密操作。只读会话有两种状态，当没有用户登录的时候是只读公有会话状态，当有普通用户登录的时候是只读用户功能状态，没有只读安全官员功能状态存在。其状态变迁图如下图所示：

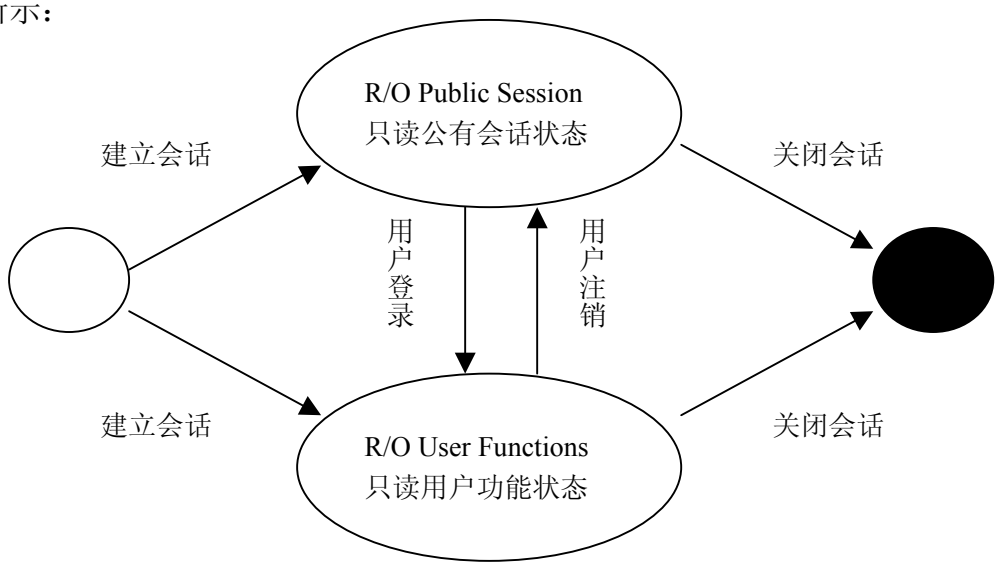

图 3－2　只读会话状态变迁图

读写会话有三种状态，分别是读写公有会话状态，读写用户功能状态，以及在安全官员登录后的读写安全官员功能状态。其状态变迁下图所示：

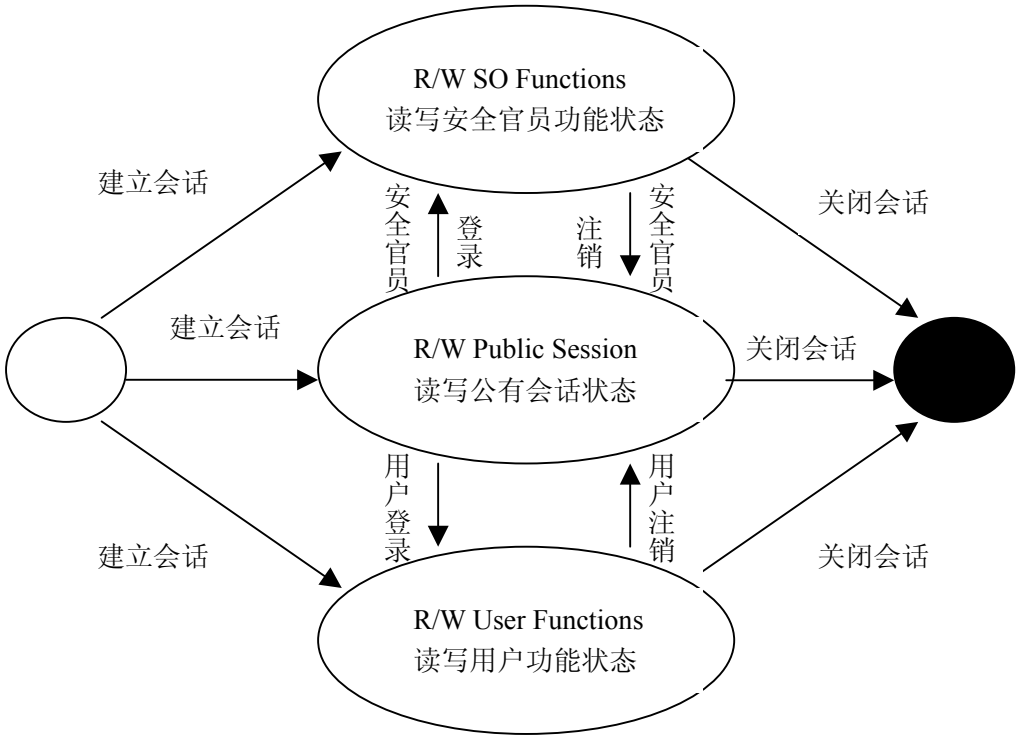

图 3－3　读写会话状态变迁图

Cryptoki 中一个应用程序可以在多个 Token 上打开多个会话，但是在同一个 Token 上的会话必须具有同样的状态。比如在一个 Token 上同时有只读公有会话状态和读写公有会话状态的会话，当有一个会话登录为普通用户，则所有的会话都将转变成用户功能状态。在同样的状况下，不能在某会话上登录为安全官员，即使是在读写公有会话状态的会话上也不能登录，这是因为应用程序已经打开了只读会话，而只读会话是没有相应的只读安全官员功能状态存在的。下表表示出不同状态的会话在不同的对象上的访问权限：

对象类型	会话类型				
	只读公有会话	读写公有会话	只读用户会话	读写用户会话	读写安全官员会话
公有会话对象	读写	读写	读写	读写	读写
私有会话对象			读写	读写	
公有 Token 对象	只读	读写	只读	读写	读写
私有 Token 对象			只读	读写	

表 3－4　会话对对象的访问权限

可以看出，某种会话对每种对象可能具有只读或读写的访问权限，也可能不具有任何访问权限。并且创建和删除一个对象需要读写权限，例如只读用户会话不能创建或删除 Token 对象。

3.1.4 抽象令牌(Token)

Cryptoki 提供了通过 Slot 访问加密设备的接口。每个 Slot 对应于系统中的物理读卡器或者是其它类似的物理接口。当有加密设备（如智能卡）被插入到读卡器中，Slot 中就出现对应的 Token。也就是说，Token 对应于如智能卡一类的加密设备。由于 Slot 和 Token 只是逻辑上的概念，因此他们不一定是对应于读卡器和智能卡，也可能对应于其它的加密设备，甚至是纯软件模块。系统中存在若干个 Slot，应用程序可以通过这些 Slot 来访问 Token 以完成安全任务。

加密设备通过特定的安全命令完成一定的安全任务，一般说来，这些命令是通过一些固定的设备驱动传向加密设备的。Cryptoki 的目的就是提供统一的接口，使

各种加密设备不管其具体的技术实现有何不同，在逻辑上看起来都一样。应用程序可以使用这种统一的接口来调用加密设备的功能，而不用直接面对加密设备的接口，甚至不用知道系统中有那些加密设备存在，Cryptoki 把这些具体问题都隐藏起来。Cryptoki 可以被实现为支持接口功能的库，应用程序链接到库上，从而调用 Cryptoki 的功能。

3.1.5 对象(object)体系结构

Cryptoki 中的 Token 用于存储 Object 和执行安全任务。Cryptoki 定义了三种 Object，数据对象、证书对象和密钥对象，数据对象由应用程序定义，证书对象用于存储各种证书，密钥对象用于存储各种密钥，包括私钥、公钥、对称密钥。Object 的层次结构下图所示：

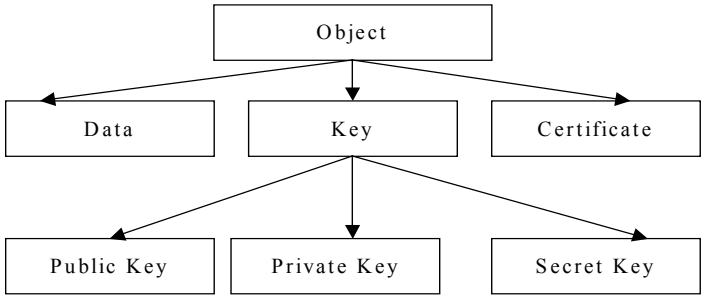

图 3－5　Object 的层次结构

3.1.5.1 分类

除了按 Object 所存储的内容分类外，也可以由其生命期和可用性分类，即分为 Token Object 和 Session Object 两类。Token Object 可以被连接到该 Token 上的有足够权限的会话使用，当该会话关闭后， Token Object 依然存在。而 Session Object 则要临时得多，当关闭会话时，该会话创建的所有 Object 都会被自动销毁，此外，一个应用程序创建的 Session Object 不能被其它的应用程序访问。

Object 也可以由访问权限来分类，即分为 Public Object 和 Private Object。应用程序不需要登录到 Token 就可以访问它的 Public Object。但是要访问 Private Object，应用程序必须先由 Token 验证身份，即在登录时由 Token 验证用户的 PIN 码。

3.1.5.2 属性

每一个对象都具有一系列的属性用于标识该对象的特性。例如，私有属性可用来标识该对象是否是私有对象，而类别属性则可用来标识该对象是数据、证书还是密钥。

对 Cryptoki 来说，常见的对象属性如下：

```
#define CKA_CLASS                 0x00000000
#define CKA_TOKEN                 0x00000001
#define CKA_PRIVATE               0x00000002
#define CKA_LABEL                 0x00000003
#define CKA_APPLICATION           0x00000010
#define CKA_VALUE                 0x00000011
#define CKA_OBJECT_ID             0x00000012
#define CKA_CERTIFICATE_TYPE      0x00000080
#define CKA_ISSUER                0x00000081
#define CKA_SERIAL_NUMBER         0x00000082
#define CKA_AC_ISSUER             0x00000083
#define CKA_OWNER                 0x00000084
#define CKA_ATTR_TYPES            0x00000085
#define CKA_TRUSTED               0x00000086
#define CKA_KEY_TYPE              0x00000100
#define CKA_SUBJECT               0x00000101
#define CKA_ID                    0x00000102
#define CKA_SENSITIVE             0x00000103
#define CKA_ENCRYPT               0x00000104
#define CKA_DECRYPT               0x00000105
#define CKA_WRAP                  0x00000106
#define CKA_UNWRAP                0x00000107
#define CKA_SIGN                  0x00000108
#define CKA_SIGN_RECOVER          0x00000109
#define CKA_VERIFY                0x0000010A
#define CKA_VERIFY_RECOVER        0x0000010B
#define CKA_DERIVE                0x0000010C
#define CKA_START_DATE            0x00000110
#define CKA_END_DATE              0x00000111
#define CKA_MODULUS               0x00000120
#define CKA_MODULUS_BITS          0x00000121
#define CKA_PUBLIC_EXPONENT       0x00000122
#define CKA_PRIVATE_EXPONENT      0x00000123
#define CKA_PRIME_1               0x00000124
#define CKA_PRIME_2               0x00000125
#define CKA_EXPONENT_1            0x00000126
```

```
#define CKA_EXPONENT_2              0x00000127
#define CKA_COEFFICIENT             0x00000128
#define CKA_PRIME                   0x00000130
#define CKA_SUBPRIME                0x00000131
#define CKA_BASE                    0x00000132
#define CKA_PRIME_BITS              0x00000133
#define CKA_SUB_PRIME_BITS          0x00000134
#define CKA_VALUE_BITS              0x00000160
#define CKA_VALUE_LEN               0x00000161
#define CKA_EXTRACTABLE             0x00000162
#define CKA_LOCAL                   0x00000163
#define CKA_NEVER_EXTRACTABLE    0x00000164
#define CKA_ALWAYS_SENSITIVE     0x00000165
#define CKA_KEY_GEN_MECHANISM    0x00000166
#define CKA_MODIFIABLE              0x00000170
/* CKA_ECDSA_PARAMS is deprecated in v2.11 */
#define CKA_ECDSA_PARAMS            0x00000180
#define CKA_EC_PARAMS               0x00000180
#define CKA_EC_POINT                0x00000181
#define CKA_SECONDARY_AUTH          0x00000200
#define CKA_AUTH_PIN_FLAGS          0x00000201
#define CKA_HW_FEATURE_TYPE         0x00000300
#define CKA_RESET_ON_INIT           0x00000301
#define CKA_HAS_RESET               0x00000302
#define CKA_VENDOR_DEFINED          0x80000000
```

3.1.6 安全扩展

作为加密设备的通用接口，Cryptoki 提供了单机和通讯系统中的基本安全服务。Cryptoki 两个要考虑的安全因素：

1. 访问 Token 的私有对象或其他安全类的操作函数可能需要经过 PIN 码的授权认证。因此单单拥有对某个密码设备的 Token 实现并不够，有必要还需要 PIN。

2. 其他体现在对私钥(private keys)和秘密密钥(secret keys)的保护上，本协议中我们将他们设为“sensitive” 或 “unextractable”属性。Sensitive 密钥意味着密钥数据不能以明文的形式暴露于 Token 之外，unextractable 密钥则意味着即便是加密保护了的也不能被暴露于 Token 之外。

基于这些特性我们应该使 Token 来提供对应用程序管理的对象的安全保护。

3.1.7 PKCS#11 协议实现接口

PKCS#11 规范定义了十三类函数接口，包括通用函数、Slot 和 Token 管理函数、Session 管理函数、Object 管理函数、加密函数、解密函数、消息摘要函数、签名和 MAC 函数、验证函数、双功能函数（即签名加密、解密验证等函数）、密钥管理函数、并行函数等。可见，这些函数接口大致分为三类：系统管理函数、对象管理函数和密码操作函数。其中系统管理函数主要实现 Cryptoki 的初始化、Session 的创建、用户的登录等功能。对象管理函数主要完成 Object 的创建、销毁、查找等功能。而密码操作函数就是指各种具体的密码算法操作了。

3.1.7.1 初始化相关

PKCS#11 定义了以下四个通用函数：

- C_Initialize：该函数用于对系统进行初始化。
- C_Finalize：该函数完成对系统的使用，进行相应的析构操作。
- C_GetInfo：该函数用于获得 Cryptoki 库的各种信息。
- C_GetFunctionList：该函数用于获得 Cryptoki 库函数的入口点。

3.1.7.2 Slot/Token 管理类

PKCS#11 定义了以下九个槽和令牌管理函数：

- C_GetSlotList：该函数用于获得系统中 slot 的列表。
- C_GetSlotInfo：该函数用于获得一个指定 slot 的信息。
- C_GetTokenInfo：该函数用于获得一个指定 Token 的信息。
- C_WaitForSlotEvent：该函数用于等待 slot 事件（如插入，删除 Token 等）的发生。
- C_GetMechanismList：该函数用于获得 Token 支持的 Mechanism 类型列表。
- C_GetMechanismInfo：该函数用于获得 Token 支持的某一 Mechanism 类型的信息。
- C_InitToken：该函数用于初始化一个 Token。
- C_InitPIN：该函数用于初始化普通用户的 PIN 码。
- C_SetPIN：该函数用于修改当前用户的 PIN 码。

3.1.7.3 会话(Session)管理类

PKCS#11 定义了以下八个会话管理函数：

- C_OpenSession：该函数用于在应用程序与 Token 之间建立一个会话。
- C_CloseSession：该函数用于关闭一个会话。
- C_CloseAllSessions：该函数用于关闭应用程序与 Token 之间建立的所有会话。
- C_GetSessionInfo：该函数用于获得某一会话的信息。
- C_GetOperationState：该函数用于获得某一会话的加密状态。
- C_SetOperationState：该函数用于设置某一会话的加密状态。
- C_Login：该函数用于登录一个用户。
- C_Logout：该函数用于注销一个用户。

3.1.7.4 对象管理

PKCS#11 定义了以下九个对象管理函数：

- C_CreateObject：该函数用于创建一个对象。
- C_CopyObject：该函数用于创建一个对象的拷贝。
- C_DestroyObject：该函数用于销毁一个对象。
- C_GetObjectSize：该函数用于获得一个对象的大小。
- C_GetAttributeValue：该函数用于获得一个对象的属性值。
- C_SetAttributeValue：该函数用于设置一个对象的属性值。
- C_FindObjectsInit：该函数用于初始化对象的查找操作。
- C_FindObjects：该函数用于查找对象。
- C_FindObjectsFinal：该函数用于结束对象的查找操作。

3.1.7.5 密码操作

PKCS#11 定义了以下二十九个密码功能函数，其中包括 4 个加密函数、4 个解密函数、5 个消息摘要函数、6 个签名函数、6 个验证函数及 4 个双功能函数：

- C_EncryptInit：该函数用于初始化加密操作。

- C_Encrypt：该函数用于加密单块数据。
- C_EncryptUpdate：该函数用于进行多块数据加密操作。
- C_EncryptFinal：该函数用于结束加密操作。
- C_DecryptInit：该函数用于初始化解密操作。
- C_Decrypt：该函数用于解密单块数据。
- C_DecryptUpdate：该函数用于进行多块数据解密操作。
- C_DecryptFinal：该函数用于结束解密操作。
- C_DigestInit：该函数用于初始化消息摘要操作。
- C_Digest：该函数用于摘要单块数据。
- C_DigestUpdate：该函数用于进行多块数据摘要操作。
- C_DigestFinal：该函数用于结束摘要操作。
- C_DigestKey：该函数用于对密钥进行摘要操作。
- C_SignInit：该函数用于初始化签名操作。
- C_Sign：该函数用于对单块数据进行签名。
- C_SignUpdate：该函数用于进行多块数据的摘要操作。
- C_SignFinal：该函数用于生成签名并结束签名操作。
- C_SignRecoverInit：该函数用于初始化签名操作，且数据能由签名恢复。
- C_SignRecover：该函数用于对数据签名，且数据能由签名恢复。
- C_VerifyInit：该函数用于初始化签名验证操作。
- C_Verify：该函数用于对基于单块数据的签名进行验证。
- C_VerifyUpdate：该函数用于对多块数据进行摘要操作。
- C_VerifyFinal：该函数用于完成基于多块数据的签名验证。
- C_VerifyRecoverInit：该函数用于初始化数据能从签名中恢复出来的签名验证操作。
- C_VerifyRecover：该函数用于进行数据能从签名中恢复出来的签名验证操作。
- C_DigestEncryptUpdate：该函数用于进行多块数据同时摘要及加密的操作。
- C_DecryptDigestUpdate：该函数用于进行多块数据同时解密及摘要的操作。
- C_SignEncryptUpdate：该函数用于进行多块数据同时签名及加密的操作。

- C_DecryptVerifyUpdate：该函数用于进行多块数据同时解密及签名验证的操作。

3.1.7.6 密钥管理

PKCS#11 定义了以下七个密钥管理函数：

- C_GenerateKey：该函数用于产生一个对称密钥。
- C_GenerateKeyPair：该函数用于产生一对公私钥。
- C_WrapKey：该函数用于导出一个密钥。
- C_UnwrapKey：该函数用于导入一个密钥。
- C_DeriveKey：该函数用于从基本密钥派生出一个密钥。
- C_SeedRandom：该函数用于将额外的种子加入随机数发生器。
- C_GenerateRandom：该函数用于产生随机或伪随机数据。

3.1.7.7 并行操作管理

PKCS#11 定义了以下两个并行操作管理函数：

- C_GetFunctionStatus：该函数用于获得某函数的执行状态。
- C_CancelFunction：该函数用于取消一个函数的执行。

3.2 总体体系结构

PKCS#11 的函数接口大致分为三类：administrative operations、object management operations、cryptographic operations。其中 administrative operations 主要是 Cryptoki 的初试化、Session 的创建、用户的登录等等功能。object management operations 主要是 Object 的创建、销毁、查找等等功能。cryptographic operations 就是各种具体的算法操作了。对于这三类函数，WPKCS#11 同样采用了两层的处理策略。在上层的 Cryptoki 库中主要完成协议本身要求的句柄映射、逻辑判断等等功能，而由实现所负责的具体算法实现或必须要下层验证的 PIN 码校验等工作就交给下层的 Token 库来完成。所以 Cryptoki 库的调用方法如下图所示：

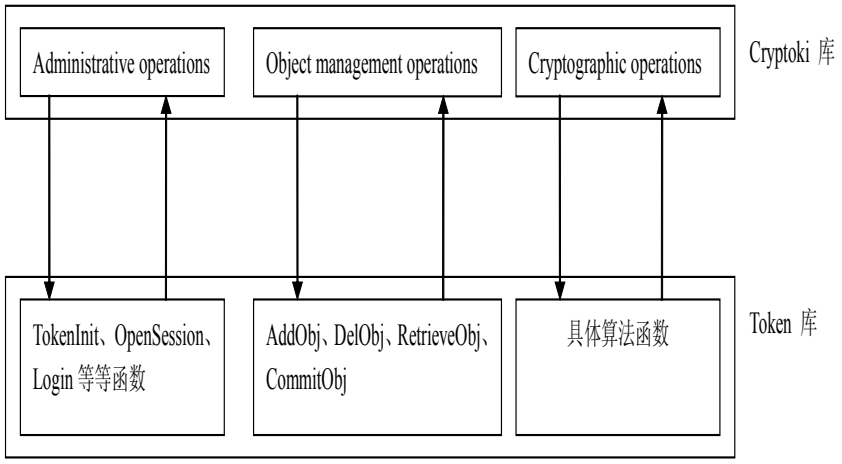

图 3－6　WPKCS#11 功能模块图

在调用 WPKCS#11 的功能时，如果是 Administrative Operations，主要是对 Cryptoki 库中的数据结构操作，这些操作大部分都是在 Cryptoki 库中完成，但需要调用 Token 库中的对应函数作相应的判断、认证等处理。如果是 Object Management Operations，主要对 Object 作操作，如果是 Token Object 则需要 Token 库作下层实现的支持，特别是需要利用 RetrieveObj 和 CommitObj 命令来完成各个进程之间的 Token Object 同步。如果是 Cryptographic Opeations，主要是完成各种算法，这时的主要操作都在 Token 库中完成，Cryptoki 库只是简单的判断参数的有效性，然后就把服务请求交给 Token 库。Crytoki 库是一静态库，Token 库是一动态库。

3.3 关键数据定义

3.3.1 插槽(slot)

slot 的数据结构定义如下：

```
struct ck_i_slot_data_st {
  CK_ULONG flags;
  CK_CHAR_PTR config_section_name;
  CK_SLOT_INFO_PTR slot_info;
  CK_I_TOKEN_DATA_PTR Token_data;
  CK_I_TOKEN_METHODS_PTR methods;
```

};

其中 flags 用于描述 Slot 的状态。Config_section_name 是配置文件中用于描述该 Slot 的段落的名称。Slot_info 是指向结构体 CK_SLOT_INFO 的指针。CK_SLOT_INFO 的结构如下所示：

```
struct CK_SLOT_INFO {
    CK_CHAR         slotDescription[64];
    CK_CHAR         manufacturerID[32];
    CK_FLAGS        flags;
    CK_VERSION      hardwareVersion;
    CK_VERSION      firmwareVersion;
};
```

可以看到其中的数据主要是 Slot 的文字描述、版本号和状态位。Token_data 是指向 ck_i_Token_data_st 数据结构的指针，表示当前 Slot 中装载的 Token，我们将在后面介绍 ck_i_Token_data_st 的数据结构。Methods 是指向 ck_i_Token_methods_st 数据结构的指针，ck_i_Token_methods_st 的结构如下所示：

```
typedef struct ck_i_Token_methods_st {
    CIP_GetTokenInfo GetTokenInfo;
    CIP_GetMechanismList GetMechanismList;
    CIP_GetMechanismInfo GetMechanismInfo;
    CIP_InitToken InitToken;
    CIP_FinalizeToken FinalizeToken;
    CIP_InitPIN InitPIN;
    CIP_SetPIN SetPIN;
    CIP_OpenSession OpenSession;
    CIP_CloseSession CloseSession;
    CIP_GetOperationState GetOperationState;
    CIP_SetOperationState SetOperationState;
    CIP_Login Login;
    CIP_Logout Logout;
```

CIP_EncryptInit EncryptInit;

CIP_Encrypt Encrypt;

CIP_EncryptUpdate EncryptUpdate;

CIP_EncryptFinal EncryptFinal;

CIP_DecryptInit DecryptInit;

CIP_Decrypt Decrypt;

CIP_DecryptUpdate DecryptUpdate;

CIP_DecryptFinal DecryptFinal;

CIP_DigestInit DigestInit;

CIP_Digest Digest;

CIP_DigestUpdate DigestUpdate;

CIP_DigestKey DigestKey;

CIP_DigestFinal DigestFinal;

CIP_SignInit SignInit;

CIP_Sign Sign;

CIP_SignUpdate SignUpdate;

CIP_SignFinal SignFinal;

CIP_SignRecoverInit SignRecoverInit;

CIP_SignRecover SignRecover;

CIP_VerifyInit VerifyInit;

CIP_Verify Verify;

CIP_VerifyUpdate VerifyUpdate;

CIP_VerifyFinal VerifyFinal;

CIP_VerifyRecoverInit VerifyRecoverInit;

CIP_VerifyRecover VerifyRecover;

CIP_DigestEncryptUpdate DigestEncryptUpdate;

CIP_DecryptDigestUpdate DecryptDigestUpdate;

CIP_SignEncryptUpdate SignEncryptUpdate;

CIP_DecryptVerifyUpdate DecryptVerifyUpdate;

CIP_GenerateKey GenerateKey;

```
CIP_GenerateKeyPair GenerateKeyPair;
CIP_WrapKey WrapKey;
CIP_UnwrapKey UnwrapKey;
CIP_DeriveKey DeriveKey;
CIP_SeedRandom SeedRandom;
CIP_GenerateRandom GenerateRandom;
CIP_GetFunctionStatus GetFunctionStatus;
CIP_CancelFunction CancelFunction;
CIP_WaitForSlotEvent WaitForSlotEvent
CIP_TokenObjAdd TokenObjAdd;
CIP_TokenObjCommit TokenObjCommit;
CIP_TokenObjDelete TokenObjDelete;
} CK_I_TOKEN_METHODS ;
```

虽然很长，然而类型却比较单一，全部都是函数指针，包括 Token 本身的管理函数，Pin 的管理函数，会话管理函数，登录函数以及大量的算法相关函数。

3.3.2 令牌(Token)

Token 的数据结构定义如下:

```
typedef struct ck_i_Token_data_st {
    CK_TOKEN_INFO_PTR Token_info;
    CK_SLOT_ID slot;
    CK_I_HASHTABLE_PTR object_list;
    CK_VOID_PTR impl_data;
    } CK_I_TOKEN_DATA;
```

其中 Token_info 是指向 CK_TOKEN_INFO 数据结构的指针。

```
typedef struct CK_TOKEN_INFO {
    CK_CHAR          label[32];
    CK_CHAR          manufacturerID[32];
    CK_CHAR          model[16];
```

```
CK_CHAR          serialNumber[16];
CK_FLAGS         flags;
CK_ULONG         ulMaxSessionCount;
CK_ULONG         ulSessionCount;
CK_ULONG         ulMaxRwSessionCount;
CK_ULONG         ulRwSessionCount;
CK_ULONG         ulMaxPinLen;
CK_ULONG         ulMinPinLen;
CK_ULONG         ulTotalPublicMemory;
CK_ULONG         ulFreePublicMemory;
CK_ULONG         ulTotalPrivateMemory;
CK_ULONG         ulFreePrivateMemory;
CK_VERSION       hardwareVersion;
CK_VERSION       firmwareVersion;
CK_CHAR          utcTime[16];
} CK_TOKEN_INFO;
```

很显然这是描述 Token 信息的数据结构，其中包括 Token 的名称，序列号标志位，各种会话的记数，pin 长度，存储空间，版本号等等。Slot 是整数类型的成员变量，用于保存自己所在的 Slot 的 ID 号。Object_list 是一个 hash 表，用于保存在这个 Token 上建立的 Object。GPKCS#11 中用 CK_I_HASHTABLE 数据结构来保存 hash 表。在这个数据结构中，保存有 hash 表长度和单元数量。在 hash 表单元中除了整数型的 key 和 index 外，使用 void 指针指向实际的对象。在 GPKCS#11 中在各种地方使用了大量的 CK_I_HASHTABLE 变量。Impl_data 是一个 void 指针，用该指针可以获取卫士通安全模块的句柄。

3.3.3 对象(Object)

3.3.3.1 对象数据结构

Object 的数据结构定义如下:

```
struct ck_i_obj_st {
```

```
CK_I_HASHTABLE_PTR table;
CK_ATTRIBUTE_PTR CK_PTR lookup;
CK_I_SESSION_DATA_PTR session;
CK_BBOOL changed;
CK_ULONG ref_count;
};
```

其中 lookup 指向描述对象属性的结构，这个结构比较简单，只有一个编号和一个整数，用于描述 Object 类型所拥有的属性。Session 是指向 CK_I_SESSION_DATA 数据结构的指针，表明拥有这个对象的 Session。Changed 表明这个对象是否发生了变化。Token Object 在各个进程间具有持久性，而 WPKCS#11 本身不是 deamon 程序，无法在各个进程间保持持久性。因此这个持久性是由加密设备来提供的。Token Object 和加密设备上的具有持久性的对象（如 IC 卡上的文件）是对应的，因此在发生变化后，应该及时的更新加密设备上的对象，以保证其它的进程能获得同一 Token Object 的变化。Ref_count 是这个对象的引用记数。由于一个对象可能被多个 session 使用，因此这个引用记数是必须的。

3.3.3.2 对象关键技术

为了处理方便，设计中把 object 的属性分为内部属性和外部属性，内部属性用头“CK_IA_”来表示。外部属性用头“CKA_”来表示。内部属性用来表示对象各属性的实际值，而外部属性主要用来表示对象实际拥有的属性，一般用在代码中。外部属性和内部属性是一一对应的，如：CKA_CLASS<－－>CK_IA_CLASS；CKA_TOKEN<－－>CK_IA_TOKEN。程序中可通外部属性取得某对象的内部属性值。

下面叙述内部属性和外部属性之间存在一个转换过程

首先，WPKCS#11 定义了一内部对象（即 WPKCS#11 支持的对象）数组，其定义如下：

```
CK_I_OBJECT_CLASS CK_I_obj_class_xlate[] = {
  CK_IO_DATA,
```

```
        CK_IO_CERTIFICATE,
        CK_IO_PUBLIC_KEY,
        CK_IO_PRIVATE_KEY,
        CK_IO_SECRET_KEY
};
```

CK_I_obj_class_xlate 数组是 CK_I_OBJECT_CLASS 型，CK_I_OBJECT_CLASS 的定义如下：

typedef CK_ULONG CK_I_OBJECT_CLASS

而数组的初始值是由如下宏定义：

```
#define CK_IO_DATA              0x00000001
#define CK_IO_CERTIFICATE    0x00000002
#define CK_IO_PUBLIC_KEY     0x00000004
#define CK_IO_PRIVATE_KEY    0x00000008
#define CK_IO_SECRET_KEY    0x00000010
```

其次，WPKCS#11 用宏定义对象的内部属性，其定义如下：

```
#define CK_IA_CLASS                 0
#define CK_IA_TOKEN                 1
#define CK_IA_PRIVATE               2
#define CK_IA_MODIFIABLE            3
#define CK_IA_LABEL                 4
#define CK_IA_APPLICATION           5
#define CK_IA_VALUE                 6
#define CK_IA_CERTIFICATE_TYPE      7
#define CK_IA_ISSUER                8
#define CK_IA_SERIAL_NUMBER         9
#define CK_IA_SUBJECT               10
#define CK_IA_KEY_TYPE              11
#define CK_IA_ID                    12
#define CK_IA_START_DATE            13
#define CK_IA_END_DATE              14
```

```
#define CK_IA_DERIVE                15
#define CK_IA_LOCAL                 16
#define CK_IA_ENCRYPT               17
#define CK_IA_VERIFY                18
#define CK_IA_VERIFY_RECOVER        19
#define CK_IA_WRAP                  20
#define CK_IA_MODULUS               21
#define CK_IA_MODULUS_BITS          22
#define CK_IA_PUBLIC_EXPONENT       23
#define CK_IA_PRIME                 24
#define CK_IA_SUBPRIME              25
#define CK_IA_BASE                  26
#define CK_IA_ECDSA_PARAMS          27
#define CK_IA_EC_POINT              28
#define CK_IA_SENSITIVE             29
#define CK_IA_DECRYPT               30
#define CK_IA_SIGN                  31
#define CK_IA_SIGN_RECOVER          32
#define CK_IA_UNWRAP                33
#define CK_IA_EXTRACTABLE           34
#define CK_IA_ALWAYS_SENSITIVE      35
#define CK_IA_NEVER_EXTRACTABLE     36
#define CK_IA_PRIVATE_EXPONENT      37
#define CK_IA_PRIME_1               38
#define CK_IA_PRIME_2               39
#define CK_IA_EXPONENT_1            40
#define CK_IA_EXPONENT_2            41
#define CK_IA_COEFFICIENT           42
#define CK_IA_VALUE_BITS            43
#define CK_IA_VALUE_LEN             44
```

```
#define CK_IA_SSL_VERSION              45
#define CK_IA_PERSISTENT_KEY           46
```

第三，WPKCS#11 定义了一个全局静态数组，该数组是一个二维数组，用来存放外部属性和数字序号之间的对应关系，实质就是外部属性和内部属性的对应转换。该数组定义如下：

```
static unsigned int CK_I_attrib_xlate[][2] = {
        {0, CKA_CLASS},
        {1, CKA_TOKEN},
        {2, CKA_PRIVATE},
        {3, CKA_MODIFIABLE},
        {4, CKA_LABEL},
        {5, CKA_APPLICATION},
        {6, CKA_VALUE},
        {7, CKA_CERTIFICATE_TYPE},
        {8, CKA_ISSUER},
        {9, CKA_SERIAL_NUMBER},
        {10, CKA_SUBJECT},
        {11, CKA_KEY_TYPE},
        {12, CKA_ID},
        {13, CKA_START_DATE},
        {14, CKA_END_DATE},
        {15, CKA_DERIVE},
        {16, CKA_LOCAL},
        {17, CKA_ENCRYPT},
        {18, CKA_VERIFY},
        {19, CKA_VERIFY_RECOVER},
        {20, CKA_WRAP},
        {21, CKA_MODULUS},
        {22, CKA_MODULUS_BITS},
        {23, CKA_PUBLIC_EXPONENT},
```

```
        {24, CKA_PRIME},
        {25, CKA_SUBPRIME},
        {26, CKA_BASE},
        {27, CKA_ECDSA_PARAMS},
        {28, CKA_EC_POINT},
        {29, CKA_SENSITIVE},
        {30, CKA_DECRYPT},
        {31, CKA_SIGN},
        {32, CKA_SIGN_RECOVER},
        {33, CKA_UNWRAP},
        {34, CKA_EXTRACTABLE},
        {35, CKA_ALWAYS_SENSITIVE},
        {36, CKA_NEVER_EXTRACTABLE},
        {37, CKA_PRIVATE_EXPONENT},
        {38, CKA_PRIME_1},
        {39, CKA_PRIME_2},
        {40, CKA_EXPONENT_1},
        {41, CKA_EXPONENT_2},
        {42, CKA_COEFFICIENT},
        {43, CKA_VALUE_BITS},
        {44, CKA_VALUE_LEN},
        {45, CKA_SSL_VERSION},
        {46, CKA_PERSISTENT_KEY}
};
```

第四，WPKCS#11 定义了一个 CK_BYTE 的数组，该数组中存放的内容用来表示外部属性和 WPKCS#11 定义的内部对象之间的对应关系，该数组下标表示 CK_I_attrib_xlate 全局静态数组中存放的数字序号（外部属性序号），数组元素的值表示外部属性所对应的对象，该数组定义如下：

```
CK_I_attributes[] = {
CK_IO_DATA|CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_K
```

```
EY|CK_IO_SECRET_KEY, /* CKA_CLASS */
CK_IO_DATA|CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_K
EY|CK_IO_SECRET_KEY, /* CKA_TOKEN */
CK_IO_DATA|CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_K
EY|CK_IO_SECRET_KEY, /* CKA_PRIVATE */
CK_IO_DATA|CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_K
EY|CK_IO_SECRET_KEY, /* CKA_MODIFIABLE */
CK_IO_DATA|CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_K
EY|CK_IO_SECRET_KEY, /* CKA_LABEL */
            /* data */
        CK_IO_DATA, /* CKA_APPLICATION */
CK_IO_DATA|CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_K
EY|CK_IO_SECRET_KEY, /* CKA_VALUE */
          /* certificate */
        CK_IO_CERTIFICATE, /* CKA_CERTIFICATE_TYPE */
        CK_IO_CERTIFICATE, /* CKA_ISSUER */
        CK_IO_CERTIFICATE, /* CKA_SERIAL_NUMBER */
        CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY,     /*
CKA_SUBJECT */
    /* keys, common */
        CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,     /*
CKA_ID */
        CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,     /*
CKA_START_DATE */
        CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,     /*
CKA_END_DATE */
        CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,     /*
CKA_DERIVE */
        CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,     /*
CKA_KEY_TYPE */
```

```
    CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,     /*
CKA_LOCAL */
    /* public keys */
    CK_IO_PUBLIC_KEY|CK_IO_SECRET_KEY, /* CKA_ENCRYPT */
    CK_IO_PUBLIC_KEY|CK_IO_SECRET_KEY, /* CKA_VERIFY */
    CK_IO_PUBLIC_KEY, /* CKA_VERIFY_RECOVER */
    CK_IO_PUBLIC_KEY|CK_IO_SECRET_KEY, /* CKA_WRAP */
    CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY, /* CKA_MODULUS */
    CK_IO_PUBLIC_KEY, /* CKA_MODULUS_BITS */
    CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY,                          /*
CKA_PUBLIC_EXPONENT */
    CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY, /* CKA_PRIME */
    CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY, /* CKA_SUBPRIME */
    CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY, /* CKA_BASE */
    CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY, /* CKA_ECDSA_PARAMS */
    CK_IO_PUBLIC_KEY, /* CKA_EC_POINT */
      /* private keys */
    CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY, /* CKA_SENSITIVE */
    CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY, /* CKA_DECRYPT */
    CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY, /* CKA_SIGN */
    CK_IO_PRIVATE_KEY, /* CKA_SIGN_RECOVER */
    CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY, /* CKA_UNWRAP */
    CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY, /* CKA_EXTRACTABLE */
    CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,                          /*
CKA_ALWAYS_SENSITIVE */
    CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY,                          /*
CKA_NEVER_EXTRACTABLE */
    CK_IO_PRIVATE_KEY, /* CKA_PRIVATE_EXPONENT */
    CK_IO_PRIVATE_KEY, /* CKA_PRIME_1 */
    CK_IO_PRIVATE_KEY, /* CKA_PRIME_2 */
```

CK_IO_PRIVATE_KEY, /* CKA_EXPONENT_1 */

CK_IO_PRIVATE_KEY, /* CKA_EXPONENT_2 */

CK_IO_PRIVATE_KEY, /* CKA_COEFFICIENT */

CK_IO_PRIVATE_KEY, /* CKA_VALUE_BITS */

CK_IO_SECRET_KEY, /* CKA_VALUE_LEN */

/* Vendor Defined */

CK_IO_SECRET_KEY, /* CKA_SSL_VERSION */

CK_IO_DATA|CK_IO_CERTIFICATE|CK_IO_PUBLIC_KEY|CK_IO_PRIVATE_KEY|CK_IO_SECRET_KEY /* CKA_PERSISTENT_KEY */

};

通过全局静态数组 CK_I_attrib_xlate[][2]可知该外部属性的数字序号，从而可求得内部属性；通过全局静态数组 CK_I_attributes[]可求得该属性属于哪些内部对象。具体的转换方式是通过 hash 表。

3.3.4 Session(会话)

session 的数据结构定义如下:

```
struct ck_i_session_data_st {
  CK_SESSION_HANDLE session_handle;
  CK_USER_TYPE user_type;
  CK_VOID_PTR pApplication;
  CK_I_APP_DATA app_data;
  CK_NOTIFY Notify;
  CK_SESSION_INFO_PTR session_info;
  CK_I_SLOT_DATA_PTR slot_data;
  CK_I_HASHTABLE_PTR object_list;
  CK_I_FIND_STATE_PTR find_state;
  CK_VOID_PTR digest_state;
  CK_MECHANISM_TYPE digest_mechanism;
  CK_VOID_PTR encrypt_state;
```

```
CK_MECHANISM_TYPE encrypt_mechanism;
CK_VOID_PTR decrypt_state;
CK_MECHANISM_TYPE decrypt_mechanism;
 CK_VOID_PTR sign_state;
CK_MECHANISM_TYPE sign_mechanism;
CK_VOID_PTR verify_state;
CK_MECHANISM_TYPE verify_mechanism;
  CK_VOID_PTR implement_data;
  CK_I_OBJ_PTR obj;
 };
```

在这个结构中各个 state 和 mechanism 成员变量，都是供各算法使用的内部数据。其中 state 变量是 void 指针，指向由算法定义的表示中间状态的结构。而 mechanism 变量是无符号长整型变量，是各种具体的算法和模式的编号，例如 DES_KEY_GEN 是 0x120， DES_ECB 是 0x121， DES_CBC 是 0x122。Session_handle 是长整型变量，用于表明 session 的句柄。User_type 表明现在运行该 session 的用户类型，普通用户或者是 SO。Papplication 和 app_data 都是自定义的数据结构，在 session 的回调函数中使用。Session_info 是指向 CK_SESSION_INFO 数据结构的指针，用于描述当前本 session 的状态和信息。CK_SESSION_INFO 的结构如下：

```
typedef struct CK_SESSION_INFO {
    CK_SLOT_ID      slotID;
    CK_STATE        state;
    CK_FLAGS        flags;
   CK_ULONG         ulDeviceError;
 } CK_SESSION_INFO;
```

其中 slotID 是当前 session 使用的 slot 的 ID 号。State 是当前 session 的状态，只读用户使用状态或读写公用会话状态等等。Flags 是标志位。UlDeviceError 是错误类型代号。Slot_data 是指向 CK_I_SLOT_DATA 结构的指针。使用这个成员变量 Session 把自己和自己使用的 Slot 连接了起来，间接地也连接到了 Token 上。

Object_list 是和 CK_I_TOKEN_DATA 中的 object_list 相同类型的 hash 表。其中所指向的是这个 Session 所使用的 object。Find_state 是 Session 在使用查找 object 功能时所需要的内部状态。Obj 是对象指针，该指针是把 Token 库中的 Token Object 移到 Cryptoki 库的中转站，这样便于内存的释放。

3.4 Cryptoki 库和 Token 库的内部结构关系

Cryptoki 库和 Token 库之间的内部机构如下图所示：

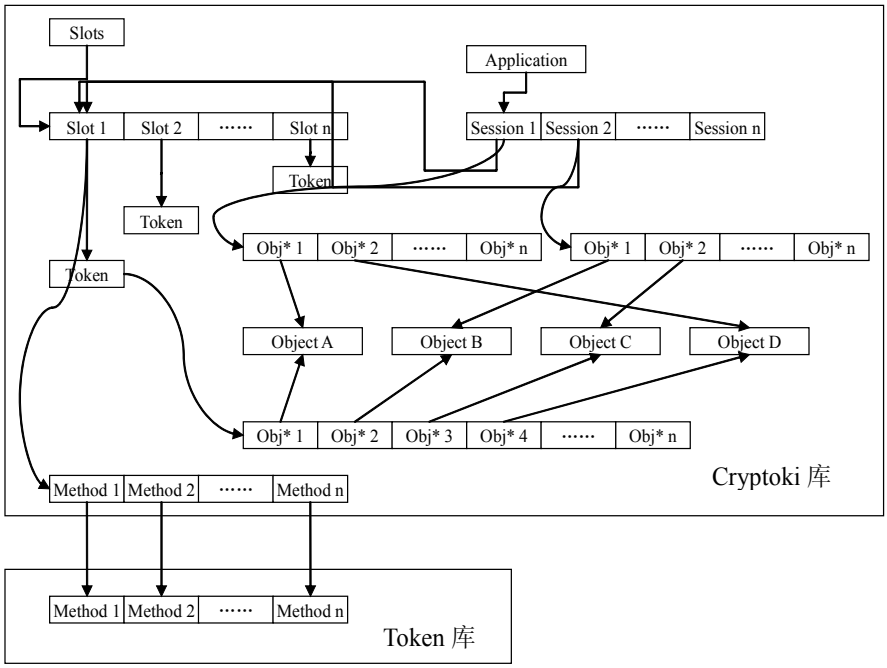

图 3－7 Cryptoki 和 Token 的内部结构

3.5 多线程同步机制实现

应用程序在使用 WPKCS#11 时，可以在调用 C_Initialize 函数时指定多线程环境的同步方式，共有四种方式，包括：

5. 应用程序不会在多线程方式中使用 WPKCS#11，这样 Cryptoki 库可以不使用同步机制。
6. 应用程序要在多线程方式中使用 WPKCS#11，Cryptoki 库必须使用本地操作系统的同步机制来保证线程安全性。
7. 应用程序要在多线程方式中使用 WPKCS#11，Cryptoki 库使用应用程序提供的方法来保证线程安全性。

8. 应用程序要在多线程方式中使用 WPKCS#11，要使用本地操作系统的同步机制或应用程序提供的方法来保证线程安全性。

在后两种情况中，应用程序要自己提供方法来实现同步机制。它需要提供四个方法来操作互斥对象（Mutex Object），包括 CreateMutex（创建互斥对象）、DestroyMutex（销毁互斥对象）、LockMutex（对互斥对象加锁）及 UnlockMutex（对互斥对象解锁）。

WPKCS#11 定义了四个函数指针来指向这四种方法，然后用这四种指针和一个 flags 标志来组成一个称为 CK_C_INITIALIZE_ARGS 的结构。调用 WPKCS#11 的 C_Initialize 函数时，就是用一个指向这样的结构的指针作为参数。根据 flags 中的相应标志位 CKF_OS_LOCKING_OK 的值和四个函数指针的值就可以指定应用程序的四种线程同步方式了，其对应关系如下：

1. CKF_OS_LOCKING_OK 不设置，四个函数指针为 NULL_PTR，对应线程方式 1。
2. CKF_OS_LOCKING_OK 设置，四个函数指针为 NULL_PTR，对应线程方式 2。
3. CKF_OS_LOCKING_OK 不设置，四个函数指针不为 NULL_PTR，对应线程方式 3。
4. CKF_OS_LOCKING_OK 设置，四个函数指针不为 NULL_PTR，对应线程方式 4。

这样，如果应用程序要使用自己的同步机制，就可以把同步机制的四个函数指针放在这样的结构中，递交给 C_Initialize。而 C_Initialize 就会用这四个函数来构建自己内部的同步机制。这样在 WPKCS#11 内部以后使用同步机制的时候，在实际中就会调用应用程序自己的同步机制。

WPKCS#11 中设置了一个数据结构 CK_I_EXT_FUNCTION_LIST，如下所示：

```
typedef struct CK_I_EXT_FUNCTION_LIST {
  CK_CREATEMUTEX   CreateMutex;
  CK_DESTROYMUTEX   DestroyMutex;
  CK_LOCKMUTEX   LockMutex;
  CK_UNLOCKMUTEX   UnlockMutex;
```

} CK_I_EXT_FUNCTION_LIST;

可见，这个结构中就包含了上述的四个函数指针变量。

同时，我们实现了 CreateMutex、DestroyMutex、LockMutex 和 UnlockMutex 四个内部函数，用以完成具体的同步机制：CK_RV CreateMutex (CK_VOID_PTR_PTR ppMutex) ;CK_RV DestroyMutex (CK_VOID_PTR pMutex) ;CK_RV LockMutex (CK_VOID_PTR pMutex) ;CK_RV UnlockMutex (CK_VOID_PTR pMutex);

下面是一个实际使用中的例子：

```
CK_VOID_PTR  Mutex = NULL_PTR;
CreateMutex ( & Mutex );
LockMutex ( Mutex );
/* 同步操作 */
UnlockMutex ( Mutex );
DestroyMutex ( Mutex );
```

3.6 可定制日志功能

为了记录 WPKCS#11 运行的全部过程，并反映当前其工作状况和进展，就必须使用日志文件来记载各函数的执行情况（包括函数执行成功与否的多种情况），因此日志文件设计的是否完善，对于用户理解、开发、使用和维护 WPKCS#11 有着重要的意义。

在 WPKCS#11 中，我们设置了两类日志文件：一类是一般操作日志，记录一般函数的执行情况，系统给存放该类型日志的文件取名为 PKCS#11.log；另一类是内存分配日志，记录 WPKCS#11 中 TC_malloc、TC_calloc、TC_free 等内存使用函数的分配内存和释放内存的情况，系统给存放该类型日志的文件取名为 PKCS#11.mem.log。

当调用 WPKCS#11 中 C_Initialize 函数初始化 Cryptoki 时，在 C_Initialize 中将调用 CI_LogEntry 函数完成日志文件的建立工作，并记载初始化进展信息。针对日志文件应该存放在哪个目录下、日志文件的名称是什么、用户能否自己设置日志文件的路径和名称等问题，WPKCS#11 都作了比较全面的考虑。在

WPKCS#11 中设置日志文件的存取路径和名称的方法有三种：

（1）在函数代码中设置日志文件的存取路径和名称。在函数 CI_EvalLogFileName 中定义了一个常量 CK_I_LOG_FILE，该常量用来存放日志文件的存取路径和名称，其宏代码如下：

```
#if defined(CK_GENERIC)
#define CK_I_LOG_FILE "/tmp/PKCS#11.log"
#elif defined(CK_Win32)
#define CK_I_LOG_FILE "c:\\PKCS#11.log"
#endif
```

（2）通过操作系统的环境变量 WPKCS#11_LOG、WPKCS#11_MEMLOG 设置日志文件的存取路径和名称，注意的是这两个变量不是操作系统本身的环境变量，需要相关操作系统设置。WPKCS#11 通过 getenv("WPKCS#11_LOG")、getenv("WPKCS#11_MEMLOG")函数指令来获取这两个变量的值。

在配置文件中设置日志文件的存取路径和名称。设置方法如下例的配置文件 config.txt 内容所示：

```
[PKCS#11-DLL]
LoggingFile=C:\\PKCS#11.log   //一般操作日志文件的设置
MemLoggingFile=C:\\PKCS#11.mem.log //内存使用日志文件的设置
TokenList = tryToken, Token1, Token2
[tryToken]
TokenDLL=D:\\config\\tryToken.dll
InitSym=Token_init   //InitSym 为 Token 的初始化函数名
[Token1]
  … //略
 [Token2]
  … //略
```

在 WPKCS#11 的实现中共包含两个库：一个是静态库 Cryptoki，一个是动态库 WToken。两个库中的内存分配和回收由于变量的频繁传递很容易发生混淆，

同时，在静态库中分配的内存并不能在动态库中释放，反之亦然。因此 WPKCS#11 定义了三个内存分配和回收函数，分别是 TC_malloc、TC_calloc 和 TC_free。事实上，这三个函数的实现是对 C 语言的三个库函数 malloc（）、calloc（）和 free（）进行的简单封装，特点是实现的三个函数要将内存分配和回收的信息写入日志文件，以便于跟踪对系统内存的使用情况。

3.7 异种设备支持

WPKCS#11 体系中的各个实体，即 Cryptoki 和各个 Token 都被作为独立的模块看待，互相之间处于松耦合的关系，模块之间的关系在运行时才能确定下来。为实现 WPKCS#11 对多种加密设备动态加载功能的支持，我们设计了相应的配置文件。从而使得 Cryptoki 库通过在运行时读取配置文件动态地加载所需的安全实现库来完成具体的安全服务。

在使用 WPKCS#11 之前，用户应使用文本编辑器编写配置文件 config.txt，在 config.txt 文件中指明各个 Token 动态链接库所在的目录，以便 WPKCS#11 在初始化时加载。以下为一个配置文件 Config.txt 的例子：

```
[PKCS#11-DLL]
  TokenList = tryToken, Token1, Token2   //待加载的 Token 名称列表，各 Token
                                          名称之间须用逗号分隔
  [tryToken]
  TokenDLL=D:\\config\\tryToken.dll   //Token 动态链接库 tryToken 所在的目录
  InitSym=SMToken_init   //SMToken_init 为 Token 库 tryToken 的初始化函数名
  [Token1]
    TokenDLL=D:\\config\\Token1.dll
     InitSym=SMToken_init
  [Token2]
     TokenDLL=D:\\config\\Token2.dll
     InitSym=OpensslToken_init
```

WPKCS#11 在初始化时将按照该配置文件的设置，把系统中的一系列 Token 动态链接库加载到内存中。加载过程为：

a) 打开配置文件 config.txt。

b) 初始化系统保存 Token 库信息的全局 hash 表 CK_I_dll_list。

c) 从配置文件中取出 Token 列表，取得当前第一个 Token 的名称、该 Token 对应的动态链接库的全路径名及其初始化函数的名称。

d) 完成该 Token 动态链接库的加载工作，并运行它的初始化函数。

e) 完成 Token 对应的 Slot 的注册工作，在 WPKCS#11 中每一个 slot 都唯一对应一个 Token。slot 通过设置其数据结构的 slotID 属性与该 Token 发生联系。从而就可以将当前 Slot 加入到 CK_I_slot_info_table 数组中，CK_I_slot_info_table 是系统保存 Slot 信息的指针数组。

f) slotID 值递增 1，重复 c)到 e)的步骤，直到将配置文件中描述的所有 Token 全部加载到内存中为止。

此后，应用程序就可以随意地调用各个安全设备来完成指定的安全操作了。

第四章 PKCS#11 协议层关键技术实现

4.1 PKCS#11 协议公共管理平台(Cryptoki)的软件实现

Cryptoki 在安全平台中处于核心的位置，它是 Cryptographic Token Interface（加密设备接口）的简写，出于效率的考虑我们将它实现为静态链接库，因而在本文中简称为 Cryptoki 库。Cryptoki 库用来实现 PKCS#11 标准中的各种逻辑模型，提供统一的算法接口供用户使用，是具体安全实现的上层接口部分，也是用户在开发安全应用系统时必须了解的部分，它随着 PKCS#11 标准的发展而发展，相对而言变化较小，但需要解决的逻辑问题却比较复杂。

值得注意的是，现有的大多数应用系统中，安全软件部分的实现往往与整个系统密不可分，二者保持着一种紧耦合的关系；另一方面，企业中真正掌握安全技术的人员往往有限，因此，企业自行为其应用系统开发相应的安全软件往往显得代价过高。因此，所设计和实现的 Cryptoki 库应该充分地考虑这些原因，使其具有可重用性，从而使安全软件部分从整个应用系统中分离出来成为通用的软件，使二者之间形成松耦合的关系。具有平台无关性的 Cryptoki 库能够在不改变

源代码的情况下在各种 Unix 环境和 Windows 平台上运行。

用户的请求由 Cryptoki 库来处理，Cryptoki 库中的接口部分将把请求分发到不同的处理模块，由相应的处理模块来完成基本的逻辑判断：用户的请求是否符合 PKCS#11 的标准，用户请求的前提条件是否已经满足（例如 Cryptoki 库是否已经被初始化）等，从而决定是否完成用户的请求。如果用户的请求符合要求，则 Cryptoki 库就会调用相应的模块来完成它，涉及到具体安全操作的请求还会被转交给对应的安全实现库来实现。安全实现库利用安全厂商提供的安全设备完成了用户的请求后会根据任务执行的情况把相应的结果返回到 Cryptoki 库，Cryptoki 库再把最终结果交给用户。图 4-1 展示了设计流程：

Cryptoki 在安全平台中处于核心的位置，它是 Cryptographic Token Interface（加密设备接口）的简写，出于效率的考虑我们将它实现为静态链接库，因而在本文中简称为 Cryptoki 库。Cryptoki 库用来实现 PKCS#11 标准中的各种逻辑模型，提供统一的算法接口供用户使用，是具体安全实现的上层接口部分，也是用户在开发安全应用系统时必须了解的部分，它随着 PKCS#11 标准的发展而发展，相对而言变化较小，但需要解决的逻辑问题却比较复杂。

值得注意的是，现有的大多数应用系统中，安全软件部分的实现往往与整个系统密不可分，二者保持着一种紧耦合的关系；另一方面，企业中真正掌握安全技术的人员往往有限，因此，企业自行为其应用系统开发相应的安全软件往往显得代价过高。因此，所设计和实现的 Cryptoki 库应该充分地考虑这些原因，使其具有可重用性，从而使安全软件部分从整个应用系统中分离出来成为通用的软件，使二者之间形成松耦合的关系。具有平台无关性的 Cryptoki 库能够在不改变源代码的情况下在各种 Unix 环境和 Windows 平台上运行。

用户的请求由 Cryptoki 库来处理，Cryptoki 库中的接口部分将把请求分发到不同的处理模块，由相应的处理模块来完成基本的逻辑判断：用户的请求是否符合 PKCS#11 的标准，用户请求的前提条件是否已经满足（例如 Cryptoki 库是否已经被初始化）等，从而决定是否完成用户的请求。如果用户的请求符合要求，则 Cryptoki 库就会调用相应的模块来完成它，涉及到具体安全操作的请求还会被转交给对应的安全实现库来实现。安全实现库利用安全厂商提供的安全设备完成了用户的请求后会根据任务执行的情况把相应的结果返回到 Cryptoki 库，

Cryptoki 库再把最终结果交给用户。图 4-1 展示了设计流程：

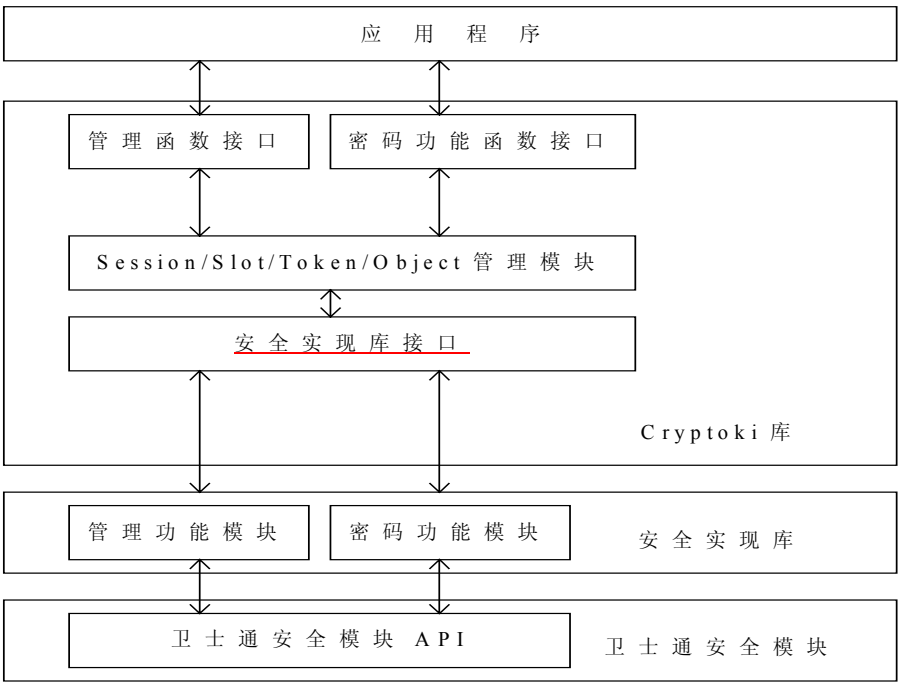

图 4－1 Cryptoki 库处理流程

基于 PKCS#11 的公共安全平台软件开发包可以直接为上层服务，既支持单机执行，也可作为服务器程序，下图为一个典型协作执行流程示例：

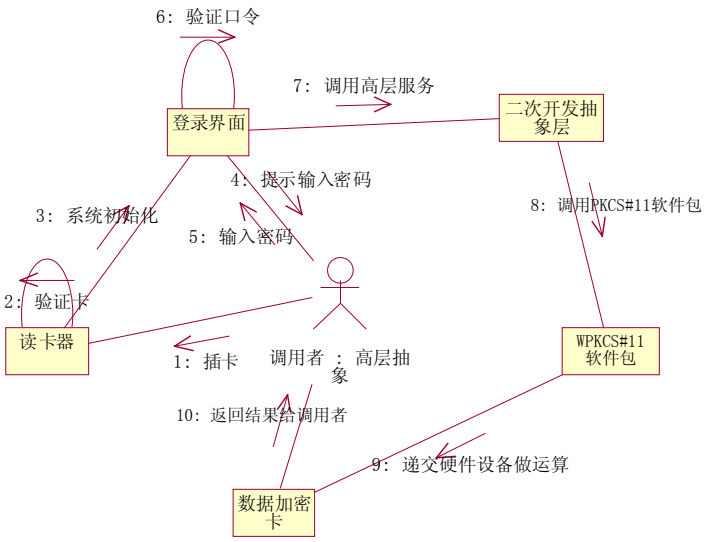

图 4－2 典型的协作流程图

4.2 初始化相关服务

应用程序在使用 WPKCS#11 时，必须首先调用 C_Initialize 函数初始化 Cryptoki。

函数原型：CK_RV C_Initialize(

CK_VOID_PTR pInitArgs

);

其中， pInitArgs 参数用于存放互斥机制模块操作函数结构的指针（CK_RV 为 PKCS#11 定义的函数返回值的数据类型）。如果应用程序要提供自己的互斥机制，可通过该参数传入。该参数的缺省值为 NULL_PTR，即使用系统提供的互斥机制。如果该函数返回值为 CKR_OK，表明初始化 WPKCS#11 成功，否则，调用失败。

其工作流程如下所述：

（初始化 Token）初始化所有的在配置表中定义的 Tokens，调用的方法是 CI_TokenInit。过程如下：

1. 设置初始的 slotID 为 0，每增加一个 Token 其自增 1，WPKCS#11 中采用一个 slot 对应一个 Token。
2. 查找配置文件，初始化动态库 hash 列表，获取配置文件的 Token 列表，解析 Token 名称，获取对应 Token 的动态库名称和动态库中定义的初始化自举函数的名称。
3. 根据自举函数名称获得这个函数的入口，调用它作一些初始设置，返回 slot 数据 slot_data。
4. 用 slotID 和 slot_data 注册之。
5. 重复上面过程直至所有的 Tokens 被加载，完成 CI_TokenInit。

（多线程）在调用 C_Initalize 函数要求给出输入，输入的数据类型为 CK_C_INITIALIZE_ARGS。同时定义了一个 CK_I_EXT_FUNCTION_LIST 类型的全局变量 CK_I_ext_functionsTR，定义互斥类型。如果输入为 NULL 则使用 WPKCS#11 内部的互斥实现，如果非 NULL 则使用自定义的互斥机制。如果是使用自定义的，首先要确保 iargs->pReserved 为 NULL，检查 iargs 中的四个关于互斥的指针确保非 NULL，再检查 iargs->flags 是否为 CKF_OS_LOCKING_OK。

如果是的话表示只能使用内部的互斥机制，否则继续选择自定义的互斥机制。无论是使用那种互斥方式，初始成功将 CK_I_global_flags 置为 CK_IGF_INITIALIZED。完成这些设置后即使用上述初始化的互斥方法创建 mutex、mutex_crypthandle、mutex_sessiontable 和 mutex_slotflag 互斥量，用于在 cryptoki 中实现同步机制。基本上我们都是采用 PKCS#11 内部的互斥实现。

（opensession 建立连接）方法是 C_OpenSession，主要输入有 slotID、flags、pApplication 和 phSession。

1. 检查是否被初始化（CK_I_global_flags 被置为 CK_IGF_INITIALIZED）和 flags 是否被置了 CKF_SERIAL_SESSION（可参看 PKCS#11v2.1section11.6）。

2. 根据 slotID 获取 slot_data，看 slot_data->slot_info->flags 是否被置了 CKF_TOKEN_PRESENT 位，判断 Token 是否存在。

3. 将 slot_data->Token_data 赋给本地变量 Token_data，检查当前的 session 数是否超出最大限

4. 检查 CK_I_app_table.session_table 是否初始化，没有的话就调用 CI_Inithashtable,hash 表单元数为 ulMaxSessionCount。

5. 如果是可读写 session 的话（CKF_RW_SESSION），检查 Token_data->Token_info->flags 是否被置为写保护状态，检查读写 session 是否达到最大限。成功后 session 标记 session_flags 置为 CKF_SERIAL_SESSION|CKF_RW_SESSION。检查 slot_data->flags 为 CK_I_SIF_RW_SO_SESSION、CK_I_SIF_USER_SESSION 和其它，则状态值分别置为 CKS_RW_SO_FUNCTIONS、CKS_RW_USER_FUNCTIONS 和 CKS_RW_PUBLIC_SESSION。

6. 如果是只读 session 的话，将 session_flags 置为 CKF_SERIAL_SESSION。检查 slot_data->flags 如果被置 CK_I_SIF_RW_SO_SESSION 位，表示已经打开了可读写 session，则不能创建只读 session，报错退出。如果被置 CK_I_SIF_USER_SESSION 位，将 state 置为 CKS_RO_USER_FUNCTIONS。

7. 将 slotID、session_flags 和 state 置给本 session 的 session_info。同时将 CK_I_app_table 置给 new_session->app_data，可以在 bensession 中对其它的 session 进行操作。

8. 分配新的 session 句柄，将 session 句柄连同 session_data 插到全局的 hash 表 CK_I_app_table.session_table 中去，完成打开一个新的 session 工作。

（登录 login）PKCS#11 规定了 5 中 session 状态，如下表所示：

前两种为未登录的 session 状态，后面的三种状态都是按照用户类型登录后的状态。R/O 和 R/W 的区别不是对所有的 object 的访问权限，而是对 Token object。P11 规定一个应用中，连接到同一个 Token 的 session 必须要具有相同的登录/注销状态，也就是说要么是 public 要么是 user 或者 sosession，请注意是相同的登录状态，而不是读/写状态。这就意味着如果一个应用的 session login/logout，则该应用的其它 session 也是自动 login/logout，举例有一个 R/O user session，则另一个 R/W session 自动的登录 user。同样如果已经已经有了一个 R/W SO session，则无法再打开一个 R/O session，因为没有 R/O SO session。

WPKCS#11 中的实现，检查是否初始化过，检查如果登录用户类型是 CKO_SO 且 session_data->session_info->state 状态是 user function，表示已经登录了用户则报错 CKR_USER_ANOTHER_ALREADY_LOGGED_IN。如果是任何一个 function 状态报错 CKR_USER_ALREADY_LOGGED_IN。再做其他的检查，参照上面 P11 的规定。检查所有的 session 发现有 R/O session 的话则无法用 SO 登录。调用 Token 函数 login。如果是 CKU_SO 且有效分别置 session_data->session_info->state 、 session_data->slot_data->flags 为 CKS_RW_SO_FUNCTIONS 和 CK_I_SIF_RW_SO_SESSION；如果是 CKO_USER 的话 session_data->slot_data->flags 置为 CK_I_SIF_USER_SESSION，根据 session_data->session_info->state 的值重新赋值：CKS_RW_PUBLIC_SESSION 则置为 CKS_RW_USER_FUNCTIONS， CKS_RO_PUBLIC_SESSION 置为 CKS_RO_USER_FUNCTIONS；调用 CI_PropagateSessionState 将本应用程序的其他 session 的状态更新为本 session 状态，结束登录。

与对 Cryptoki 的初始化相对应，当上层应用对 WPKCS#11 的使用结束时就应该调用 C_Finalize 函数来进行系统的结束处理。

4.3 Slot/Token 管理相关服务

根据对 WPKCS#11 的需求分析，我们设计和实现了下列针对 Slot 和 Token

的管理函数接口：

1. C_GetSlotList

函数原型：CK_RV C_GetSlotList

(

CK_BBOOL TokenPresent, // in 标识是否获取包含 Token 的 Slot 列表

CK_SLOT_ID_PTR pSlotList, // out slot 列表的首地址

CK_ULONG_PTR pulCount // in out 用于存放 slot 列表中 slot 个数

);

功能描述：获得系统中已加载的 Slot 列表。

使用说明：应用程序在使用该函数获得系统中的 Slot 列表时，一般应分为两个步骤，第一步令 pSlotList 为 NULL_PTR 调用该函数获得系统中的 slot 个数，此时的第三个参数 PulCount 作为输出参数；第二步调用该函数获得系统中的 slot 列表。

2．C_GetSlotInfo

函数原型：CK_RV C_GetSlotInfo

(

CK_SLOT_ID slotID， //in slot 的 ID 号

CK_SLOT_INFO_PTR pInfo， //out 用于存放指定 slot 的信息

);

功能描述：获取指定 Slot 的信息。

3．C_GetTokenInfo

函数原型：CK_RV C_GetTokenInfo

(

CK_SLOT_ID slotID, // in slot 的 ID 号

CK_TOKEN_INFO_PTR pInfo // out 用于存放指定 slot 上的 Token 信息

);

功能描述：获取指定 slot 上的 Token 的信息。

4．C_GetMechanismList

函数原型：CK_RV　C_GetMechanismList

(

CK_SLOT_ID　slotID, //in　slot 的 ID 号

CK_MECHANISM_TYPE_PTR　pMechanismList, //out 算法列表

CK_ULONG_PTR　pulCount //in out 用于存放算法列表中的算法个数

)；

功能描述：获得指定 Slot 支持的密码算法列表。

使用说明：应用程序在使用该函数获得某 Slot 支持的算法列表时，一般应分为两个步骤，第一步令 pMechanismList 为 NULL_PTR 调用该函数获得 Slot 支持的算法个数，此时的第三个参数 pulCount 作为输出参数；第二步调用该函数获得该 Slot 支持的算法列表。

5．C_GetMechanismInfo

函数原型：CK_RV　C_GetMechanismInfo

(

CK_SLOT_ID slotID,　//in　slot 的 ID 号

CK_MECHANISM_TYPE type,　//in　算法类型

CK_MECHANISM_INFO_PTR pInfo　//out　用于存放算法信息

)；

功能描述：获得某算法类型的信息。

6．C_InitToken

函数原型：CK_RV　C_InitToken

(

CK_SLOT_ID　slotID,　//in　slot 的 ID 号

CK_CHAR_PTR　pPin,　//in　SO 用户的 PIN 码

```
  CK_ULONG          ulPinLen,   //in   SO 用户 PIN 码长度
  CK_CHAR_PTR       pLabel      //in   Token 标识, 可置为 NULL_PTR
);
```

功能描述：初始化 Token。

7． C_InitPIN

```
函数原型：CK_RV   C_InitPIN
(
  CK_SESSION_HANDLE hSession,   //in  会话句柄
  CK_CHAR_PTR       pPin,       //in  用户 PIN 码
  CK_ULONG          ulPinLen    //in   PIN 码长度
);
```

功能描述：初始化用户 PIN 码。

使用说明: 该函数用来初始化连接到某 Token 上的普通用户的 PIN 码。该函数只能在 R/W SO Functions 的会话状态下调用（即只能由 SO 用户来使用）。

8. C_SetPIN

```
函数原型：CK_RV   C_SetPIN
(
    CK_SESSION_HANDLE hSession,   //in  会话句柄
    CK_CHAR_PTR       pOldPin,    //in  旧的用户 PIN 码
    CK_ULONG          ulOldLen,   //in  旧的用户 PIN 码长度
    CK_CHAR_PTR       pNewPin,    //in  新的用户 PIN 码
    CK_ULONG          ulNewLen    //in  新的用户 PIN 码长度
);
```

功能描述：用来修改当前登录用户的 PIN 码。

为实现以上这些函数接口，我们设计了如下的 Slot 数据结构：

```
struct   ck_i_slot_data_st {
```

```
  CK_ULONG    flags;
  CK_SLOT_INFO_PTR    slot_info;
  CK_I_TOKEN_DATA_PTR    Token_data;
  CK_I_TOKEN_METHODS_PTR    methods;
};
```

其中 flags 用于描述连接到该 Slot 的 Session 的用户登录状态。

Slot_info 是指向结构体 CK_SLOT_INFO 的指针，CK_SLOT_INFO 的结构如下所示：

```
struct CK_SLOT_INFO {
    CK_CHAR          slotDescription[64];
    CK_CHAR          manufacturerID[32];
    CK_FLAGS         flags;
    CK_VERSION       hardwareVersion;
    CK_VERSION       firmwareVersion;
};
```

可以看到其包含的信息主要是 Slot 的文字描述、版本号和状态位。

Token_data 是指向 ck_i_Token_data_st 数据结构的指针，表示当前 Slot 中装载的 Token，这样就将 Slot 与 Token 联系在了一起。 ck_i_Token_data_st 的数据结构定义如下：

```
struct ck_i_Token_data_st {
        CK_TOKEN_INFO_PTR Token_info;
        CK_SLOT_ID slot;
        CK_I_HASHTABLE_PTR object_list;
        CK_VOID_PTR impl_data;
    };
```

其中 Token_info 用来描述 Token 的各种信息，包括 Token 的名称，序列号，标志位，各种会话的记数，PIN 码长度，版本号等。会话计数是指对连接到该 Token 的会话的计数，这是因为加密设备允许同时打开的会话数目往往是有限的，应用程序在建立会话前首先要检查会话计数是否已达到最大值，若已经达到

最大值，则建立会话失败。

Slot 是整数类型的成员变量，用于保存该 Token 所处的 Slot 的 ID 号。

Object_list 是一个 hash 表，用于保存在这个 Token 上建立的 Object。

Impl_data 是一个 void 指针，它用来为 Token 保存自己需要的数据。

Slot 数据结构中的 Methods 域是指向 ck_i_Token_methods_st 数据结构的指针，ck_i_Token_methods_st 的结构如下所示：

```
typedef  struct  ck_i_Token_methods_st {
  CIP_GetTokenInfo GetTokenInfo;
  CIP_GetMechanismList GetMechanismList;
  CIP_GetMechanismInfo GetMechanismInfo;
  CIP_InitToken InitToken;
  CIP_FinalizeToken FinalizeToken;
  CIP_InitPIN InitPIN;
  CIP_SetPIN SetPIN;
  CIP_OpenSession OpenSession;
  CIP_CloseSession CloseSession;
  CIP_Login Login;
  CIP_Logout Logout;
  CIP_EncryptInit EncryptInit;
  CIP_Encrypt Encrypt;
  CIP_EncryptUpdate EncryptUpdate;
  CIP_EncryptFinal EncryptFinal;
  CIP_DecryptInit DecryptInit;
  CIP_Decrypt Decrypt;
  CIP_DecryptUpdate DecryptUpdate;
  CIP_DecryptFinal DecryptFinal;
  CIP_DigestInit DigestInit;
  CIP_Digest Digest;
  CIP_DigestUpdate DigestUpdate;
```

```
    CIP_DigestFinal DigestFinal;
    CIP_SignInit SignInit;
    CIP_Sign Sign;
    CIP_SignUpdate SignUpdate;
    CIP_SignFinal SignFinal;
    CIP_VerifyInit VerifyInit;
    CIP_Verify Verify;
    CIP_VerifyUpdate VerifyUpdate;
    CIP_VerifyFinal VerifyFinal;
    CIP_GenerateKey GenerateKey;
    CIP_GenerateKeyPair GenerateKeyPair;
    CIP_WrapKey WrapKey;
    CIP_UnwrapKey UnwrapKey;
    CIP_GenerateRandom GenerateRandom;
    CIP_TokenObjAdd TokenObjAdd;
    CIP_TokenObjCommit TokenObjRetrieve;
    CIP_TokenObjDelete TokenObjDelete;
} CK_I_TOKEN_METHODS ;
```

该结构包含的域很多，但它们的数据类型都是函数指针类型。这些函数指针都是指向 Token 库中库函数的指针，这也是 Cryptoki 库调用 Token 库方法的索引，包括 Token 本身的管理函数，PIN 码的管理函数，会话管理函数，登录函数以及大量的密码功能函数。Cryptoki 也即是通过该结构建立了与 Token 库的联系。

4.4 会话(Session)管理相关服务

一个典型的 WPKCS#11 上层应用程序往往通过下列步骤来执行各种安全任务：

1. 调用 C_Initialize 函数初始化 Cryptoki 库。
2. 选择一个 Token，调用 C_OpenSession 函数创建与该 Token 相关联的会话。

3．调用 C_Login 函数使用户登录到 Token 上。

4．使用创建好的会话来进行各种 Token 支持的安全操作。

5．调用 C_CloseSession 函数关闭会话。

6．调用 C_Finalize 函数完成对 Cryptoki 库的使用。

值得注意的是，一个应用程序可以同时和多个 Token 建立会话，一个 Token 也可以同时与多个应用程序建立连接。

按照 PKCS#11 规范，我们设计和实现了下列会话管理函数接口：

1．C_OpenSession

函数原型：CK_RV C_OpenSession

（

CK_SLOT_ID slotID,　// in　slot 的 ID 号

CK_FLAGS flags,　//in　session 的类型

CK_VOID_PTR pApplication,　　// in　回调函数参数

CK_NOTIFY Notify,　　// in　回调函数指针

CK_SESSION_HANDLE_PTR phSession //out 用于存放获得 session 句柄

）；

功能描述：用于在应用程序与指定加密设备之间建立会话。

通常，应用程序能同时与 Token 建立的会话数目是有限的，包括只读会话和读写会话。当会话由于数目的原因不能被创建时，函数将返回出错码 CKR_SESSION_COUNT。

应用程序只有在建立了会话之后才能获得各项安全服务，提供给用户使用的各种安全操作函数接口也都带有会话句柄这一参数，从而保证安全函数在会话成功建立的前提下执行。Session 的数据结构被设计成：

```
struct ck_i_session_data_st {
  CK_SESSION_HANDLE　session_handle;　//会话句柄
  CK_USER_TYPE　user_type;　// 用户类型
  CK_SESSION_INFO_PTR　session_info;　//会话相关信息
  CK_I_SLOT_DATA_PTR　slot_data;　//该会话连接的 Slot 的数据结构
  CK_I_HASHTABLE_PTR　object_list;　//该会话建立的对象列表
```

```
    CK_VOID_PTR    encrypt_state;   //加密状态信息
    CK_MECHANISM_TYPE    encrypt_mechanism;   //加密算法标识
    CK_VOID_PTR    sign_state;   //签名状态信息
    CK_MECHANISM_TYPE    sign_mechanism;   //签名算法标识
    ……     // 其他安全操作状态及算法信息（略）
};
```

在这个结构中，各个安全操作状态和算法标识成员变量都是供各安全操作使用的内部数据，安全操作状态成员变量是类属指针类型，它可被强制转换为安全操作中使用的表示中间状态的结构，即安全上下文。

2．C_CloseSession

函数原型：CK_RV C_CloseSession

```
(
    CK_SESSION_HANDLE hSession       //in    要关闭的会话的句柄
);
```

功能描述：用于关闭一个会话。

当一个会话被关闭后，所有由该会话创建的会话对象都将被自动删除，即使当前有其他会话正在使用这些会话对象。

3．C_CloseAllSessions

函数原型：CK_RV   C_CloseAllSessions

```
(
    CK_SLOT_ID slotID    //in    session 句柄
);
```

功能描述：用于关闭在某个加密设备（slot）上建立的所有会话。

4． C_GetSessionInfo

函数原型：CK_RV   C_GetSessionInfo

```
(
    CK_SESSION_HANDLE hSession,    //in    会话句柄
    CK_SESSION_INFO_PTR pInfo    // out    用于存放会话信息
);
```

功能描述：用于获取指定会话的信息。

5．C_Login

函数原型：CK_RV　C_Login

(

　　CK_SESSION_HANDLE hSession,　// in　会话句柄

　　CK_USER_TYPE　userType,　// in　用户类型

　　CK_CHAR_PTR　pPin,　// in 用户 PIN 码

　　CK_ULONG　ulPinLen　// in 以后 PIN 码长度

);

功能描述：用来登录用户。

6．C_Logout

函数原型：CK_RV　C_Logout

(

　　CK_SESSION_HANDLE　hSession　// in　会话句柄

);

功能描述：用来注销用户。

4.5 对象管理类服务

Object 在 Cryptoki 中占有相当重要的地位，也是 Cryptoki 中较为复杂的部分，本文将对它进行详细介绍。

PKCS#11 提出的 Session, Slot, Token 和 Object 等逻辑概念在 Cryptoki 中联系紧密，Session 和 Token 结构中都保存有 Object 列表。我们知道，PKCS#11 中的 Object 按其内容被分为三类：数据对象、证书对象和密钥对象。其中，数据对象由应用程序定义，证书对象用于存储各种证书，密钥对象用于存储各种密钥，包括公钥、私钥和对称密钥。事实上，大多数在 Token 上进行的安全操作都要用到 Object。例如，应用程序需要对一段数据进行非对称加密，就必须从某个 Token 中获得公钥对象来完成加密操作。Object 的特点在于它保存有其代表的对象的各种属性，以对称密钥对象为例，它就具有密钥类型、可导出性、密钥值等属性。需要指出的是，Token 也是 Object 的一种属性，它表明 Object 是否具有持久性，

具有这种属性的 Object 被称为 Token Object。所谓持久性是指对象不会因为应用程序结束会话而消亡，典型的 Token Object 是保存智能卡上公、私钥数据的相应密钥对象。与 Token Object 相对的是 Session Object，它不具有持久性，会随着创建它的会话的结束而消亡，典型的 Session Object 是对称密钥对象。

我们设计的 Object 数据结构为：

```
struct ck_i_obj_st {
  CK_ATTRIBUTE_PTR CK_PTR lookup;
  CK_I_SESSION_DATA_PTR session;
  CK_ULONG ref_count;
};
```

其中 lookup 指向描述对象属性的结构，用于描述 Object 所拥有的属性。每一个 Object 包含一个属性集，属性集中每一个属性都有一个特定的值。应用程序正是通过 lookup 指针访问 Object 的各种属性。

Session 是指向 CK_I_SESSION_DATA 数据结构的指针，表示拥有这个对象的会话。

Ref_count 是这个对象的引用记数。由于一个对象可能被多个 session 使用，因此这个用于引用记数的变量是必须的。

4.5.1 对象的属性

在WPKCS#11中，每一个Object都包含有一个属性集，属性集中每一个属性都应该对应有一个特定的值，不管是创建时上层接口赋予的还是系统默认值。为了有效地表示属性，我们将对象的属性定义为如下的结构体类型：

```
typedef   struct   CK_ATTRIBUTE {
          CK_ATTRIBUTE_TYPE     type;
          CK_VOID_PTR           pValue;
          CK_ULONG              ulValueLen;
} CK_ATTRIBUTE;
```

其中，type 表示属性的类型，pValue 指向属性值，ulValueLen 表示属性的长度(以字节为单位)。以 DES 密钥对象为例，它具有以下的相关属性：

属性	数据类型	含义
CKA_CLASS	CK_OBJECT_CLASS	对象类型
CKA_TOKEN	CK_BBOOL	如果对象为 token object，则值为 TRUE；否则缺省为 FALSE
CKA_PRIVATE	CK_BBOOL	如果对象为私有 object，则值为 TRUE;否则为 FALSE
CKA_MODIFIABLE	CK_BBOOL	如果对象能被修改则为 TRUE ，缺省值也为 TRUE
CKA_LABEL	字符串	对象的文字描述，缺省为空串
CKA_KEY_TYPE	CK_KEY_TYPE	密钥类型
CKA_ID	字节流	密钥的标识 ID，缺省为空
CKA_DERIVE	CK_BBOOL	如果其他密钥能从该密钥派生出，则为 TRUE；否则缺省为 FALSE
CKA_KEY_GEN_MECHANISM	CK_MECHANISM_TYPE	生成该密钥的算法标识
CKA_SENSITIVE	CK_BBOOL	如果密钥是敏感的则为 TRUE; 否则缺省为 FALSE
CKA_ENCRYPT	CK_BBOOL	如果该密钥可以被用来加密则为 TRUE
CKA_DECRYPT	CK_BBOOL	如果该密钥可以被用来解密则为 TRUE
CKA_WRAP	CK_BBOOL	如果该密钥可以被用来加密导出密钥则为 TRUE
CKA_UNWRAP	CK_BBOOL	如果该密钥可以被用来解密导入密钥则为 TRUE
CKA_EXTRACTABLE	CK_BBOOL	如果该密钥是可导出的则为 TRUE
CKA_ALWAYS_SENSITIVE	CK_BBOOL	如果该密钥的 CKA_SENSITIVE 属性一直为 TRUE 则为 TRUE
CKA_NEVER_EXTRACTABLE	CK_BBOOL	如果该密钥的 CKA_EXTRACTABLE 属性从不为 TRUE 则为 TRUE
CKA_VALUE	字节流	密钥的值

为获取和设置一个对象的属性值，我们实现了两个对象属性操作函数：

1．C_GetAttributeValue

函数原型：CK_RV C_GetAttributeValue

(

CK_SESSION_HANDLE hSession,　// in　session 句柄

CK_OBJECT_HANDLE hObject,　// in　对象句柄

CK_ATTRIBUTE_PTR pTemplate,　// in　out　属性值

CK_ULONG ulCount　// in　获取的属性个数

);

功能描述：获取一个对象的指定属性的值。

2．C_SetAttributeValue

函数原型：CK_RV C_SetAttributeValue

(

CK_SESSION_HANDLE hSession,　// in　session 句柄

CK_OBJECT_HANDLE hObject,　// in　对象句柄

CK_ATTRIBUTE_PTR pTemplate, // in　属性值

CK_ULONG ulCount　// in　设置的属性个数

);

功能描述：设置一个对象的指定属性的值

范例：设置对象 hData 的 CKA_VALUE 属性值为“New value”

```
CK_RV rv;
CK_BYTE NewValue[] = "New value";
CK_ATTRIBUTE attr_new_template[] = {
                    {CKA_VALUE, &NewValue, sizeof(NewValue)},
                };
rv = C_SetAttributeValue(hSession, hData, attr_new_template, 1);
//其中 hSession 由 C_OpenSession 函数获得；hData 为对象句柄
```

4.5.2 对象的创建、拷贝与删除

要创建一个对象可以使用 C_CreateObject 函数，而 C_GenerateKey, C_GenerateKeyPair, C_UnwrapKey 和 C_DeriveKey 函数也都事实上创建了一个对象，不过它们创建的是密钥对象，而 C_CreateObject 则可以创建任意类型的对象。当创建对象的时候，我们需要提供被创建对象的属性模板，也即属性组。

对象的拷贝构造是通过 C_CopyObject 函数实现，与常规意义的拷贝不同的是，该函数也可以在拷贝对象的同时修改对象的某些属性。需要注意的是，根据对象的访问控制规则，在拷贝秘密密钥的过程中，密钥的 CKA_EXTRACTABLE 属性只能从 TRUE 改为 FALSE，而不能由 FALSE 改为 TRUE。 与此类似，CKA_SENSITIVE 属性只能从 FALSE 改为 TRUE，而不能由 TRUE 改为 FALSE。此外，拷贝时可以将密钥的 CKA_TOKEN 属性和 CKA_PRIVATE 属性设为 TRUE，这种情况发生在由一个 session object 复制出一个 Token object 的情况下。而当应用程序提供的对象属性模板中有与原始对象属性不兼容的情况时，对象的拷贝将失败。

对象的删除过程则相对简单，其接口为：

```
函数原型：CK_RV C_DestroyObject
(
    CK_SESSION_HANDLE hSession,    // in  会话句柄
    CK_OBJECT_HANDLE hObject,      // in   要删除的对象的句柄
);
```

4.5.3 对象的查找

当我们想知道目前系统中有多少个 DES 密钥对象时，对象的查找函数接口就派上用场了，事实上它可以按用户指定的属性匹配要求来查找任意类型的对象。对象的查找过程是典型的三段式，即一次对象的查找过程由不可分隔的三个步骤组成：

1．对象查找的初始化

2．按指定的查找要求进行对象的查找工作

3．结束对象的查找

需要注意的是，在一个会话中，每次只能有一个查找操作在进行。为此，我们在 session 的数据结构中设有 find_state 结构成员，用于表示当前 session 的查找状态。

查找对象前，首先调用 C_FindObjectsInit 函数初始化 session 中的成员变量 Find_state，使之处于活动状态，若发现 find_state 状态已被激活则报错返回。函数 C_FindObjectsInit 原型如下：

```
CK_RV C_FindObjectsInit(
        CK_SESSION_HANDLE    hSession, // session 句柄
        CK_ATTRIBUTE_PTR    pTemplate, //  待查对象的匹配属性
        CK_ULONG          ulCount     //待查对象的匹配属性个数
      );
```

在初始化过程中，我们将创建与 Token 结构关联的 Object 列表一一对应的互斥对象，这是因为应用程序创建的每一个对象都将保存在 Token 结构的 Object 列表上，所以在查找对象时显然要使用这个全局 hash 表，而它就是临界资源，需要在多线程调用环境下互斥访问。此后，将调用 C_FindObjects 函数来进行具体的对象查找工作，该函数的原型如下：

```
CK_RV C_FindObjects(
    CK_SESSION_HANDLE       hSession,    //会话句柄
    CK_OBJECT_HANDLE_PTR    phObject, //  查找到的对象句柄
    CK_ULONG                ulMaxObjectCount,    //要查找的对象的最多个数
    CK_ULONG_PTR            pulObjectCount    //  查找到的对象的实际个数
  );
```

查找对象的过程就是一个与 Object 列表中的每个对象依次比较、匹配的过程，每一次查找到的对象的个数>=0。而对象的查找操作只能找到当前会话有权限查看的对象，例如，读写公有会话中的查找操作就不能查找到任何私有对象。

完成查找过程后，还必须做一些收尾工作。上层应用必须调用函数 C_FindObjectsFinal 来结束对象的查找，其函数原型如下：

```
CK_RV C_FindObjectsFinal(
        CK_SESSION_HANDLE hSession    //会话句柄
```

);

4.6 密钥处理服务

- C_GenerateKey：该函数用于产生一个对称密钥。
- C_GenerateKeyPair：该函数用于产生一对公私钥。
- C_WrapKey：该函数用于导出一个密钥。
- C_UnwrapKey：该函数用于导入一个密钥。

C_GenerateRandom：该函数用于产生随机或伪随机数据。

4.7 密码运算服务

Cryptoki 将提供密码功能接口，而具体的密码功能将由指定的 Token 来实现，在第五章中我们将详细的介绍密码功能的具体实现模块。目前，WPKCS#11 已提供以下的密码功能接口：

- C_EncryptInit：该函数用于初始化加密操作。
- C_Encrypt：该函数用于加密单块数据。
- C_EncryptUpdate：该函数用于进行多块数据加密操作。
- C_EncryptFinal：该函数用于结束加密操作。
- C_DecryptInit：该函数用于初始化解密操作。
- C_Decrypt：该函数用于解密单块数据。
- C_DecryptUpdate：该函数用于进行多块数据解密操作。
- C_DecryptFinal：该函数用于结束解密操作。
- C_DigestInit：该函数用于初始化消息摘要操作。
- C_Digest：该函数用于摘要单块数据。
- C_DigestUpdate：该函数用于进行多块数据摘要操作。
- C_DigestFinal：该函数用于结束摘要操作。
- C_SignInit：该函数用于初始化签名操作。
- C_Sign：该函数用于对单块数据进行签名。
- C_SignUpdate：该函数用于进行多块数据的摘要操作。
- C_SignFinal：该函数用于生成签名并结束签名操作。

- C_VerifyInit：该函数用于初始化签名验证操作。
- C_Verify：该函数用于对基于单块数据的签名进行验证。
- C_VerifyUpdate：该函数用于对多块数据进行摘要操作。
- C_VerifyFinal：该函数用于完成基于多块数据的签名验证。

第五章 PKCS#11 令牌(Token)适配层关键技术实现

5.1 令牌(Token)的设计

在总体结构中处于下层位置的是安全实现库，它用来实现具体的安全服务，并封装各种安全实现成为 Cryptoki 库可以调用的形式。

WToken 库基于卫士通 PCI 安全模块实现，提供各种各样的算法服务，不仅算法种类不同，有加密、解密、签名、验证、生成随机数、生成密钥等等，即使同一类算法也有若干种可以选择，例如对称加密算法就有 DES、3DES、商密 33 算法等多种不同的算法，同一种算法又包含多种模式，例如 DES 算法就有 ECB、CBC 等模式。作为一个功能完善的 Token，WToken 提供尽可能多的算法和算法模式，而且我们还考虑到了在今后为 WToken 添加其它算法的可能性。

PKCS#11 标准指出，安全实现库可以实现为静态库或动态库。显然，若实现为静态库，对于系统运行效率是有利的，但若把安全实现库实现为动态链接库的形式，就能使得 Cryptoki 库可以在运行时动态地加载所需的实现库来完成具体的安全服务，系统通过在运行时读取配置文件来达到这一目的。然而，选择动态库也有其弊端，这是因为若安全实现库被恶意替换，则 Cryptoki 库的用户 PIN 码等重要数据就可能被截取，因此应该引入模块间可信性认证机制。在 Cryptoki 库加载安全实现库之前，将采用 X.509 规范的三次握手来完成整个强认证过程。X.509 规范描述了如何在 PKI 体系中利用非对称算法，使用私钥和证书在两个实体之间作双方的身份认证。

安全实现库对于用户而言是透明的，用户在使用安全平台时根本不需要了解实现库的具体细节，但却可以通过上层的接口 Cryptoki 库来获取下层实现部分所支持的功能。安全实现库可以由安全厂商来开发，实现方法可以不同，但必须解决与 Cryptoki 库和加密安全模块的接口问题。

尽管在 PKCS#11 中，Slot 和 Token 分别对应于读卡器和智能卡，但它们只

是逻辑上的概念，因此在实现中也可以对应于其它的加密设备，甚至是纯软件模块。由于 Slot 与 Token 密不可分，每一个 Slot 中装载一个 Token，因而可将 Slot 和 Token 都实现到一个安全实现库中。

安全实现库除了实现具体的安全服务外，一个重要的功能是对用户登录进行 PIN 码校验，以便控制用户对安全实现库的访问。用户建立了会话后，还必须登录成功才能获得安全模块提供的各种安全服务。

WToken 安全实现库提供基于卫士通 PCI 安全模块的安全实现供上层的 Cryptoki 库调用，其在 WPKCS#11 体系中的位置如图 5-1 所示：

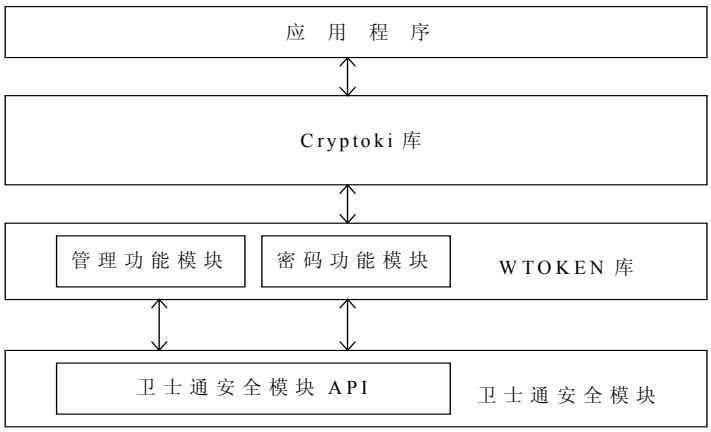

图 5－1　WToken 安全实现库在 WPKCS#11 体系中的位置

WToken 库将卫士通安全模块所提供的主要功能按照 Cryptoki 接口规范提供给 Cryptoki 库调用，这些功能包括：

- 提供安全通道 Channel，Channel 相当于 PKCS#11 中的 Session，即用户在使用其功能前要先建立一个 Channel，然后在使用其功能的过程中，把运算的中间结果、密钥等安全上下文放置在 Channel 的数据结构中。
- 提供使用 PIN 码登录的过程。
- 提供存储和查找非对称密钥及证书的功能。
- 提供生成对称密钥和非对称密钥对的功能。
- 提供将密钥加密导出及解密导入的功能；
- 对用户的数据作对称/非对称算法的加解密；
- 对用户的数据作 RSA 算法的签名和验证。

Slot Data 是 WToken 库需要使用的主要数据结构之一，它用来描述关于

WToken 的属性和函数方法，由 WToken 和 Cryptoki 共同维护。其数据结构定义如下：

```
struct    ck_i_slot_data_st {
   CK_ULONG    flags;
   CK_SLOT_INFO_PTR    slot_info;
   CK_I_TOKEN_DATA_PTR    Token_data;
   CK_I_TOKEN_METHODS_PTR    methods;
};
```

其中 flags 用于描述连接到该 Slot 的 Session 的用户登录状态。slot_info 包含的信息主要是该 Slot 的文字描述、版本号和状态位。

Token_data 是指向 ck_i_Token_data_st 数据结构的指针，表示当前 Slot 中装载的 WToken，这样就将 Slot 与 Token 联系在了一起。 ck_i_Token_data_st 的数据结构定义如下：

```
struct ck_i_Token_data_st {
        CK_TOKEN_INFO_PTR Token_info;
        CK_SLOT_ID slot;
        CK_I_HASHTABLE_PTR object_list;
        CK_VOID_PTR impl_data;
    };
```

其中 Token_info 用来描述 WToken 的各种信息，包括 WToken 的名称，序列号，标志位，各种会话的记数，PIN 码长度，版本号等。会话计数是指对连接到 WToken 的会话的计数，这是因为卫士通加密设备允许同时打开的会话数目是有限的 32 个，应用程序在建立会话前首先要检查会话计数是否已达到最大值，若已经达到最大值，则建立会话失败。

Slot 是整数类型的成员变量，用于保存 WToken 所处的 Slot 的 ID 号。

Object_list 是一个 hash 表，用于保存在 WToken 上建立的 Object。

Impl_data 是一个 void 指针，它用来为 WToken 保存自己需要的数据。由于卫士通 PCI 加密设备能够存储 4 对公私钥对及其证书，我们将 WToken 的 impl_data 实现为指向如下结构的指针:

```
typedef struct ck_i_Token_impl_data_st {
```

```
    CK_OBJECT_HANDLE hPriKey;
    CK_OBJECT_HANDLE hPubKey;
    CK_OBJECT_HANDLE hPriKey2;
    CK_OBJECT_HANDLE hPubKey2;
    CK_OBJECT_HANDLE hPriKey3;
    CK_OBJECT_HANDLE hPubKey3;
    CK_OBJECT_HANDLE hPriKey4;
    CK_OBJECT_HANDLE hPubKey4;
    CK_OBJECT_HANDLE hCert;
    CK_OBJECT_HANDLE hCert2;
    CK_OBJECT_HANDLE hCert3;
    CK_OBJECT_HANDLE hCert4;
} CK_I_TOKEN_IMPL_DATA;
```

可见，这个结构用来记录在 WToken 中已经导入的私钥、公钥及证书的句柄。如果尚未导入其句柄，则被设置为 0。

Session 是 Cryptoki 中描述应用到 Token 的连接的数据结构。这个结构中的大部分数据都在 Cryptoki 库中分配空间和维护。但是 Session 数据结构中的 implement_data 域却必须在 WToken 中维护。在 WToken 中，implement_data 被实现为指向当前打开的安全模块所得到的 Channel（安全通道）的句柄。Channel 对应于 Session，即用户在使用加密设备的功能前要先建立一个 Channel，然后在使用其功能的过程中，把运算的中间结果、密钥等安全上下文放置在 Channel 的数据结构中。

在 Cryptoki 中已定义的 Session 结构中，有这样一些域：

```
    CK_VOID_PTR digest_state;          /* Created by C_DigestInit */
    CK_MECHANISM_TYPE digest_mechanism;
    CK_VOID_PTR encrypt_state;                  /* Created by C_EncryptInit */
    CK_MECHANISM_TYPE encrypt_mechanism;
    CK_VOID_PTR decrypt_state;                  /* Created by C_DecryptInit */
    CK_MECHANISM_TYPE decrypt_mechanism;
    CK_VOID_PTR sign_state;                     /* Created by C_SignInit */
```

CK_MECHANISM_TYPE sign_mechanism;

CK_VOID_PTR verify_state; /* Created by C_VerifyInit */

CK_MECHANISM_TYPE verify_mechanism;

分别用于保存摘要、加密、解密、签名和验证运算的安全上下文及其使用的算法，而所有的安全上下文都是类属指针类型的。在算法的初始化操作中，将初始化安全上下文数据结构，把它放入 Session 的相应数据域中；在 update 操作中，取出上下文中的内容加以处理，完成算法，在 final 操作中，释放上下文占用的空间。

5.2 Token 的自举初始化过程

应用程序在使用 WPKCS#11 时，必须首先调用 C_Initialize 函数初始化 Cryptoki，在 Cryptoki 的初始化过程中将会按照配置文件的设置，将系统中的一系列的 Token 加载到内存中，并初始化 Token。

WToken 初始化的工作流程如下所述：

1、为 WToken 中涉及的全局变量设置对应的互斥变量。在 WToken 库中将会创建各种密钥对象，它们都具有唯一的句柄，因此必须保证分配句柄操作的同步性，从而要创建名为 mutex_Tokenhandle 的互斥对象。同时，由于 WToken 库涉及到对公私钥及证书的存储，因此也必须保证对卫士通加密设备的非易失存储区域的互斥操作，从而要创建名为 mutex_nv 的互斥对象。

2、调用卫士通加密设备 API 提供的 SM_GetResourceStatus 函数获得 WToken 允许上层应用同时打开的会话的数目，以便设置 Token 结构的相应域。

3、完成 WToken 对应的 Slot 数据结构的分配与设置。Cryptoki 也即是通过该结构建立了与 Token 库的联系。在设置过程中，将把指向 WToken 库中库函数的指针，即 Cryptoki 库调用 Token 库方法的索引，传给 Cryptoki 库。

与对 Cryptoki 的初始化相对应，当上层应用对 WPKCS#11 的使用结束就应该调用 C_Finalize 函数来进行系统的结束处理，在这个过程中将会完成对 WToken 的结束处理，例如销毁已创建的互斥对象，删除未销毁的各种对象等。

5.3 卫士通安全模块简述

5.3.1 PCI 型硬件加密卡

WToken 库基于卫士通 PCI 加密卡实现，卫士通加密卡系国家级项目，是经国家密码管理办公室鉴定批准的密码设备，是以 PCI 插卡的形式安装在用户计算机中的数据加/解密设备。它提供数据加/解密、用户登录验证、数字签名与验证、密钥生成等功能，通过与应用程序的安全集成，能够在同一硬件平台上实现对各种安全应用的支持，为计算机信息提供安全保密服务，以防止各种安全威胁和攻击，特别适合计算机和网络设备的数据加/解密，整体技术处于国内先进水平。

该加密卡的密码算法、整机安全性设计方案均通过国家主管部门审查批准，具有完善的系统保护措施：所有密码处理均在卡内由硬件实现，所有的密钥决不以明文的形式出现在卡外；提供口令保护、屏蔽罩保护；密钥、密码算法支持自毁。

同时，其开发接口灵活，支持国际标准，产品互换性和升级性好，从而便于用户进行二次开发，可实现不同版本之间的平滑升级。

5.3.2 密钥管理

WToken 需要管理的密钥分为两类：对称密钥和非对称密钥。PKCS#11 中将证书以证书对象的形式来进行处理， WToken 也对它进行管理。为了使用的安全和方便，WToken 并不把实际的密钥上传给用户。取而代之的是把密钥对象的句柄传递给用户；用户使用密钥时，传入的也只是密钥对象的句柄。

5.3.2.1 对称密钥管理

WToken 支持 DES，3DES 及商密 33 算法密钥的创建，而在实际的使用中这些对称密钥并不需要保存在加密设备中，即它们是 Session Object，不具备持久性。每一个对称密钥对象都具有唯一对应的密钥句柄，在密钥生成之后，密钥句柄和密钥对象都将被插入到创建它的会话结构中的对象链表里，这样一来，无论是使用密钥还是删除密钥都会处理方便。

对称密钥的生成将启动安全模块内部的物理噪声源随机产生对称算法所需

的 IV 和 WK，并存放在安全模块内部供加密过程使用。因此，对于不再使用的密钥应及时销毁，以便释放安全模块内部占用的资源。而密钥对象作为自定义数据结构，如果在系统使用完毕后不加以删除势必会造成内存的泄漏，从而为系统的运行造成不良的后果。为此，我们在关闭会话的过程中，将检查当前会话拥有的密钥对象是否删除完毕，若还有剩余则及时销毁。这是因为，对称密钥对象是由会话创建并保存的，因而让每个会话管理自己的对称密钥对象是合理的。

此外，调用卫士通 PCI 安全模块 API 生成对称密钥返回给 WToken 的仅仅是该密钥在安全模块内部的索引值。WToken 将把该索引值作为对称密钥对象的 Value 属性值，从而提高了密钥数据的安全性。当用户需要获得密钥数据本身时，只需使用密钥导出函数即可。

5.3.2.2 非对称密钥管理

与对称密钥相比，非对称密钥的管理就要复杂些。这是因为在实际的创建中，非对称密钥即可被设置为 Session Object，也可被设置为 Token Object。当其作为 Session Object 时，其管理方式同对称密钥。当其作为 Token Object 时，由于其具有持久性，因而必须保存在安全模块的非易失性存储区内。而非易失性存储区的容量是非常有限的，因此，我们只将非对称密钥对象的 CKA_ID、CKA_SUBJECT 属性值和密钥内容保存在其中，从而能够有效地区分所保存密钥，便于对密钥的使用、查找及删除。事实上，对象的 CKA_ID 属性就是用来唯一性的标识不同的密钥的。对于一对公私钥，其 CKA_ID 和 CKA_SUBJECT 属性值都应保持相同的值，并且必须与其对应的证书的这些属性相一致。

当用户需要使用保存在非易失性存储区中的非对称密钥及证书时，必须使用 CKA_ID 或 CKA_SUBJECT 属性值作为查询条件调用 WPKCS#11 的对象查找函数来进行查找，以获得对象句柄来使用。为了便于查询，我们将这些非对称密钥和证书在对象查询初始化时一并以对象的形式插入到 WToken 库中的全局对象链表中。同样地，由于密钥对象作为自定义数据结构，如果在系统使用完毕后不加以删除会造成内存的泄漏，因此在系统结束处理的过程中，将检查该全局对象链表拥有的密钥对象是否删除完毕，若还有剩余则及时销毁。

同时为了安全性，用户使用这些非对称密钥对象时获得的也仅仅是密钥的句

柄，而不是密钥数据本身。

5.4 Token 接口设计与实现

1）安全通道的建立与关闭

卫士通 PCI 加密卡具备安全通道的概念，安全通道相当于 PKCS#11 中的会话，即用户在使用加密卡的功能前要首先建立一个安全通道，然后在使用其功能的过程中，把运算的中间结果、密钥等安全上下文放置在安全通道对应的数据结构中。在 WToken 库中，建立安全通道的过程就与 Cryptoki 中建立会话的过程相联系，即用户在使用卫士通安全模块提供的密码功能前必须调用 Cryptoki 提供的 C_OpenSession 函数建立会话，而该函数功能的实现就包含了对卫士通加密卡安全通道的建立过程。

安全通道的建立函数将为调用者在安全模块内部分配资源。事实上，在使用卫士通 PCI 加密卡编程接口中定义的任何函数之前，必须已经成功地建立了安全通道。多个调用者可以独立地调用该函数，从而得到各自的安全通道操作句柄。安全通道的资源是有限的，因此在调用该函数成功返回后，如不再使用加密卡应调用安全通道的关闭函数来关闭一个安全通道。否则，会造成安全模块内部资源的无效占用。

2）用户的登录与注销

1. CI_SM_Login

函数原型：CK_RV CI_SM_Login

(

CK_I_SESSION_DATA_PTR session_data, 会话数据

CK_USER_TYPE userType, 用户类型

CK_CHAR_PTR pPin, 用户 PIN 码

CK_ULONG ulPinLen 用户 PIN 码长度

)；

功能描述：该函数将使用 Cryptoki 层提供的用户 PIN 码调用卫士通 PCI 加密卡 API 的用户登录函数 SM_Login 来进行用户登录。如果成功，它将按顺序对该 session 所在 slot 中的所有 session 重复调用 SM_Login。如果有失败，则必须对所

有调用 SM_Login 成功的 session 调用 SM_Logout。这种对连接到同一 Token 的所有 Session 保持同样的登录状态的实现方式是基于对 PKCS#11 规范的支持。在成功调用该函数以前，安全模块内部的各种敏感数据都处于加密状态。该函数成功调用后，安全模块将敏感数据在模块内部脱密，为以后的涉密操作做好准备。否则，所有涉密的编程接口函数调用都将返回失败。因此，在使用卫士通 PCI 加密卡的密码功能前调用该函数是必不可少的。

2. CI_SM_Logout

函数原型：CK_RV CI_SM_Logout(

CK_I_SESSION_DATA_PTR session_data 会话数据

);

功能描述：该函数与用户登录函数相对应，实现用户的注销功能。它将清除安全模块内部所有脱密的敏感数据。该函数调用后，如要调用涉密的编程接口函数，则必须重新调用 CI_SM_Login 函数。

3）数据的加密与解密

对于数据的加密，可以通过组合使用 CI_SM_EncryptInit、CI_SM_Encrypt 或 CI_SM_EncryptInit、CI_SM_EncryptUpdate、CI_SM_EncryptFinal 来完成。类似地，对于数据的解密，也可以通过组合使用 CI_SM_DecryptInit、CI_SM_Decrypt 或 CI_SM_DecryptInit、CI_SM_DecryptUpdate、CI_SM_DecryptFinal 来完成。

1. CI_SM_ EncryptInit

函数原型：CK_RV CI_SM_EncryptInit

(

CK_I_SESSION_DATA_PTR session_data, 会话数据

CK_MECHANISM_PTR pMechanism, 加密算法

CK_I_OBJ_PTR key_obj 加密密钥

);

功能描述：该函数用 key_obj 所标识的密钥（可以是对称密钥、也可以是非对称密钥）来初始化加密过程。该函数和 CI_SM_EncryptUpdate、CI_SM_EncryptFinal

一起构成了多块（也可以是单块）数据块加密的三段式结构。对于大量数据来说，可以首先调用该函数，然后多次调用 CI_SM_EncryptUpdate，最后调用一次 CI_SM_EncryptFinal 完成对数据块的加密；对于少量数据来说，可以首先调用该函数，然后调用一次 CI_SM_Encrypt 函数完成单块数据加密。

2. CI_SM_ Encrypt

函数原型：CK_RV CI_SM_Encrypt

(

CK_I_SESSION_DATA_PTR	session_data,	会话数据
CK_BYTE_PTR	pData,	明文数据
CK_ULONG	ulDataLen,	明文数据长度
CK_BYTE_PTR	pEncryptedData,	密文数据
CK_ULONG_PTR	pulEncryptedDataLen	密文数据长度

);

功能描述：该函数完成一次单块数据块的加密。应用程序在使用该函数对数据进行加密时，一般应分为两个步骤，第一步令 pData 为 NULL_PTR 调用该函数获得所用加密设备的加密数据块长度（令 pEncryptedData 为 NULL_PTR 时调用该函数可获得加密数据后生成的密文数据块长度），此时的参数 pulEncryptedDataLen 将获得相应的数据块长度值（以字节为单位）。第二步调用该函数进行数据加密操作。

由于篇幅所限，对 CI_SM_EncryptUpdate、CI_SM_EncryptFinal 及解密函数不做详细介绍。目前，WToken 库对加、解密支持国密办商密 33 算法、3DES 算法、DES 算法及 RSA 算法。

4）数据的摘要、签名与验证

WToken 提供 CI_SM_DigestInit、CI_SM_Digest、CI_SM_DigestUpdate 及 CI_SM_DigestFinal 函数来完成数据的摘要过程，目前支持的算法有：MD2、MD5、SHA、SHA1 及卫士通 HASH 算法。

对数据的签名与验证，则分别实现了 CI_SM_SignInit、CI_SM_Sign、

CI_SM_SignUpdate 及 CI_SM_SignFinal 函数和 CI_SM_VerifyInit 、CI_SM_Verify、CI_SM_VerifyUpdate 及 CI_SM_VerifyFinal 系列函数。为了实现签名验证的互通性，我们对卫士通加密卡 RSA 运算所得出的私钥加密结果在 WToken 库中进行了标准的填充。

5）密钥的生成

WToken 提供对称和非对称密钥的生成，具体由以下两个函数实现。

1. CI_SM_ GenerateKey

函数原型：CK_RV CI_SM_GenerateKey (

CK_I_SESSION_DATA_PTR session_data, 会话数据

CK_MECHANISM_PTR pMechanism, 密钥生成算法

CK_I_OBJ_PTR key_obj, 生成的密钥对象

CK_ATTRIBUTE_PTR pTemplate, 密钥对象模板

CK_ULONG ulCount, 密钥对象模板属性个数

CK_OBJECT_HANDLE_PTR phKey 生成的密钥对象句柄

);

功能描述：该函数生成一个对称密钥。在 WPKCS#11 中，每一个 Object 都包含有一个属性集，属性集中每一个属性都应该对应有一个特定的值，不管是创建时上层接口赋予的还是系统默认值。因此，在生成一个密钥对象时就需要为其属性设定相应的值，针对该函数，如果调用者不为将生成的对称密钥对象设置对象属性，可以将 pTemplate 置为 NULL_PTR, ulCount 置为 0，此时函数内部将把密钥对象的所有属性设置为系统默认属性。

下面以生成 DES 密钥对象为例进行说明对其属性模板的设置：

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_DES;
CK_UTF8CHAR label[] = “A DES secret key object”;
CK_BBOOL true = TRUE;
CK_ATTRIBUTE template[] = {
   {CKA_CLASS, &class, sizeof(class)},
   {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
```

```
    {CKA_TOKEN, &true, sizeof(true)},
    {CKA_LABEL, label, sizeof(label)-1},
    {CKA_ENCRYPT, &true, sizeof(true)},
    {CKA_DECRYPT, &true, sizeof(true)},
};
```

其中，CKA_CLASS 属性说明对象类型为对称密钥对象，CKA_KEY_TYPE 属性说明要生成的对称密钥为 DES 密钥，CKA_TOKEN 属性说明创建的密钥对象具有持久性，CKA_LABEL 属性则用来描述该对象的信息，CKA_ENCRYPT 和 CKA_DECRYPT 分别说明该密钥对象可用来对数据进行加密和解密。

2. CI_SM_ GenerateKeyPair

函数原型：CK_RV CI_SM_GenerateKeyPair (

```
  CK_I_SESSION_DATA_PTR     session_data,    会话数据
  CK_MECHANISM_PTR          pMechanism,      密钥生成算法
  CK_I_OBJ_PTR              public_key_obj,  生成的公钥对象
  CK_I_OBJ_PTR              private_key_obj, 生成的私钥对象
  CK_ATTRIBUTE_PTR  pPublicKeyTemplate,  公钥对象模板
  CK_ULONG  ulPublicKeyAttributeCount,   公钥对象模板属性个数
  CK_ATTRIBUTE_PTR  pPrivateKeyTemplate, 私钥对象模板
  CK_ULONG  ulPrivateKeyAttributeCount,  私钥对象模板属性个数
  CK_OBJECT_HANDLE_PTR phPublicKey,      生成的公钥对象句柄
  CK_OBJECT_HANDLE_PTR phPrivateKey      生成的私钥对象句柄
);
```

功能描述：该函数生成一对 RSA 密钥对。

典型的 RSA 公钥对象的属性模板具有如下属性:

CKA_ENCRYPT 属性说明该对象可用来对数据进行加密；

CKA_VERIFY 属性说明该对象可用来对数字签名进行验证；

CKA_WRAP 属性说明该对象可以被用来加密导出密钥；

CKA_MODULUS_BITS 属性说明该公钥对象的 MODULUS 的长度；

CKA_PUBLIC_EXPONENT 属性说明该公钥对象的 Public Exponent 的值。

而典型的 RSA 私钥对象的属性模板则具有如下属性:

CKA_PRIVATE 属性说明该对象为私有对象；

CKA_SUBJECT 属性则指明该对象的主题描述；

CKA_ID 属性指明该对象的 ID 值；

CKA_SENSITIVE 属性说明该对象是敏感的，即不能以明文导出；

CKA_DECRYPT 属性说明该私钥对象可用来对数据进行解密；

CKA_SIGN 属性说明该私钥对象可用来对数据进行签名；

CKA_UNWRAP 属性说明该对象可以被用来解密导入密钥。

6）密钥的导入与导出

所谓密钥的导出是指将加密设备产生的密钥安全地从加密设备中以字节流地形式加密导出（公钥数据则不需要被加密）。被导出的密钥数据由于已被加密，因此就能在非安全的传输信道上进行传送，并且也能够存储在不具备安全性的存储介质中。这样一来就使得密钥的传递更加方便和安全，因为仅仅获得被加密的密钥数据是不能得到密钥本身的，接收者还需要相应的解密密钥才能够使用被导出的密钥。

而密钥的导入则是密钥导出的逆过程，它将被加密的导出密钥用对应的解密密钥解密导入到相应的加密设备中，从而获得对该密钥的操作句柄。

1. CI_SM_WrapKey

函数原型：CK_RV CI_SM_WrapKey (

CK_I_SESSION_DATA_PTR session_data, 会话数据

CK_MECHANISM_PTR pMechanism, 密钥导出算法

CK_I_OBJ_PTR wrap_key_obj, 用来导出密钥的加密密钥

CK_I_OBJ_PTR key_obj, 待导出密钥

CK_BYTE_PTR pWrappedKey, 被加密导出的密钥数据

CK_ULONG_PTR pulWrappedKeyLen 被加密导出的密钥数据字节长度

);

功能描述：该函数将密钥加密导出（公钥数据则不需要被加密）。

目前，WToken 的密钥导出算法为 RSA 算法，因而用来导出密钥的加密密钥 wrap_key_obj 即为接收导出密钥的接收者的公钥。这样就保证了用接收者的公钥加密的导出密钥能够且仅能够被接收者正常使用，因为只有接收者的私钥能解密被加密的导出密钥。当然，如果待导出的密钥为公钥，则 wrap_key_obj 参数将置为 0。

2. CI_SM_UnWrapKey

函数原型：CK_RV CI_SM_UnwrapKey (

CK_I_SESSION_DATA_PTR	session_data,	会话数据
CK_MECHANISM_PTR	pMechanism,	密钥导入算法
CK_I_OBJ_PTR	unwrap_key_obj,	用来导入密钥的解密密钥
CK_BYTE_PTR	pWrappedKey,	待导入的密钥数据字节流
CK_ULONG	ulWrappedKeyLen,	待导入的密钥数据字节流长度
CK_ULONG_PTR	pKey_handle	获得的导入密钥的对象句柄

);

功能描述：该函数将密钥数据解密导入到加密设备中获得对应的密钥操作句柄（公钥数据则不需要被解密）。

目前，WToken 的密钥导入算法为 RSA 算法，因而用来导入密钥的解密密钥 unwrap_key_obj 即为接收导出密钥的接收者的私钥。这样就保证了用接收者的公钥加密的导出密钥能够且仅能够被接收者正常使用，因为只有接收者的私钥能解密被加密的导出密钥。当然，如果被导出的密钥为公钥，则 unwrap_key_obj 参数也将被置为 0。

密钥的导入与导出被广泛地用于保护被传递的密钥数据，下面以文件 F1 的安全传送为例来说明其使用流程，其中 A 为文件的发送者，B 为文件的接收者：

1．A 产生一个会话密钥 K。
2．A 以 B 的公钥及会话密钥 K 为参数调用密钥导出函数 CI_SM_WrapKey，得到一定格式的密钥数据字节流。
3．A 将 2 中得到的密钥数据字节流作为新创建文件 F2 的数据头。

4．A 用会话密钥 K 加密文件 F1 的内容，将密文作为待传送文件 F2 的数据正文 W。

5．A 将文件 F2 传送给 B。

6．B 从文件 F2 中提取出数据头（即会话密钥 K 的导出数据），连同 B 的私钥作为参数调用密钥导入函数 CI_SM_UnWrapKey，得到会话密钥 K 的操作句柄 Hkey。

7．B 从文件 F2 中提取出数据正文 W，并以 Hkey 为解密密钥解密 W，获得传送文件的明文 P。

5.5 密钥管理

WToken 需要管理的密钥分为两类：对称密钥和非对称密钥。PKCS#11 中将证书以证书对象的形式来进行处理， WToken 也对它进行管理。为了使用的安全和方便，WToken 并不把实际的密钥上传给用户。取而代之的是把密钥对象的句柄传递给用户；用户使用密钥时，传入的也只是密钥对象的句柄。

第六章 基于 PKCS#11 的 Token 开发包设计与实现

PKCS#11 协议是一个分层协议，RSA 公司规范了协议层的标准，给用户调用的是协议层之上的抽象层。因此常常需要为用户提供从协议层到 Token 的完整服务。PKCS#11 软件提供者通常需要提供上述的两个模块（形式上可以整合在一起），一般说来针对不同的下层设备协议层无需更改，而令牌层则必须针对该设备重新改写。对繁多的设备来说这就存在很大的冗余：不同的 Token 有许多相同相似之处，而且尽管不同的设备提供的服务表面上不尽相同，但是还是可以通过类似模板的方式套用。因此为了加快 Token 开发速度，提高 Token 层软件的质量，抽象出了 Token 开发包。

Token 开发包为基本 C++开发，分为全局变量设置、PKCS#11 相关安全服务、信息查询、为开发包服务的子服务，下面对变量设置、安全服务和子服务相关的类进行说明。

◆ 初始化及信息查询类继承图为：

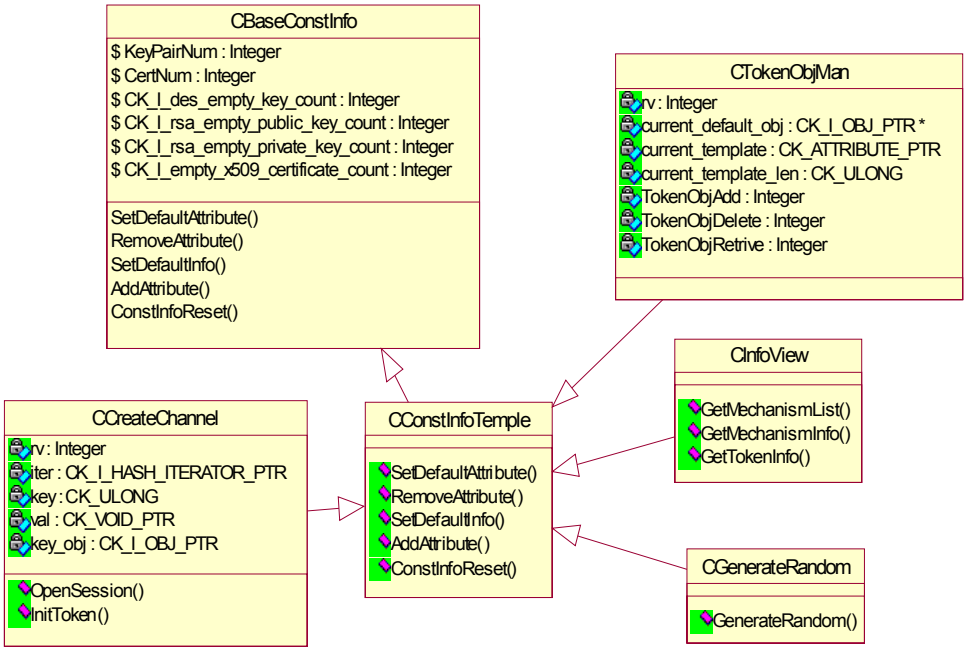

图 6－1 初始化及信息查询类继承图

◆ 加密操作类继承图为：

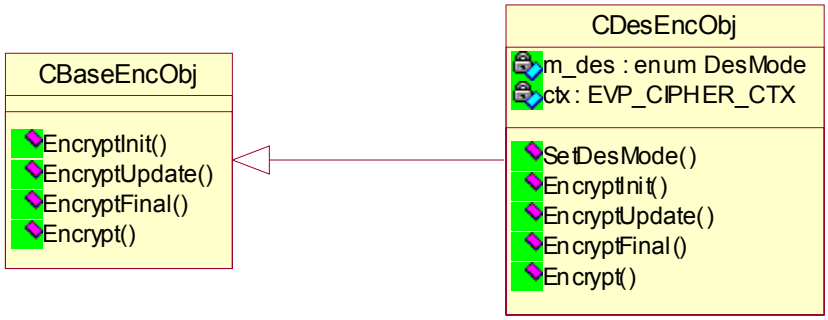

图 6－2 加密操作类继承图

◆ 摘要操作类继承图：

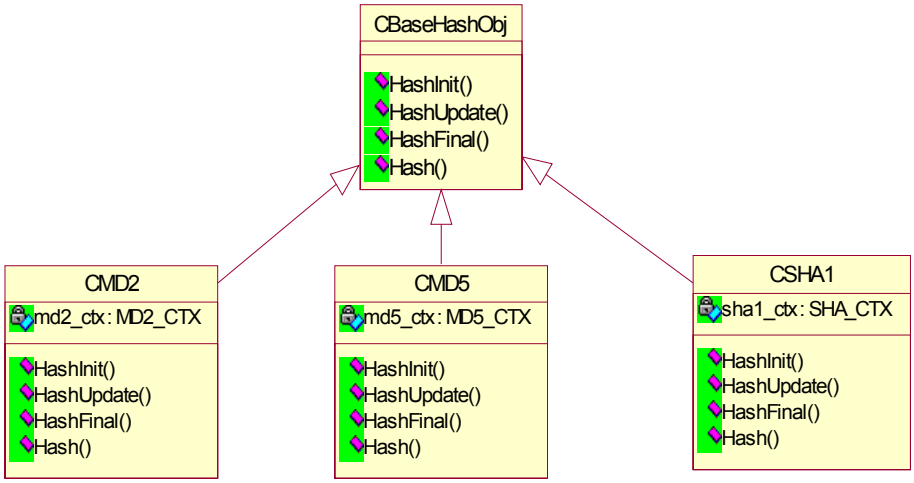

图 6－3　摘要操作类继承图

6.1 全局变量定义

6.1.1 CBaseConstInfo 类

给出全面的常量声明，接口完成初始化等工作，此类为所有非软算法的抽象基类。要求在工具库实现中重载 CBaseConstInfo 类，完成常用的一个实现类 CConstInfo，并允许以后的开发者对它重载完成其它的常量设置、更新等工作。具体定义如下：

```
class TOOLKITS_API CBaseConstInfo
{
protected:
    static    int KeyPairNum;
    static    int CertNum;
        //define all objects attribute template's length
    static    int CK_I_des_empty_key_count;
    static    int CK_I_rsa_empty_public_key_count;
    static    int CK_I_rsa_empty_private_key_count;
    static    int CK_I_empty_x509_certificate_count;

        //all statics must be init by base_class or derive_class
    static    CK_TOKEN_INFO SM_Token_info;
    static    CK_I_TOKEN_IMPL_DATA SM_impl_data;
    static    CK_I_TOKEN_DATA SM_Token_data;   //only for SM_slot_data
    static    CK_SLOT_INFO SM_slot_info;              //for SM_slot_data &
SM_TokenInit
    ……
public:
    CBaseConstInfo();
```

```
    virtual ~CBaseConstInfo();

    virtual int SetDefaultAttribute(int rsa_count,int certs_count,int des_count,int
rsa_pubkey_count,int rsa_prikey_count,int cert_count) = 0;
    virtual int RemoveAttribute(CK_ATTRIBUTE_TYPE att_type,void *pValue,
CK_ULONG valueLen) = 0;
    virtual int SetDefaultInfo() = 0;
    virtual int AddAttribute(CK_OBJECT_CLASS
obj_type,CK_ATTRIBUTE_TYPE att_type,void *pValue,CK_ULONG valueLen) =
0;
    virtual void ConstInfoReset() = 0;

protected:
    static char* globalpin;
        };
```

派生类中使用了子对象（基类的对象），而基类的构造函数非缺省而且需要改变参数，则要求在派生类的构造函数中将子对象一同初始化。

6.1.2 CconstInfoTemple 类

```
    class TOOLKITS_API CConstInfoTemplate : public CBaseConstInfo
{
public:
    int CI_SM_NewHandle(CK_ULONG_PTR handle);
    virtual int SetDefaultAttribute(int rsa_count,int certs_count,int des_count,int
rsa_pubkey_count,int rsa_prikey_count,int cert_count);
    virtual int RemoveAttribute(CK_ATTRIBUTE_TYPE att_type,void *pValue,
CK_ULONG valueLen);
    virtual int SetDefaultInfo();
    virtual int AddAttribute(CK_OBJECT_CLASS
obj_type,CK_ATTRIBUTE_TYPE att_type,void *pValue,CK_ULONG valueLen);
    CConstInfoTemplate(const char *TokenName);
    CConstInfoTemplate(int k_num,int c_num,int des_count,int rsa_pub_count,int
rsa_pri_count,int cert_count);
    CConstInfoTemplate();    //do nothing,just
    virtual ~CConstInfoTemplate();
    virtual void ConstInfoReset();
};
```

此类的功能是用来为不同的硬件设备提供一些初始化的设置，如：可存储密钥对数目、可存储证书个数、同时可打开多少个连接、Token 名称、Token 缺省属性等等。初始状态的设置通过该类构造和析够函数在实例化的时候根据实际情

况确定。

6.2 安全服务

6.2.1 CBaseEncObj 类

```
class TOOLKITS_API CBaseEncObj
{
public:
    virtual int EncryptInit(UCHAR *key,UCHAR *iv) = 0;
    virtual int EncryptUpdate(UCHAR *pPart,LONG PartLen,UCHAR
*pEncryptedPart,LONG& EncryptedPartLen) = 0;
    virtual int EncryptFinal(UCHAR *pEncryptedLastPart,LONG&
EncryptedLastPartLen) = 0;
    virtual int Encrypt(UCHAR *pSource,LONG SourceLen,UCHAR
*pEncrypted,LONG& EncryptedLen) = 0;

    CBaseEncObj();
    virtual ~CBaseEncObj();
};
```

此类为加密运算的虚基类，需要增加的具体加密类均由此派生出来，这样为所有的软件实现的加密类提供了一致的实现和接口，方便开发和维护。

6.2.2 CDesEncObj 类

```
class TOOLKITS_API CDesEncObj : public CBaseEncObj
{
public:
    CDesEncObj();
    virtual ~CDesEncObj();
    int SetDesMode(enum DesMode t_des);
    int SetPaddingOR(BOOL istoset);      //缺省的情况为填充模式
    int Encrypt(UCHAR *pSource,LONG SourceLen,UCHAR *pEncrypted,LONG&
EncryptedLen);
    int EncryptInit(UCHAR *key,UCHAR *iv);
    int EncryptUpdate(UCHAR *pPart,LONG PartLen,UCHAR
*pEncryptedPart,LONG& EncryptedPartLen);
    int EncryptFinal(UCHAR *pEncryptedLastPart,LONG& EncryptedLastPartLen);
private:
    enum DesMode m_des;
    EVP_CIPHER_CTX ctx;
```

```
};
```

此类为 DES 软算法的实现类。SetDesMode()：设置 DES 算法的模式（CBC、ECB 等），SetPaddingOR()：是否需要填充模式，Encrypt()、EncryptInit()、EncryptUpdate()、EncryptFinal()均为 DES 加密实现。

6.3 子服务

6.3.1 CsubToolKits 类

```
class CSubToolKit
{
public:
    CSubToolKit();
    virtual ~CSubToolKit();
    //for standard_signature
    int PreRSASign(int HashID,int rsabits,unsigned char *m,int m_len,unsigned char
*tosign,int *tosignLen,int padding);
    int PreRSAVerify(int HashID,int rsabits,unsigned char *decSign,int
decSignLen,unsigned char *outm,int *outm_len,int padding);
    int SetGlobalSessionHandle(unsigned long session_handle);
    int DelGlobalSessionHandle(unsigned long session_handle);
    int GetGlobalSessionHandlePOS(unsigned long session_handle);
};
```

这些为其它实现提供子功能，如 PreRSASign()：根据 PKCS#1 的标准将数据填充成可识别的待签名数据。PreRSAVerify()：执行 PreRSASign()的逆操作。SetGlobalSessionHandle()：设置全局会话句柄标志。DelGlobalSessionHandle()：取消设置全局会话句柄标志。GetGlobalSessionHandle()：获得全局会话句柄标志。

第七章 课题研究成果应用案例

本课题的研究成果 2003 年应用于北京信安世纪公司的网银系统，作为其使用卫士通密码设备的公用平台。从下图可以看出上层用户怎样通过 PKCS#11 软件包来实现对卫士通密码设备的无缝访问的，系统应用例图：

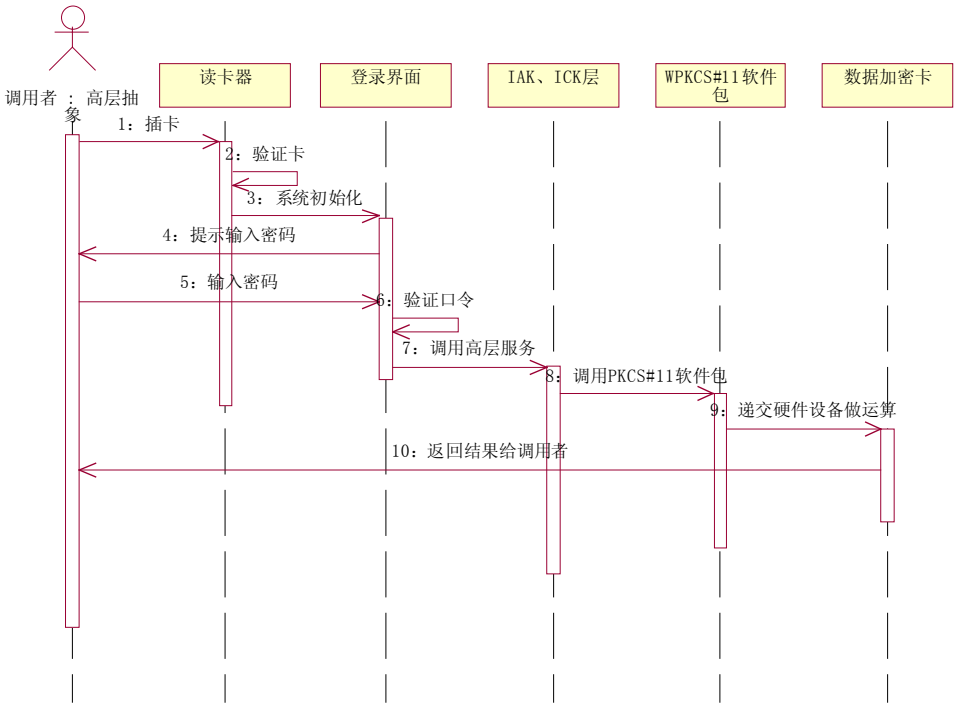

图 7－1　系统应用类图

第八章 结束语

一 项目研究的主要依据

基于 PKCS#11 的公共安全平台 WPKCS#11，由电子科大---卫士通信息安全联合实验室开发。其中，WPKCS#11 是基于 RSA 公司提出的工业标准 PKCS#11 规范的相应实现。

随着网络通讯技术的飞速发展，信息安全是网络应用不可避免的问题，密码学是解决这个问题的最根本方法。许多软硬件开发商，为了保证自己产品的安全，开发了基于复杂且种类繁多的密码学算法的安全模块，由于没有统一的技术标准，安全模块的体系结构和接口各不相同，这就给用户的使用造成了困难，安全服务很难共用。

公开密钥密码学标准（Public Key Cryptomgaphy Standard）PKCS 是 RSA 数据安全公司为公开密码学提供的一个工业标准接口，其中的第十一个规范 PKCS#11 是密码标记的 API 标准。该标准详细规定了一个称为 Cryptoki 的编程接口，并给出了一个通用的逻辑模型，这使得它可以用于各种可移植的密码设备，用户不需要知道详细的技术细节，就可以在可移植的密码设备上完成加密操作。总的说来，PKCS#11 协议有两个目标：

1. 屏蔽各种安全实现之间的差异，向用户提供统一接口，使安全服务的具体实现方法对用户透明化；
2. 使用户可以在多个安全服务具体实现之间共享资源。

WPKCS#11 公共安全平台以 PKCS#11V2.0 为目标，目前已实现 Windows、UNIX 平台版本。

公共安全平台的研究与实现不仅对加密设备的普及和推广具有广泛的经济、社会效益，对提高人们的安全意识也有着深远的意义。

二 项目研究的主要过程

研究内容：

- Cryptoki 静态库的实现
- Token 动态库的实现
- 支持多进程、多线程的研究
- 支持加密设备拔插功能的研究

- 对象访问控制的实现
- Token Object 共享方式的研究

关键技术：

- 基于 Windows，Linux 的实现
- 互斥机制
- 日志处理
- 内存分配和回收

达到的技术指标：

- 实现了 PKCS#11V2.0 规定的主要接口功能
- 支持多线程、多进程
- 用户登录和 Session 的用户状态变化更符合 PKCS#11 规范
- 实现了对象的访问控制
- 支持加密设备的拔插功能
- 日志详尽
- 执行安全功能效率高

三 结论

经过实践，WPKCS#11 可以稳定而又高效地完成 PKCS#11V2.0 规定的接口功能。目前对公共安全平台的实现在国内比较少见，因此具有广阔的市场前景。

四 经验体会

在开发过程中，我们始终关注着国内外公共安全平台的最新发展情况，并以创新和实用为基点，力求软件的高可靠性和高效性。这些都在系统中得到体现。

本系统更有待向产品化的方向发展，只有让更多的用户使用和评价我们系统，才能构成发展的良性循环。这些也正是我们下一步工作的重点。

第九章 参考文献

[1] PKCS #11 v2.11 Final Draft: Cryptographic Token Interface Standard，RSA Laboratories , June 2001

[2] PKCS #1 *RSA Encryption Standard* Version 2.0, RSA Laboratories, October 1, 1998

[3] PKCS#8 Private-Key Information Syntax Standard Version 1.2, RSA Laboratories, November 1,1993

[4] The New Integrated Security Platform ， Security Vol.37, No.6 pp.23-24

[5] *Balacheff, Boris*; *Chen, Liqun*; Computing platform security in cyberspace Information Security Technical Report 5 1 2000 Elsevier Sci Ltd pp. 54-63 1363-4127

[6] Burce Schneier. 应用密码学：协议、算法与 C 源程序(吴世忠，祝世雄，张文政等译). 北京：机械工业出版社，2000

[7] 关振胜. 公钥基础设施 PKI 与认证机构 CA. 北京：电子工业出版社

[8] Carlise Adams Steve Lloyd. 公开密钥基础设施——概念、标准和实施(冯登国等译). 北京：人民邮电出版社，2001

[9] Naganand Doraswamy, Dan Harkins. IPSec——新一代因特网安全标准. 北京：机械工业出版社，2000

[10] 梁志龙,张志浩. 公共安全平台的研究与设计. 计算机系统应用 2001.09

[11] TongSEC安全中间件白皮书. http://www.tongtech.com/cpzn/TongSEC.pdf

[12] PKCS#11用户手册. 成都：卫士通信息技术有限公司

[13] Preiss,胡广斌. 面向对象的 C++设计模式. 北京：电子工业出版社

[14] Plauger,王昕. C++ STL 中文版. 北京：中国电力出版社

[15] Bruce Eckel. C++编程思想. 北京：机械工业出版社

[16] Stanley. Essential C++中文版. 武汉：华中科技大学出版社

[17] Herb Sutter. Exception C++中文版. 北京：中国电力出版社

第十章 致谢

在论文即将完成之际，我谨向我的导师佘堃教授致以真诚的感谢！佘老师在信息安全和中间件计算领域有卓越的成就。作为佘老师的硕士研究生，无论在专业还是做人他都给我我莫大的无私的提点和帮助，使我终身受益。此外我还要感谢指导我们工作学习的的周明天教授。他细致入微的工作和严谨的学术态度使我受益非浅。

我还要感谢成都卫士通公司的谭兴烈教授以及其他关心、鼓励和帮助我的朋友们。

另外我还要感谢我的师兄许立、罗建超、谢鸿波，我的同窗好友杨四铭、黄均才、陈阳、熊杰颖、牛新征等，没有他们的帮助我不可能顺利的完成毕业设计。

真诚的感谢我的父母、姐姐对我的关心和支持。

最后，感谢曾经教育和帮助过我的所有老师，衷心地感谢为评审论文而付出辛勤劳动的教授和专家们！

第十一章 个人简历、在学期间的研究成果及发表的学术论文

一、个人简历

沈仟，江苏扬州人，2001 年 6 月毕业于电子科技大学应用数学学院，获得工学学士学位，同年考取电子科技大学计算机学院攻读计算机软件与理论硕士学位。

本科阶段，成绩优异，分别获得人民奖学金特等奖、二等奖、三等奖，院荣誉学生，校优秀毕业生，全国大学生数学建模竞赛一等奖，美国大学生数学建模竞赛荣誉提名(honorable mention)；

二、研究生期间参与完成的项目课题及研究成果

1、 作为主研参与开发了基于 PKCS#11 协议的公共安全平台项目，该项目通过鉴定，并获得四川省科技进步三等奖，目前应用于北京信安世纪网银系统项目中

2、 自主开发了基于安全模块的 WestoneCSP，该项目通过鉴定，并获得成都市科技进步一等奖，目前应用于北京国家计划经济委员会的内网项目中

3、 参与开发了企业级 CA，目前该项目应用于成都党政网的安全认证

三、学术论文

沈仟,佘堃,周明天等. 智能卡安全中间件的研究与开发. 计算机应用, 2004.6

佘堃,沈仟,周明天. 背包问题在硬币抛掷协议上的研究. 电子科技大学学报, 2003.6