同济大学
TONGJI UNIVERSITY

硕士学位论文

基于安全 SD 卡的 Elastos 数字认证机制的研究与实现

（国家核高基重大专项计划资助项目　编号：2009ZX01039-002-002）

姓　　名：申波

学　　号：0820080326

所在院系：电子与信息工程学院

学科门类：计算机科学与技术

学科专业：计算机应用技术

指导教师：陈榕 教授

副指导教师：顾伟楠 教授

二〇一一年三月

同濟大學
TONGJI UNIVERSITY

A dissertation submitted to

Tongji University in conformity with the requirements for

the degree of Master of Science

The Research and Implementation of Elastos PKI Environment Based on Security SD Card

(Supported by the XXX Plan, Grant No.XXX)

Candidate: Bo Shen

Student Number: 0820080326

School/Department: School of Electronics and Information Engineering

Discipline: Computer Science and Technology

Major: Computer Application Technology

Supervisor: Prof. Rong Chen

March, 2011

基于安全SD卡的ELASTOS数字认证机制的研究与实现

申波 同济大学

学位论文版权使用授权书

本人完全了解同济大学关于收集、保存、使用学位论文的规定，同意如下各项内容：按照学校要求提交学位论文的印刷本和电子版本；学校有权保存学位论文的印刷本和电子版，并采用影印、缩印、扫描、数字化或其它手段保存论文；学校有权提供目录检索以及提供本学位论文全文或者部分的阅览服务；学校有权按有关规定向国家有关部门或者机构送交论文的复印件和电子版；在不以赢利为目的的前提下，学校可以适当复制论文的部分或全部内容用于学术活动。

学位论文作者签名：

年　　月　　日

同济大学学位论文原创性声明

本人郑重声明：所呈交的学位论文，是本人在导师指导下，进行研究工作所取得的成果。除文中已经注明引用的内容外，本学位论文的研究成果不包含任何他人创作的、已公开发表或者没有公开发表的作品的内容。对本论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明。本学位论文原创性声明的法律责任由本人承担。

学位论文作者签名：

年　　月　　日

摘要

随着移动互联网的深入发展，移动设备，尤其是智能手机的计算能力得到大幅度提升，它与外界的数据交换越来越频繁，数据交换量也越来越大，在这些设备上从事电子商务活动成为传统电子商务活动的一种新的方式。而移动电子商务成为现实的基础是保证通信双方的身份真实性和传输的数据的安全性。

Elastos是一款基于微内核的完全面向构件化的操作系统，具有灵活的构件组装的特点。CAR（Component Assembly Runtime）构件技术是Elastos操作系统中面向构件的编程模型，表现为一组编程规范，规定了构件间相互调用的标准，包括构件、类、对象、接口等定义与访问构件对象的规定，使得二进制构件能够自描述，能够在运行时动态链接，不同的构件动态组装成功能更丰富的能力。

移动互联网环境下，Elastos上的业务应用和服务越来越丰富，其中移动电子商务应用，例如电子签章和电子合同等，要求对传输的数据进行加密，保证敏感数据的保密性，同时还必须保证通信双方的真实身份。此外，它们的运行还需要Elastos具有一个安全稳定的运行环境，提供运行时安全检查。

为满足这些安全需求，本文提出了基于安全SD卡的Elastos数字认证机制，通过该机制来验证CAR构件运行时安全，并以安全构件的形式向其他CAR构件或上层应用程序提供安全服务，满足了移动设备在移动互联网范围内对传输的数据完整性和身份真实性认证的安全需求，为移动电子商务和分享服务等移动互联网业务服务提供安全技术支持，同时采用Java与CAR互操作技术，满足了Elastos和Android两种计算模型对安全服务的需要。

论文首先介绍了移动互联网环境下移动设备数据交换涉及的安全性问题，以及国内外对这些问题的研究现状。然后，介绍了Elastos、CAR构件技术、Java与CAR互操作技术和移动设备安全SD智能卡相关技术。其次，结合安全SD智能卡的特点，设计了Elastos数字认证机制，实现了相关功能模块。最后，对这种数字认证机制的课题研究进行阶段性总结和对后续工作进行展望。

关键字：Elastos，CAR构件，移动设备，安全，数字认证

ABSTRACT

With the further development of mobile internet, the computing ability of mobile devices, particularly smart phones, has been greatly enhanced. The exchanged data of these devices with others is increasingly frequent, and the amount is more than ever before. And so e-commerce activities in these devices of mobile internet become a new way of the traditional one in the Internet. The mobile e-commerce will come true, ensuring the authenticity of the identity of communicating parties and the security of data transmitted.

Elastos is a complete micro-kernel based and Component oriented operating system, with flexible component assembly features. CAR (Component Assembly Runtime) technology is the component-oriented programming model in the Elastos, which is a set of programming specifications, including definition of the components, classes, objects, interfaces, and access requirements between component objects. By this technology, the binary components can be self-described, dynamically linked at runtime by Elastos, and dynamically assembled to provide more capacity.

There are an increasing number of applications and services on Elastos in the mobile internet environment, including such mobile e-commerce applications as electronic signatures and electronic contracts. They require the data transformed be encrypted to ensure confidentiality and the authentic identity of the parties. Besides, Elastos is demanded to be a secure and stable environment to provide run-time security checks.

To meet these security requirements, this paper presents the SD card based Elastos digital authentication mechanism to ensure the safe CAR runtime and provide other components or applications security services. And it meets security requirements, which mobile devices in the mobile internet demand the data integrity and the identity authenticity. In addition, employing the interoperability of Java and CAR technology, it offers security services to the different computing models, Android and Elastos.

In this paper, firstly, the security issues of data exchanged of mobile devices in the mobile internet environment, and domestic and abroad research on these issues are introduced. And then, Elastos, CAR technology, the interoperability of Java and CAR technology and security SD card related technology are analyzed. Moreover,

according to security SD card features, Elastos digital authentication mechanism is designed, and the relevant functional modules are achieved. Finally, the paper sums up and rewiews current work for digital authentication mechanism topic, and gived the outlook for future work.

Key Words: Elastos, CAR, Mobile Device, Security, Digital Authentication

目录

第1章 前言

1.1 背景介绍

移动互联网技术是指移动设备通过移动通信网络与互联网互联互通的技术，同时具有移动设备便携性及随时随地接入网络的特性和互联网的丰富资源，融合了两者的优势，使人类社会生活正经历着一场前所未有的全方位的深刻变革，它改变了传统的事务处理方式，对社会的进步和发展起着巨大的推动作用。

移动通信技术的发展，加上传输量更大、传输速度更快的电信系统以及第三代、第四代移动通信技术相继出现，以及移动通信网络与互联网深度融合，移动设备尤其是手机与外界的数据交换越来越频繁，数据交换量也越来越大，数据传输速度也越来越快，在这些设备上从事一些类似互联网上的活动成为趋势，比如移动电子政务、移动电子商务、安全信息交换和数据分享服务等领域。

移动电子商务是指人们在移动互联网环境下使用移动通信设备实现商务交易的活动，是在传统电子商务的基础上发展起来的[1]。传统电子商务主要通过桌面计算机进行，是有线形式的电子商务；而移动电子商务，则是可以随时随地通过手机等移动通信设备与互联网相结合而进行的电子商务活动。互联网技术、移动通信技术和其它技术的完善组合创造了移动电子商务。与原有电子商务技术相比较，移动电子商务具有灵活、简单、方便的特点，不受地域、设备等条件的制约，突破了原有电子商务技术的局限性。同时它也极大的改变了我们的商业模式，尤其对企业与客户之间的互动关系产生了巨大的影响。企业完全能根据客户的喜好需求提供给客户极其个性化的服务。客户也可以通过移动电子商务，随时随地获取所需的服务、应用、娱乐等信息。

移动设备与桌面计算机有很大的不同，具有较小功率的 CPU、较小的内存和显示屏及 CPU 计算能力弱的输入设备，是一种受限的计算环境。无线网络跟有线网络相比具有较低的带宽、较长的延时、较不稳定的连接和较大的不可预测性。移动电子商务跟传统电子商务相比，其安全性更加薄弱，传输承载的交换信息更易受到窃取和攻击。

移动互联网下进行各种商务活动最关键问题在于保证安全性，也决定着移动电子商务能否快速和持续发展[2]。除了与在互联网上进行电子商务活动遇到的相同的网络安全问题外，移动设备还面临着移动互联网带来的新的安全威胁，这些安全威胁来自很多方面，主要集中在数据传输链路、移动设备、应用系统、与电

子商务内网的数据交换等，为保护传输的敏感数据信息和系统安全，必须对这些数据进行强制安全检查。涉及到的安全检查包括能否有效阻止不法分子对交换数据的截取和攻击、如何防止信息灾难事故、如何进行数据完整性检查、如何进行身份鉴别认证、采用数字签名防止抵赖和篡改、如何保证交易数据的安全等。

1.2 课题的目的和意义

本课题提出了依靠 SD 智能卡的 Elastos 数字认证机制，目的在于解决 Elastos 运行时安全检查和网络传输的数据的完整性和身份认证。

本文所做的工作也是围绕着数字认证机制展开，首先提出和分析了依靠 SD 智能卡的 Elastos 数字认证机制，然后分析了该模型的具体设计需求，采用构件和分层的思想设计该认证机制。使用 CAR 构件技术将操作 PKCS#11 库模块封装成基础安全构件，命名为 ElCryptoCAR，为应用程序和其他 CAR 构件提供统一的调用接口。应用程序分为 Java 应用程序和 Elastos 普通应用程序，因此将基础安全构件采用 Java 与 CAR 互操作技术封装成 Java 层面接口，命令名为 ElCryptoJava。

本课题的研究具有重要的现实意义，基于这种数字认证机制的应用程序和 CAR 构件可以直接使用基础安全构件提供的安全功能服务，隐藏了安全 SD 智能卡底层实现细节，简化了开发过程，缩短了开发周期；将基础安全构件按照 Java 与 CAR 互操作技术封装出 Java 层面接口，使应用程序的类型扩展到了 Java 应用程序范围，改变了单纯依靠 Java 类库提供安全功能服务的状况，提高了 Java 应用程序在安全功能服务方面的效率；基于 CAR 构件思想有利于工程化开发和提高软件复用度；将安全构件、SD 智能卡和应用程序分离，有利于芯片提供商、智能卡和 USB 生产厂商、PKI 以及移动运营商在最短的时间内完成产品和应用的整合；在 SD 智能卡内完成密码运算，保障了密钥的安全性，加上物理防篡改技术的采用，伪造的难度和成本很高，大大提高了其安全性，此外不需要额外驱动程序就可以直接使用，具有良好的便携性。

1.3 国内外研究现状

在计算机网络安全中，身份认证的安全是整个网络安全的核心。在互联网领域，数字证书、USB KEY 的解决方案[3]已经被成熟地应用和推广。目前，许多科研单位也都在研究一些高强度加密算法，这些算法采用硬件加密卡或者纯软件的形式供开发者使用。安全模块的体系结构和函数接口也因开发商的不同而各不

相同，使应用程序的开发很困难，显然，也不能共用这些安全服务。这这结安全服务开发商做的是垂直的安全解决方案，包括从底层到最上层的应用程序，代码复用程序很低，几乎不可以得用复用，导致大量重复开发代码，降低了开发效率，增加研制周期长等问题。此外，协议定义在各安全开发商之间是不统一的，所以产品的标准化、可移植性、不同产品之间的互操作性是不可能实现的。另外随着应用安全需求增加，单一的密码服务已经不能够满足应用的安全保护要求[4]。

这就需要一个通用数据安全服务接口，具体安全应用提供商可专注于应用产品的开发，数据安全服务提供商也能专心研究和提供具有专业水准的安全服务，以此来促进各种各样的应用的开展。同时，很多厂商和科研机构也都意识到了这些问题，一致认为应重新设计一种新的安全基础体系，它能够在基础的密码、证书等服务之上定义出统一的接口，定义通用的运算规则和相应的协议，这种安全基础体系能提高代码复用性、缩短开发周期和提高安全性，能够为所有应用程序提供安全服务的基础结构。

目前，国外已经出现了一些满足这些设计需求的通用数据安全服务接口标准，例如 CAPI、CDSA、PKCS#11、SSAF 等，其中 CAPI 是由 Microsoft 公司开发的安全体系架构，它是 Windows 系统安全体系结构的一部分，其源代码是闭源的，而且也只能在 Windows 平台上使用，不具备跨平台特性。此外，该安全体系层次结构并不是很清晰，各模块之间耦合度比较高。CAPI 安全体系实现了目前常用的密码服务、数据存储服务和证书服务，但是并不支持新安全服务类，仅仅是上述三类基本服务，该安全体系也不能够全面支持。当然，CAPI 也有自身的优点，主要是它能够较好的支持模块之间的通讯，以及单独的控制模块检测防止模块的篡改[5]。

CDSA 安全体系结构是由 Intel 花费了四年的时间开发的一个安全体系架构规范，它是一个相对安全、健壮、可扩展的安全基础架构。能够较好地支持密码服务、数据存储服务、证书服务、可信策略和授权计算五类安全服务，但是框架代码体积比较大。五类安全服务中的可信策略和授权计算，在实际安全应用程序中不太常使用。此外，CDSA 的安全基础部件嵌入式完整性服务库 EISL，它的代码体积约 200K，所有的 CDSA 模块均载入一个 EISL 库，对于移动应用而言负担较重 [6] [7]。

PKCS#11 安全体系结构是由 RSA 公司开发的，专门为硬件加密而设计的标准，从它的体系结构上来看，该标准主要用于对硬件加密设备进行抽象[8]，对于硬件加密设备有很好的支持，该安全体系层次结构清晰。调用该加密操作的应用程序和真正的加密操作执行之间有 Cryptoki 库，所以说它的执行效率还是比较高的。此外，与其他几种安体体系结构相比，PKCS#11 代码体积很小，只有大约

700K，同时 PKCS#11 的所有源代码都是开源的，可以获得与它相关的所有信息。

SSAF 是美国国防信息管理局（DISA）在美国国防基础设施公共操作环境（DIICOE）中提出的一种安全服务体系结构框架，与 CAPI 一样，它的源代码也是闭源的。该安全体系主要包括密码服务、证书服务、数据存储服务和信任策略库，分别为这四类基本服务定义了应用程序调用 API 和 SPI 等接口。因为该框架设计目的的特殊性，设计了通用的国防安全应用层，它位于应用程序与框架管理层之间，而且在通用的国防安全应用层中增加了多类安全策略设计，使用得整个体系结构相对复杂，层次结构不清晰，各模块耦合度高。此外，因为是美国国防信息管理局设计的，所以开发了很多国防专用模块，对于普通的安全应用是没有什么用的[9]。

在传统的 SD 存储卡里，SD 控制芯片只能连接闪存，而没有办法再外接一个安全模块。在安全 SD 智能卡解决方案的存储卡里，全新设计的 SD 安全控制器可以同时连接闪存模块和安全模块，并且可以实现同步的读写访问。可以支持的安全模块包括各个厂商的智能卡芯片、Java 卡芯片或者是安全芯片。采用 Mass Storage Diver 作为驱动协议，这样在 Windows、WinCE、Linux、Symbian、Android 等系统下都可以直接使用，不用安装额外的驱动。针对各个不同智能卡厂商现有的 CSP、PKCS#11 甚至私有 API 库以及 COS，都不需要做复杂的改动，只需要按照专用的固件协议修改函数接口，就可以实现命令的转换和传输，并立即拥有自己的安全移动产品。

在国内，移动领域中的移动银行、移动炒股等移动业务相继出现，促使了基于智能卡的可应用于手机等移动设备的安全解决方案，比如捷德公司、中国华大、国民技术、上海华申等一些企业推出了自己相应的加密产品。其中捷德公司与中国金融认证中心（CFCA）、泰康移动通信发展有限公司合作推出了国内首例移动 PKI 试点工程，将 PKI 体系和 CA 认证中心概念引入无线商务领域，通过手机上的微型浏览器及 SIM 卡对数字签名的存储认证等功能的实现，确保各类移动交易安全性的解决方案。

1.4 论文的组织结构

本文主要介绍了基于安全 SD 卡的 Elastos 数字认证机制，以及相关的基本理论知识，并提出了设计目标和分析了它的体系结构，详细描述了其内部各个安全功能模块以及接口实现细节，最后对课题研究进行阶段性总结和对后续工作进行展望。

第一章前言部分介绍了课题研究的背景以及国内外的研究现状，并指出本课

题的目的和现实意义。

第二章介绍了相关技术理论内容，首先分析了 Elastos 操作系统、CAR 构件技术和 Java 与 CAR 互操作技术知识，然后介绍了 WPKI 概念以及相关技术，最后分析了本文的硬件基础安全 SD 卡技术。

第三章详细介绍了安全 SD 智能卡加密设备的实现技术基础 PKCS#11 规范，包括该规范的设计目的、通用模型、函数接口和安全机制。

第四章研究并设计了基于安全 SD 卡的 Elastos 数字认证机制的总体架构，然后详细分析了架构中的各个安全功能子模块，以及它们的功能和工作原理。

第五章研究和定义了数字认证机制的基础安全构件 ElCryptoCAR 构件接口和 ElCryptoJava 接口，包括密钥管理、证书管理、身份认证管理、安全传输管理和运行时安全检查管理。

第六章总结了本论文的研究工作和本人重点参与的工作，并对以后的研究工作提出了展望。

最后，是本文的一些参考文献和资料。

第2章 相关技术研究

本章主要介绍了 SD 智能卡数字认证相关的一些技术理论内容。首先介绍了 Elastos 和 CAR 构件技术，然后介绍了 Java 与 CAR 构件互操作技术，最后分析了 WPKI 相关技术和安全 SD 卡技术。

2.1 Elastos操作系统

Elastos 是构件化的网络嵌入式操作系统，具有多进程、多线程、抢占式、多优先级任务调度等特性[10]。它是由上海科泰世纪科技有限公司开发的拥有自主知识产权的操作系统。目前，Elastos 已经可以在包括 PC、ARM、MIPS 等多种体系架构上运行。Elastos 提供的功能模块全部基于 CAR（Component Assembly Runtime）构件技术，这是 Elastos 操作系统的精髓。CAR 构件技术规定了构件间相互调用的标准，每个 CAR 构件都包含自描述信息，可以在运行时动态裁剪组装。CAR 构件技术贯穿整个 Elastos 操作系统技术体系中。

从传统的操作系统体系结构的角度来看，Elastos 可以看成是由微内核、构件支持模块和系统服务三大部分组成的。

微内核主要可分为四大部分：硬件抽象层，对硬件的抽象描述，为该层之上的软件模块提供统一的接口；内存管理，规范化的内存管理接口，虚拟内存管理；任务管理，进程管理的基本支持，支持多进程，多线程；进程间通信，实现进程间通信的机制，是构件技术的基础设施。

构件支持模块提供了对 CAR 构件的支持，实现了构件运行环境。构件支持模块并不是独立于微内核单独存在的，微内核中的进程间通讯部分为其提供了必要的支持功能。

系统服务在 Elastos 中的系统服务都是以动态链接库的形式存在的。包括文件系统、设备驱动、网络支持等，它们是由系统服务器提供的。

此外，Elastos 中的功能模块全部基于 CAR 构件技术构建的，它们都是可动态加载和卸载的构件，应用系统按照需要剪裁组装，或者在运行时动态加载必要的构件，还可以替换已有模块。

Elastos 全面面向构件技术，这是它最大特点，也是它最大的优势，在操作系统层提供了对构件运行环境的支持；用构件技术实现了“灵活”的操作系统。

在新一代互联网应用中，越来越多的移动产品需要支持 Web 服务，而提供

Web 服务一定是基于构件的。在这种新的应用中，用户通过网络获取服务程序，这些服务程序必须是带有自描述信息的构件，然后，本地系统就可以从构件中读取自描述信息，为这个服务程序建立必需的运行时环境，最后自动加载运行。这是新一代互联网应用的需要，是必然的发展方向。Elastos 就是为满足这种需要而设计和开发的，首先在面向嵌入式系统应用的操作系统中实现了面向构件服务的技术。

面向构件、中间件的编程技术是实现 Web 服务的关键技术之一。Web 服务提供的软件服务是可执行的功能模块，可以说就是构件。在构件中包含着自描述信息，是对其功能的详细描述。Elastos 可以在整个网络环境下动态查找、动态链接并运行构件，使 Web 服务的运行有了支撑平台。

Elastos 的应用很广泛，可以应用在信息家电、工业控制、传统工业改造、国防、商业电子等领域，已经开发了 PDA/掌上电脑、数控机床、工业远程监控设备、医疗仪器等应用。

2.2 CAR构件技术

CAR（Component Assembly Runtime）构件技术是一个面向构件的编程模型，也可以说是一种编程思想，表现为一组编程规范，规定了构件间相互调用的标准，包括构件、类、对象、接口等定义与访问构件对象的规定，使得二进制构件能够自描述，能够在运行时动态链接[11]。

CAR 构件技术极大地简化了构件的开发过程，编写 CAR 文件后，通过 CAR 编译器的编译，就可以生成基本的代码框架，开发者就可以在这个基本代码框架上添加自己的具体功能代码，最终开发出特定功能的构件，提供给客户端使用。这种方式提高了构件开发的速度及质量。CAR 的编程思想是 Elastos 技术的精髓，它贯穿于整个 Elastos 技术体系的实现中。

Elastos 为 CAR 构件提供运行时支持，它的重要组成部分就是 CAR 构件运行时环境，包括虚拟机及构建在 Elastos 可移植层上的各种类库，在不考虑可移植性的情况下，开发者也可以利用本地技术构建本地类库，从而丰富系统的类库。

Elastos 构件运行平台提供了一整套符合 CAR 规范的系统服务构件和支持构件相关编程的 API 接口函数，实现并且支持原生系统构件和用户自定义构件互相调用的机制，为 CAR 构件提供了编程运行环境。不同操作系统上的 Elastos 构件运行平台的实现是有所不同的，只要符合 CAR 编程规范的应用程序，就可以通过该平台实现二进制级跨操作系统平台而兼容。在 Windows、WinCE、Linux 等其他操作系统上，Elastos 构件运行平台屏蔽了底层传统操作系统的具体特征，

实现了一个构件化的虚拟操作系统。在 Elastos 构件运行平台上开发的应用程序，可以不经修改、不损失太多效率、以相同的二进制代码形式，运行于传统操作系统之上。

CAR 具有以下技术特性：

（1）CAR 是一种基于构件的软件运行支持技术。构件运行支持能力直接决定所支持的构件的编写方法、结构设计，甚至算法选择。CAR 支持满足“故障独立性”（即某个部件崩溃不会引起其它部件的崩溃，这是硬件系统可靠性的基本特性）的运行环境，如过程、Domain 等。CAR 通过这种环境所提供的构件动态组装，以完成外部请求的预计的计算任务。

（2）CAR 是一种构件化的开发语言，它仅仅是负责框架部分，由 C/C++等编程语言实现具体的实现逻辑。CAR 所描述的框架部分以元数据的形式存在于构件的发行格式中，元数据通过反射（reflection）机制参与构件组装计算。框架是将具体的应用逻辑通过类似于 COM 的方式[13]（计数管理、接口查询、构件聚合）隐藏起来，并把自己暴露在外的最终运行封装。

（3）CAR 支持构件分布式布置在不同计算容器（进程、Domain、机器）中，从而实现分布式、协同计算。CAR 明确定义了构件在各种情况下的通信方式，故障处理方式、安全机制等。

（4）CAR 提供了构件的标准，不同的应用程序可以使用二进制构件，使软件构件真正能够成为“零件”，从而提高软件生产效率。

（5）CAR 定义了一种软件工程化方法，软件发行与配置策略，从而定义了面向服务的软件商业模型。

（6）CAR 支持灵活软件架构策略，通过类硬件的构件技术，方便各种软件架构下对构件的使用。

CAR 构件技术主要解决的问题包括不同来源的构件实现互操作，构件升级不会影响其他的构件，构件独立于编程语言，构件运行环境的透明性。

2.3 Java与CAR互操作技术

Java 与 CAR 互操作技术的运行时环境是 Java&CAR 虚拟机[12][14][15]，它是在 Android 原生 Dalvik 虚拟机的基础上，通过构建 CAR 对象并将其绑定到 Java 对象上、增加 Java 对象与 CAR 对象的相互访问功能。这种编程模型可以增强原来单一 Java 编程或是单一的 CAR 构件编程的能力，将 Java 的易开发和 CAR 构件的跨平台与运行高效率的优点结合起来。在保持原来的编程模型不变的前提下，增加了一种新的编程模型。一部分人可以只开发 CAR 构件，而另一部分人通过

Java 编程方式像使用普通 Java 对象一样来使用这些 CAR 构件，对这些人来说这些 CAR 构件是完全透明的。

Java 与 CAR 互操作技术的关键技术是在构造 Java 对象的时候同时构造所需要的 CAR 对象，在析构 Java 对象的时候同时析构 CAR 对象，从而保证两者的生命周期是一致的。此外，还需要保证 Java 对象与 CAR 对象的内部数据的一致性。图 2.1 表示的是 Java 对象 CAR 对象设计规则示意图，说明了两者之间的绑定关系。

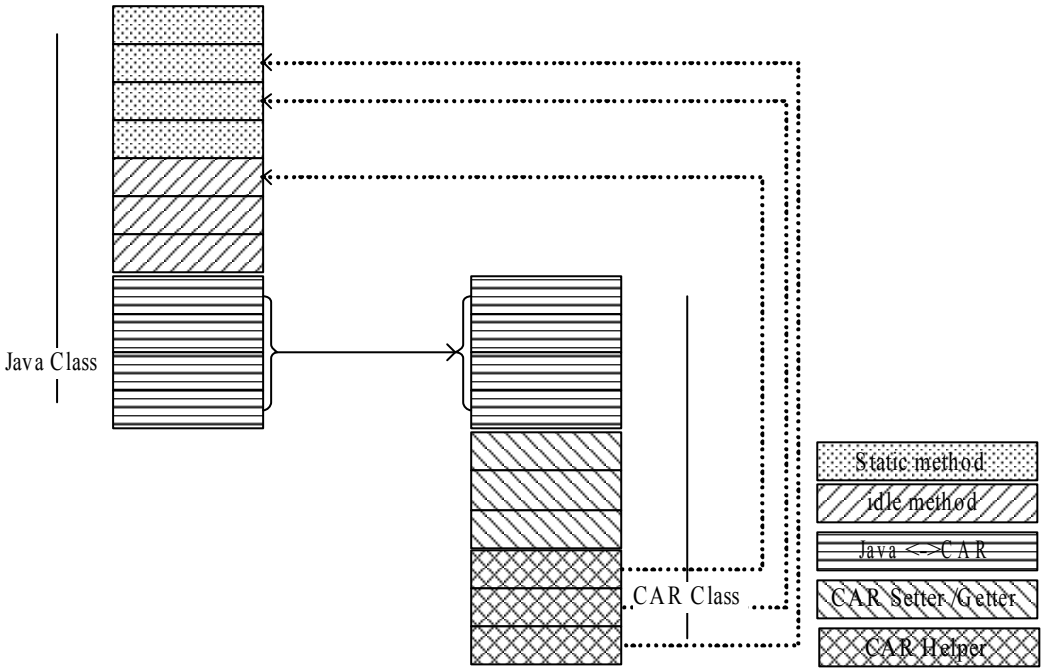

图 2.1 Java 对象 CAR 对象设计规则示意图

Java 对象的存在范围是 Java 虚拟机，而 CAR 对象的存在范围是 CAR 构件运行时，所以 Java&CAR 虚拟机包括两个必不可少的部分，一个是改造后的 Dalvik Java 虚拟机，一个是 CAR 构件运行时，即 Elastos Runtime。Java 对象如果要操作 CAR 对象时，当然 Java 对象是不知道 CAR 对象的类型的，就像操作普通 Java 对象一样。此时，Java 虚拟机会通过 ElastosAPI 进入到 Elastos Runtime 中，进而真正执行相应的操作。如果在 CAR 对象中操作 Java 对象时，通过 JNI 方式就可以[16] [17]。

图 2.2 表示的就是 Java&CAR 虚拟机结构图，其中包含调用关系示意。

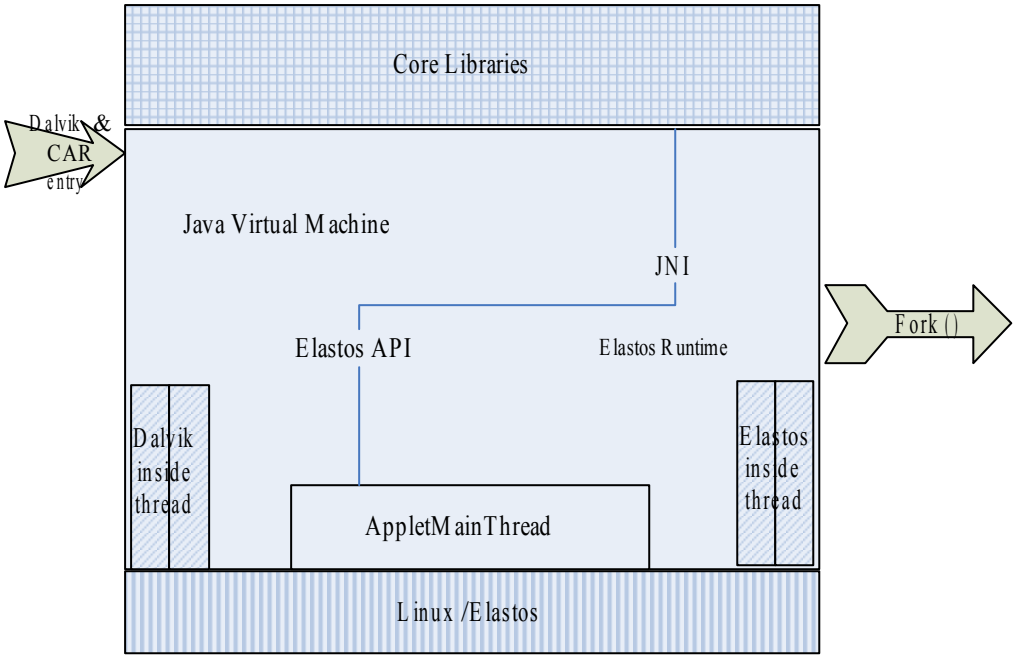

图 2.2 Java&CAR 虚拟机

2.4 WPKI技术

无线公钥基础设施（WPKI，Wireless Public Key Infrastructure）是把互联网电子商务中 PKI（Public Key Infrastrcture）安全机制引入到无线网络环境中的一套遵循既定标准的密钥及证书管理平台体系[19][20]，这样，在移动网络环境中使用的公钥和数字证书都可以通过 WPKI 来管理的，从而建立比较安全的和可以值得信任的无线网络环境。

WPKI 仅仅是将传统的 PKI 技术扩展到无线环境中，因此它实际上并不是一个全新的 PKI 标准。WPKI 中采用了优化的 ECC 椭圆曲线加密和压缩的 X.509 数字证书[20]，采用数字证书来管理公钥，通过认证中心验证用户的身份的真实性，从而达到安全传输数据信息。这个认证中心一般是由专业的第三方可信任机构承担。

WPKI 的系统架构图如图 2.3 所示，包含各个机构，分别担当着不多的功能角色，基于移动网络的各类移动终端用户和移动数据服务提供商的业务系统需要安全保证，而 WPKI 就是为它们提供基于 WPKI 体系的各种安全服务。

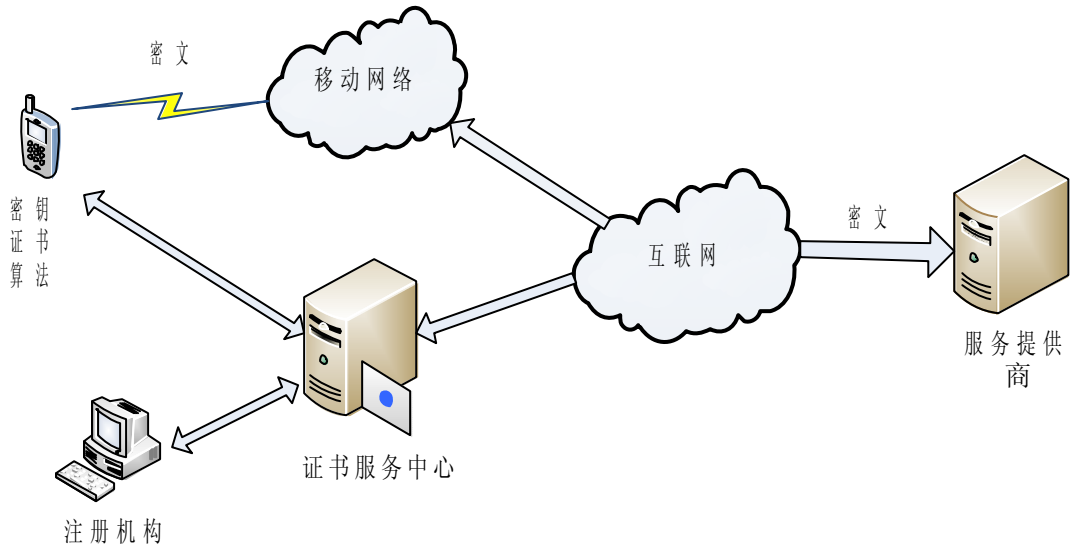

图 2.3 WPKI 系统架构图

从 WPKI 系统架构示意图中，可以看到无线终端通过注册机构向证书服务中心申请数字证书，然后，证书服务中心必须经过严格地审核用户身份，通过用户身份审核以后，就可以给用户签发数字证书，并传送给用户。之后，用户可以把私钥和数字证书存储在某一存储设备中，一般存储在 SIM 或是安全 SD 卡中，为了安全保证，通常中将它们加密以后再存储起来。无线终端在无线移动网络上进行电子商务操作时，便可以使用数字证书保证通信双方的真实身份性，以达到安全。服务提供商为了确定用户的真实身份，需要验证其数字证书，通过验证后就可以为用户提供相应的服务，实现电子商务在无线移动网络上的安全进行[23]。

完整的 WPKI 系统需要包括权威证书签发机构(CA)、数字证书库、证书作废系统、密钥备份及恢复系统、应用接口等基本部分[25]，所以需要围绕着这五大基本部分组建整个系统。

证书签发机构是数字证书的申请、审核和签发机构，所以它必须具备权威性的特征。数字证书库用来保存已签发的数字证书及公钥，如果其他用户需要另外用户的数字证书和公钥，就可以通过访问数字证书库来获取。证书作废系统是 WPKI 系统中必须部分。就像我们日常生活中的身份证，数字证书在它有效期内由于某种原因，也是有可能需要作废的，比如说密钥存储设备的遗失或用户身份变更等等。密钥备份及恢复系统是为了备份和应急恢复而专门设立的基本部分，是 WPKI 提供备份与恢复密钥的机制，避免用户丢失解密数据的密钥。当然，由于密钥的保密级别很高，安全要求也很高，必须由可信的机构来负责密钥的备份与恢复工作，而且，密钥备份与恢复只能针对解密密钥，签名私钥为确保其唯一性而不能够作备份。应用接口是 WPKI 系统为方便第三方应用开发而设计的，使

各种各样的应用能够与 WPKI 以安全、一致和可信的方式交互，确保安全网络环境的完整性和易用性。

随着移动互联网的深入发展，移动设备通过移动无线网络上网成为人们上网的最主要的方式，无线通信技术在银行、证券、商务、贸易、办公、教育等各方面的需求越来越多，无线通信的安全性也显得日益重要，因而 WPKI 技术也渐渐发展起来，它为解决移动环境和无线环境下的安全认证和支付奠定了基础[26]。

2.5 安全SD智能卡技术

2.5.1 安全 SD 卡结构

安全 SD 卡的内部结构与普通 SD 卡的结构是不同的，安全 SD 卡中的安全控制器可以同时连接闪存模块和安全模块，并且可以实现同步的读写访问[27]。可以支持的安全模块包括各个厂商的智能卡芯片、Java 卡芯片或者是安全芯片。其结构图如图 4.2 所示。

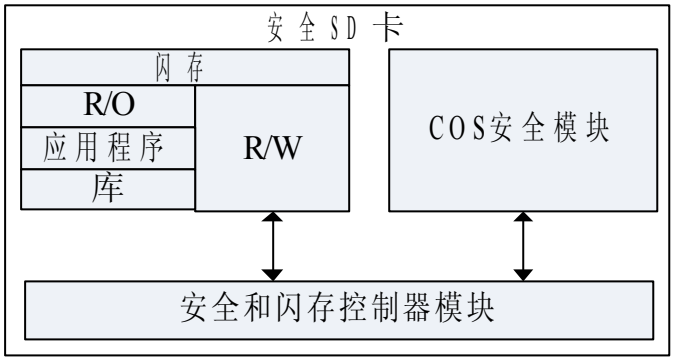

图 2.4 安全 SD 卡架构示意图

采用 Mass Storage Diver 作为驱动协议，在 Windows、Linux、WinCE、Symbian、Android 等系统下都可直接使用，不再需要安装额外的驱动。针对各个不同智能卡厂商现有的 CSP、PKCS#11 或者私有 API 函数库以及 COS（Chip Operating System），都不需要做复杂的改动，只需要按照专用的固件协议修改函数接口，就可以实现命令的转换和传输，并立即拥有自己的安全移动产品。

安全 SD 卡的闪存中的只读区域可以放置安全中间件、应用程序，当 SD 卡插入移动设备的 SD 卡插槽中可以实现自动安装的功能。通过安全 SD 卡的算法芯片可以实现对闪存数据的高安全自动加解密功能。

安全 SD 卡具有很多特点[29]，例如：体积小而且轻，便于携带；存储容量大，防磁、防静电、抗干扰能力强，可靠性高。信息可读写数万次，使用寿命长；保

密性强，安全性高。控制芯片完全控制存储区的读写，加密密钥存放在智能卡芯片中，确保密钥的安全；支持逻辑分区，并且每个逻辑分区使用独立的密码，可以分别设置读、写控制口令；闪存芯片上的数据全部加密存储，且加密密钥与加密运算都在 IC 芯片中进行；支持多种平台的安全应用；支持 Micro SD 接口；支持一次一密的数据交互方式。

可以广泛应用在基于不同移动终端平台的数据加密、密钥保护和证书管理等安全领域。基于 SD Key 产品可以支持多种平台的安全应用，包括桌面、手机、PDA 操作系统。

2.5.2 安全 SD 卡操作系统

COS(Chip Operating System)即芯片操作系统[30]，一般指智能卡操作系统。COS 在智能卡中类似于 PC 中的 Windows、Unix 等操作系统，但 COS 是一个专用系统，而不是通用系统，COS 主要是如何进行处理和响应外部命令，而不需要涉及共享、并发的管理和处理等复杂的通用操作系统问题。

COS 的主要功能是控制智能卡和外界的信息交换，管理卡内的存储器，并在卡内部完成各种命令的处理。安全控制是 COS 的核心，也是衡量 COS 性能的一个重要部分。COS 级的系统安全主要表现在以下两个方面：一是 COS 对卡内数据的安全保障能力，二是 COS 对外提供的安全定义，即应用程序能够利用 COS 提供的安全机制自定义出其卡内数据的访问控制。

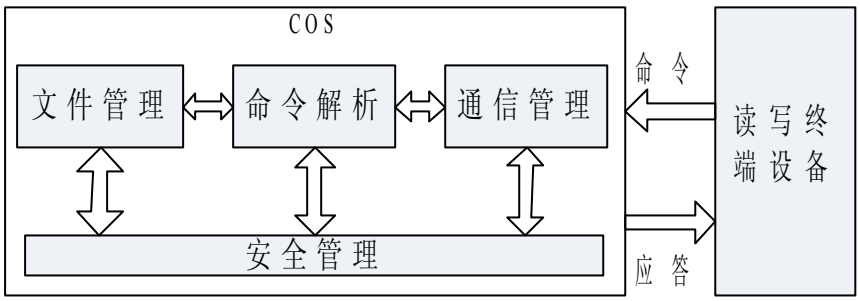

图 4.13 COS 功能模块图

COS 主要包括四个功能模块，分别是传输管理、文件管理、安全管理、命令应答[25]。物理层上的传输信号经过解码传递给安全管理模块，如果是线路级加密传输，安全管理模块将进行解密命令，然后将解密后的命令传送给命令管理应答模块。如果线路没有加密命令，那么就直接把命令传送给命令应答模块，命令应答模块执行命令要求的操作。当涉及到数据信息储存文件管理时，文件管理模块会检查命令是否具有足够的文件操作权限，如果具有足够的权限，那么执行命

令要求的相应操作，这些操作执行完成之后就发送相应的响应信息。

开发 COS 必须遵循一些国际标准，包括 ISO/IEC 7816-1/2/3/4 标准系列、CEN、EMV、ETSI、SET、C-SET 等。其中，ISO/IEC 7816 标准[28]的第一部分定义塑料基片的物理和尺寸特性，第二部分定义触点的尺寸和位置，第三部分定义信息交换的底层协议描述，第四部分论述跨行业的命令集。

第3章 PKCS#11规范概述

3.1 PKCS#11的设计目的

PKCS（Public Key Cryptography Standards）是公钥密码标准，它是由美国RSA数据安全公司专门为硬件密码设备提供的一个工业标准接口[8]。PKCS标准包括一系列标准，其中，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）的标准编程接口，它可以用于各种可移植的密码设备。Cryptoki提出了一个通用逻辑模型，安全应用开发者可以在不知道底层硬件操作细节的情况下，利用可移植设备完成加密解密等安全相关服务的操作，从而可以快速构建应用。

PKCS#11标准主要有两个目的[31]：第一个目的就是屏蔽不同的安全实现提供商在底层安全服务实现上的不同，为安全应用程序开发者提供出统一的接口，使不同这始实现提供商的具体实现方法相对于开发者或用户来说是透明的；第二个目的就是使安全应用开发者或者用户，能够在多个安全服务具体实现之间共享资源，让一个应用程序能够访问多个设备，也能够使一个设备可以被多个应用程序访问。

3.2 PKCS#11的通用模型

PKCS#11标准规范的通用模型如图3.1所示。通用模型开始于一个或多个必须执行某些密码安全操作的应用程序，结束于一个或多个密码设备，也就是说在密码设备上执行某些或全部加密解密等安全操作。

从通用模型结构图中可以看出，PKCS#11包含着会话（Session）、槽（Slot）、令牌（Token）、对象（Object）组成部分，用于抽象对各种安全设备的使用过程。其中，会话抽象地表示应用到设备的连接，槽抽象地表示读卡器等固定安全设备，令牌抽象地表示智能卡等与个人身份相关的安全设备，对象抽象地表示在安全操作中使用和产生的各种数据和结构。该模型的最上层是安全相关应用程序，它需要执行各种加密操作，在模型的最下层则分别对应各种加密设备。

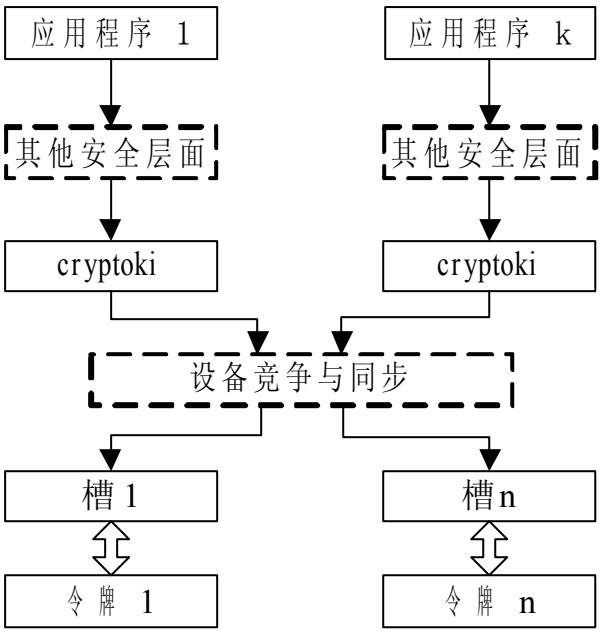

图 3.1 Cryptoki 通用模型

3.2.1 槽和令牌

PKCS#11中定义了一系列的通过槽访问加密设备的函数接口。槽和令牌是其中很重的概念。每个槽对应于系统中的物理读卡器，或者是其他类似的物理接口。当加密设备，例如SD卡、TF卡等智能卡放入到读卡器中时，槽中就出现对应的令牌。可以说，令牌对应于像智能卡一类的加密设备。因为槽和令牌只是逻辑上的概念，所以它们也不一定与读卡器和智能卡一一对应，有时也可能对应于其他加密设备，也有可能是纯软件模块模拟出来的。应用程序可以通过系统中的若干个槽访问令牌，然后完成所需要的安全加密任务。

加密设备如果完成所需要的安全加密任务，就需要发送特定的安全命令。这些命令一般是通过一些固定的设备驱动发送给加密设备的。PKCS#11的设计目的是为了向用户提供统一的函数接口，屏蔽各种加密设备的具体的底层技术实现的差异，让它们在逻辑上是一样的。所以，应用程序就不再直接面对加密设备的接口，而且也不再需要知道系统中存在着哪些加密设备，而是通过这种统一的接口来调用加密设备的功能。在具体实现时，可以PKCS#11实现为支持接口功能的动态链接库，在使用时，应用程序就可以链接到这些动态链接库上，从而调用真正的安全加密服务的功能。

3.2.2 对象

在PKCS#11中，令牌主要用来存储对象和执行安全任务。按照对象中存储的内容的不同进行分类，PKCS#11标准中总共定义了数据对象、证书对象和密钥对象[8]三类对象，应用程序定义数据对象，证书对象用于存储各种证书，密钥对象用于存储各种密钥，它还包括私钥、公钥、保密密钥三种类型。对象的层次结构图如图3.2所示。

当然，也可以不按对象存储的内容分类，而是按其生命期和可用性分类，那么对象分为令牌对象和会话对象两类。如果按访问权限来分类，那么对象分为公共对象和私有对象。

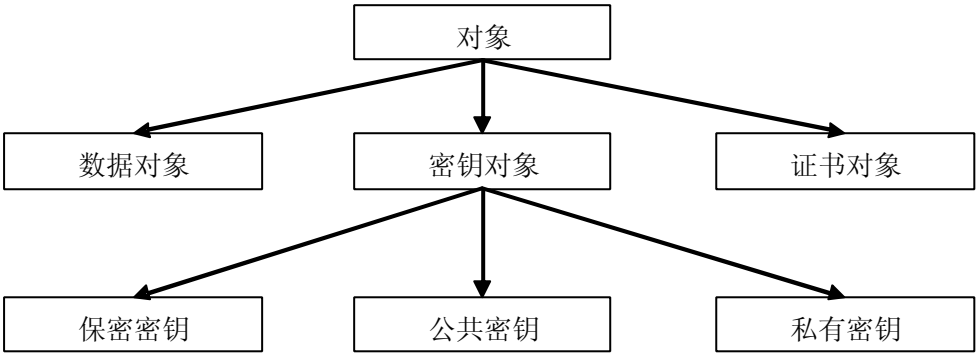

图 3.2 对象的层次结构

3.2.3 会话

当应用程序需要使用Cryptoki的安全服务时，首先在令牌上建立一个或多个会话，然后才能访问令牌上的对象和安全功能，其中，会话是应用程序和令牌之间的逻辑连接。但是如果仅是建立了会话，还不能执行令牌提供的加密操作，必须在建立会话之后在使用加密等安全操作之前登陆为普通用户。Cryptoki定义了两种用户，安全官员（Security Officer）和普通用户（User）。安全官员用来对令牌进行初始化和设置普通用户的PIN码，此外，它还可以操作公有对象。只有普通用户可以访问私有对象、执行令牌提供的加密操作。普通用户也只有在安全官员设置完毕其PIN码后才能进行登陆。可以看出，安全官员和普通用户有可能是同一个人，也有可能是不同的人。

如果将会话细分的话，可分为只读会话和读写会话两种类型。这里所指的只读和读写都是针对令牌对象而言，而不是会话对象。这是因为两种会话都可以创建、读取、写入和销毁会话对象和读取令牌对象，而只有使用读写会话的应用程序才能创建、修改和销毁令牌对象。

会话包含只读会话状态和读写会话状态，这些状态决定了会话对象的访问权限和在其上进行的加密操作。只读会话有两种状态，当没有用户登陆的时候是只

读公共会话状态（R/O公共会话），当有普通用户登陆的时候是只读用户会话状态（R/O用户会话），不存在只读安全官员功能状态，其状态转换图如图3.3所示。

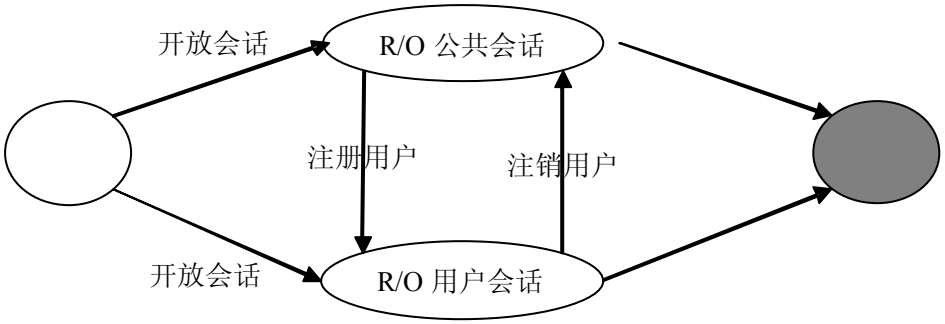

图 3.3 只读会话状态转换图

只读公共会话状态和只读用户会话状态的含义如表3.1所示。

表 3.1 只读会话的状态说明

状态	描述
R/O公共会话	应用程序已打开一个只读会话。应用程序对公共令牌对象进行只读访问，对公共会话对象进行读写访问。
R/O用户会话	普通用户由令牌鉴别。应用程序对所有的令牌对象（公共和私有）进行访问，对所有会话对象（公共和私有）进行访问。

读写会话有三种状态，分别是读写公有会话状态（R/W公共会话状态）、读写用户会话状态（R/W用户会话状态）和读写安全官员会话状态（R/W SO会话状态）。其状态转换图如图3.4所示：

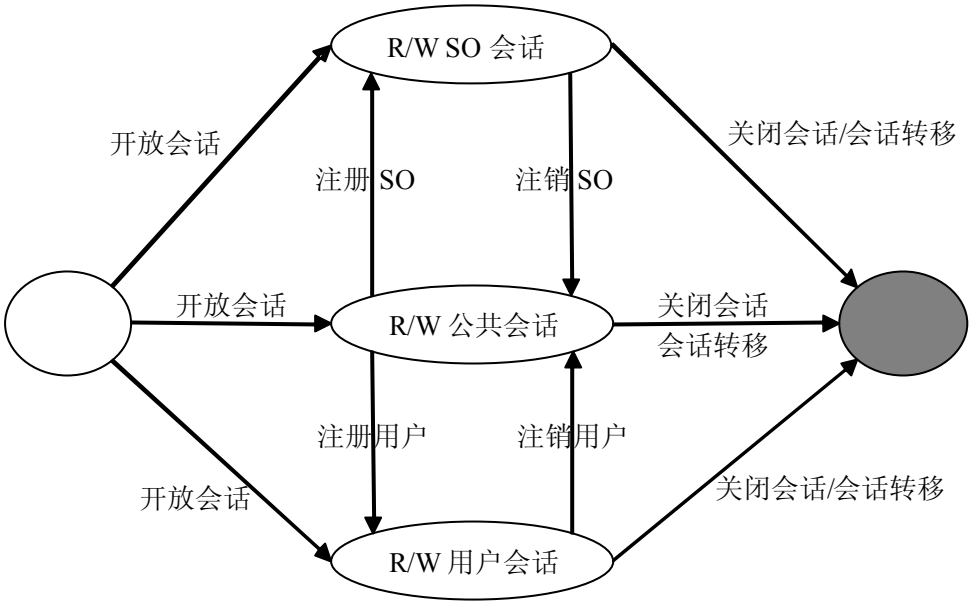

图 3.4 读写会话状态转换图

读写公共会话状态、读写官员会话状态和读写用户会话状态的含义如表3.2所示。

表 3.2 读写会话的状态说明

状态	描述
R/W公共会话	应用程序已经打开一个读/写会话。应用程序对所有公共对象进行读/写访问。
R/W SO会话	安全执行官已经由令牌授权。应用程序只能对令牌上的公共对象进行访问。SO能设置普通用户的PIN。
R/W用户会话	普通用户已经由令牌授权。应用程序对所有对象进行读/写访问。

Cryptoki中允许一个应用程序在多个令牌上打开多个会话，但是在同一个令牌上的会话必须具有同样的状态。例如，在一个令牌上同时有只读公有会话状态和读写公有会话状态的会话，当有一个会话以普通用户的身份登陆，那么所有的会话都将转变成用户会话状态。同样的情况下，不能在某会话上登陆为安全官员，即使是在读写公有会话状态的会话上也不能登陆，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。

3.3 PKCS#11的函数接口

PKCS#11规范中的函数接口很丰富，在规范中定义了很多标准函数，包括通用函数、槽和令牌管理函数、会话管理函数、对象管理函数、加密函数、解密函数、消息摘要函数、签名和MAC函数、验证函数、双效功能函数（即签名加密、解密验证等函数）、密钥管理函数、并行操作管理函数等。

3.4 PKCS#11的安全机制

PKCS#11是针对硬件加密设备抽象出的开发标准，只是规定了密码服务相关函数接口，而没有定义这些函数是如何实现的。PKCS#11的安全机制，分类区分对象、用户和会话以及它们相应的访问权限，此外，对密钥进行属性设置，来实现安全保护[32]。也就是设置私钥对象和对称密钥对象的属性，有SENSITIVE、INSENSITIVE、UNEXTRACTABLE和UNEXTRACTABLE四个属性值，对它们进行保护。其中，属性为SENSITIVE的密钥对象只可以在用秘密密钥加密后以密文的形式导出令牌，不能以明文方式导出。属性为UNEXTRACTABLE的密钥对

象则不能导出令牌。

通过PKCS#11通用框架，应用程序可以访问令牌上私有对象、证书或者密码功能函数，但是必须提供PIN码，让令牌验证使用者的身份[24]。只有在通过令牌的PIN验证后，才能进一步使用密码功能函数。

为了使令牌能够保存上层应用程序管理的密钥对象，必须保证安全存储。如果某加密设备没有受保护的内存存储单元储存私有对象和敏感的数据对象，那么它就可以利用用户的PIN码产生一个密钥，然后使用这个密钥对要存储的数据进行加密，再存储这些加密的对象[33]。

密码学只是提供安全的一种因素，而令牌也只是相应信息系统中的一个组成部分。当令牌本身具备了安全性时，下一步就应该考虑操作系统的安全问题，尤其是当PIN码的传输通过操作系统的时候。否则，恶意程序就会通过操作系统的安全漏洞获取用户的PIN码。此外，恶意程序也可能监听安全设备的信息传输通信线路，从而泄露PIN码的。

具有SENSITIVE属性的密钥对象不会遭到破坏，因为只要该密钥对象的属性设置为SENSITIVE，它将一直保持这个状态，而不会改变。同样的，只要某密钥对象的属性设置为UNEXTRACTABLE，它也将一直保持这个状态。

用户登录到令牌后，PKCS#11将不会限制该用户对密码功能的使用，该用户将能够执行该令牌支持的所有操作，所以需要有效地保护PIN码。

第4章 Elastos数字认证机制研究

本章详细分析和讨论了基于安全SD卡的Elastos数字认证机制（ElPKI），明确指出了其设计目标，并根据该设计目标设计了数字认证机制的总体架构，然后详细分析了各个组成部分。

4.1 设计目标

结合移动设备和 SD 卡加密设备的特点，为满足移动互联网业务的安全需求，本文采用基于安全 SD 卡 Elastos 数字认证机制，该信息安全解决方案可以提供 CAR 构件运行时安全检查，此外，还提供两个方面的身份认证：一是实体认证，即防止信息的发送者的身份的冒充；另一个是消息认证，即验证和鉴别信息的完整性，确保数据在发送的过程中没有被篡改。它的总体设计目标为：

（1）以 WPKI 为基础，使用数字证书作为通信双方的身份，网上身份认证的依据和基础。

（2）根据应用程序和 CAR 构件层次的不同，分别设计不同层次的接口。

（3）通过结合标准认证协议和数字签名的认证系统来确认用户身份，并采用三重身份认证机制，同时提供数据的完整性和不可否认性。

（4）采用数据加密技术来保护信息的机密性。

（5）安全应用程序、安全构件和密码算法库相分离。可以根据应用系统的不同安全需求很方便地实现定制化。

（6）认证机制应该满足 Elastos 和 Android 两种计算模型环境。

4.2 总体架构

基于安全 SD 卡的 Elastos 进行数字认证机制，通过该机制来验证其系统运行时安全，并以安全构件的形式向上层应用程序提供安全服务支持，完成了移动设备在移动互联网范围内对传输的数据完整性和身份真实性的认证，

向上为应用系统提供开发接口，向下提供统一的接口。图 4.1 描述了基于 SD 卡的 Elastos 数字认证机制的总体架构。

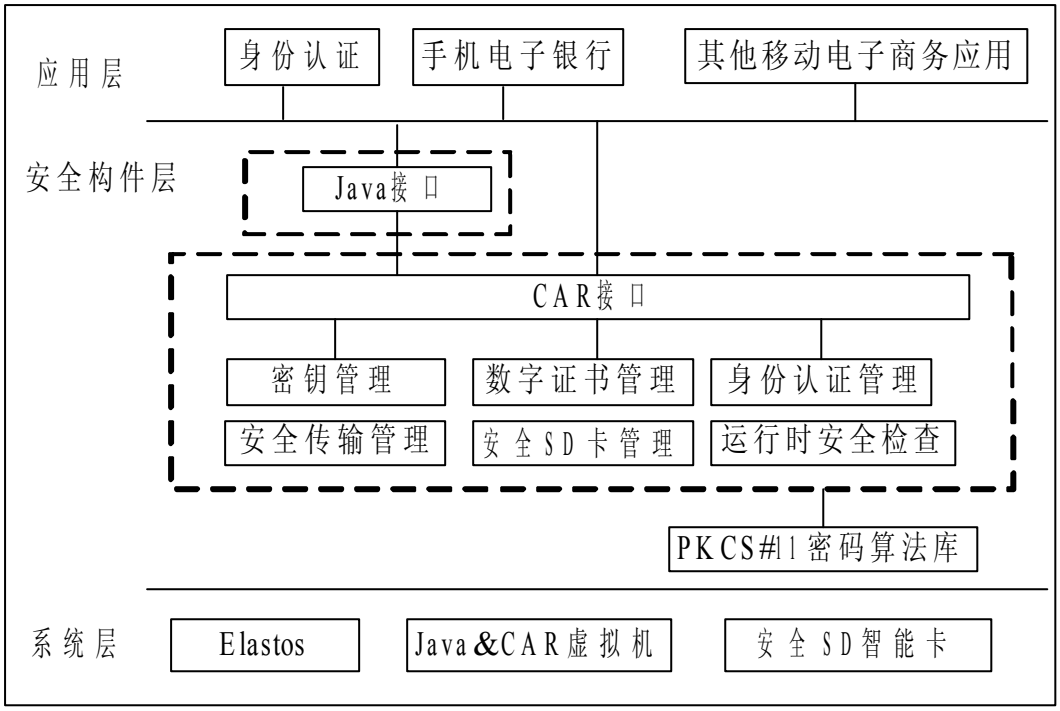

图 4.1 数字认证机制总体架构图

结合安全 SD 智能卡的 Elastos 数字认证机制，主要解决了 Elastos 运行基础 CAR 构件运行时的安全检查，以及在其上传输数据的安全性保证和为上层应用程序提供安全服务接口。它设计为以下几部分：

（1）密钥管理模块

密钥管理模块管理的主要是对称密钥和非对称密钥[34]。把密钥对象句柄传递给用户，通过密钥对象句柄进行相关操作；用户使用密钥对象时，也只是传入密钥对象句柄，而不把实际的密钥传送给用户，对用户而言，密钥对象是透明的，从而保证密钥对象的安全。

（2）数字证书管理模块

数字证书管理模块主要是管理本地数字证书的申请、获取、存储、作废和更新。数字证书是用来标识移动电子商务中企业或个人的有效身份，是标志网络用户身份信息的一系列数据，用来在网络通信中识别和验证通信双方的身份，也就是在互联网中解决“我是谁”的问题，就像在现实生活中，每一个人需要有一张身份证或者驾驶执照，用来表明我们的身份或者某种资格。数字证书是由权威公正的第三方机构签发的，这个机制就是 CA 中心，以数字证书为核心的加密技术可以对网络上传输的信息进行加密和解密、数字签名和签名验证，保证网络中传输的信息的保密性、完整性，以及交易实体身份的真实性，签名信息的不可否认性，从而保障网络应用的安全性。

（3）身份认证管理模块

身份认证模块在电子商务中的作用非常重要，无论是企业还是个人，在办理任何业务之前都需要首先认证身份。数字证书是标识身份的基础，数字签名技术的运用能够鉴别出合法的证书，而足够安全的认证协议是提高安全性、防止攻击的重要保证。身份认证模块主要包括消息摘要、数字签名和身份认证协议三个部分。

（4）安全传输管理模块

安全传输管理模块主要是用来保证通信双方的传输通道的安全性，确保在通信的过程不被其他非法用户获取。当然，在安全通道中传输的数据，也不是以明文的形式传输的，而是以密文的形式传输的，双重保证数据的安全。建立安全通道的协议采用 SSL 安全连接。此外，为了方便数据的打包和解包，采用 XML（eXtensible Markup Language）格式化存储。

（5）运行时安全检查模块

运行安全检查的目的是保证 CAR 构件运行时环境的安全性，检查 Elastos 系统中原生 CAR 构件，尤其是安全构件，和参与本地计算的第三方 CAR 构件的真实身份和数据完整性，检查 CAR 构件之间请求是否具有足够的权限。

4.3 密钥管理

密钥管理主要管理的是对称密钥和非对称密钥。把密钥对象句柄传递给用户，通过密钥对象句柄进行相关操作；用户使用密钥对象时，也只是传入密钥对象句柄，而不把实际的密钥传送给用户，保证密钥对象的安全。

（1）对称密钥管理

密钥管理原则：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创建的对称密钥对象以保证生命周期的一致性。

通过 Cryptoki 密钥库生成对称密钥，它们属性可以指定为会话对象（Session Object）类别，不具备持久性，所以不需要将它们保存在 SD 卡加密设备中。由于对称密钥的属性是会话对象，所以将生成的对称密钥句柄和对称密钥对象根据该对称密钥句柄的唯一性插入到与它相关的会话中，即插入到会话对象结构中的对象链表里，方便使用和删除密钥，提高密钥数据的安全性。对称密钥管理示意图如图 4.2 所示。

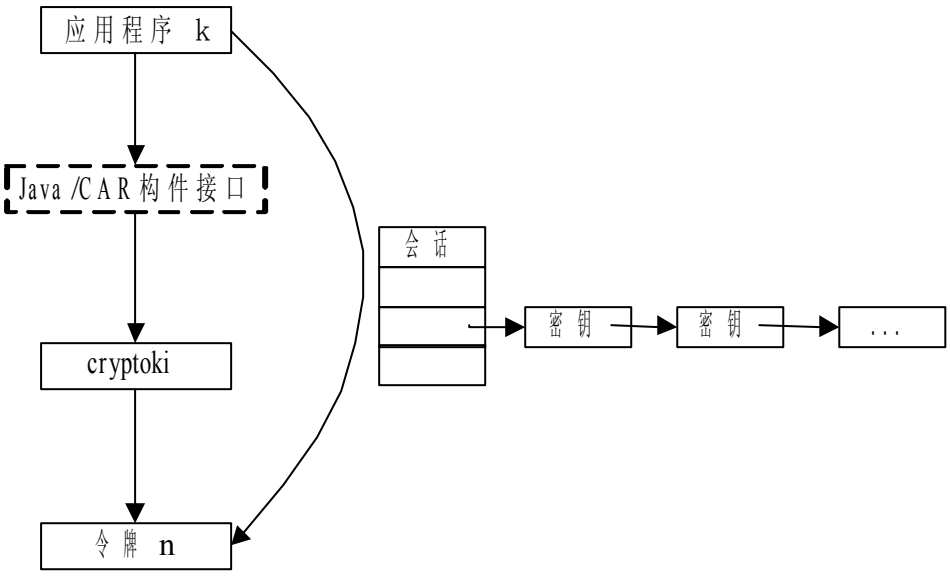

图 4.2 对称密钥管理

在安全加密处理过程中，通过对称密钥句柄访问到对称密钥，然后用它来对消息明文进行加密，使用完成之后，及时删除不再使用的密钥，释放相应的系统资源。此外，在关闭会话时，也要检查与当前会话关连的密钥对象是否还存在，如果还存在未删除的密钥对象，则清理完这些对象以后再关闭会话。否则会因为会话已经结束，而密钥对象的存在引起内存泄漏，造成系统安全隐患。

（2）非对称密钥管理

密钥管理原则：非对称密钥对象的类别不同区别对待；非对称密钥对象的生命周期是与其类别相关。

通过 Cryptoki 密钥库生成非对称密钥，它们属性可以指定为会话对象（Session Object）类别或者令牌对象（Token Object）类别

当非对称密钥属性指定为会话对象类别时，其生命周期与会话对象一致，所以其管理方式和对称密钥管理是一样的。

当非对称密钥属性指定为令牌对象类别时，具有持久性，需要保存在 SD 智能卡存储区内。密钥对象的 ElPKIA_ID 属性用来唯一标识不同的密钥对象，其 ElPKIA_ID 属性值和 ElPKIA_SUBJECT 属性值在一对公钥私钥中都应保持相同的值，而且跟其对应的证书的这些属性值是相同的。所以，为了有效地区分所保存密钥，把非对称密钥对象的 ElPKIA_ID 属性值和 ElPKIA_SUBJECT 属性值和密钥数据内容保存在其中即可，有利于使用、查找和删除密钥。非对称密钥管理示意图如图 4.3 所示。

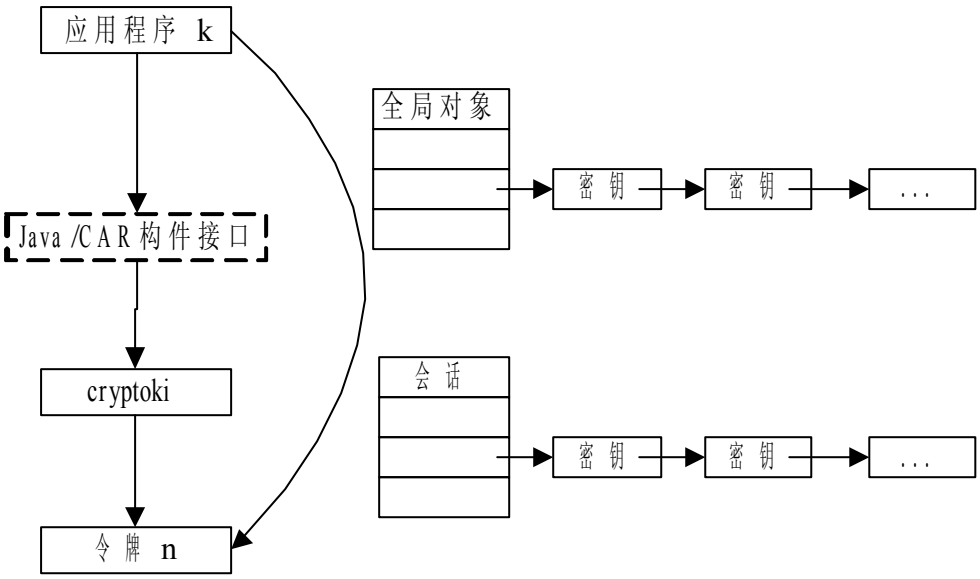

图 4.3 非对称密钥管理

在与令牌建立会话时，从 SD 智能卡中读出非对称密钥，并将它们以对象的形式存放到全局对象链表中。

当使用非对称密钥时，调用对象查找函数进行查找，参数则是唯一标识密钥对象的 ElPKIA_ID 或 ElPKIA_SUBJECT 属性值，获得密钥对象句柄后就可以使用了。

同对称密钥管理一样，通过密钥句柄访问到密钥，进行安全加密、数字签名等安全服务。在使用完之后，及时删除不再使用的密钥对象，释放相应的系统资源。此外，在系统退出时，也要检查全局对象链表中的是否还存在未释放的密钥对象，如果还存在未释放的密钥对象，那么在清理完成之后再关闭会话退出系统。否则，同样会因密钥对象的存在引起内存泄漏，造成系统安全隐患。

4.4 数字证书管理

数字证书就像人们在现生活中的身份证，它是用户的一种身份证明，它包含了用户身份信息、用户公钥信息以及有效的数字签名的数据等信息。

数字证书是由权威机构签发的数字信息，它能够对网络中传递的信息进行加密、解密、数字签名和对签证的真实性进行验证，保证网络中的信息在传递的过程中是秘密的、完整的、真实的并不能够被任意篡改和破坏，为网络的安全提供全方位的保障。

通常，为特定目的、为特定持续时间并经常为特定接收方将证书颁发给特定计算机、用户或服务。因此，在需要获取其他证书时，请续订现有证书，检查或

修改证书的属性，或者移动证书。尽管可以为单个用户、计算机或服务执行许多上述任务，但其他任务最好由管理员从服务器自动执行。本节描述了数字证书管理任务以及完成这些任务的过程。

4.4.1 数字证书的格式

数字证书是由权威公正的第三方机构即 CA 中心签发的，以数字证书为核心的加密技术可以对网络上传输的数据信息进行加密和解密、数字签名和签名验证，确保网络上传递的数据信息的机密性、完整性，以及交易实体身份的真实性，签名信息的不可否认性，从而保障网络应用的安全性。

数字证书根据使用范围和目的的不同有着各式各样格式和内容，目前最广泛使用的是 X.509v3 证书[21]。X.509 证书经历了三个版本，而 v3 版是 v1 和 v2 版的升级，它通过添加扩展字段扩充了 v2 格式，避免了 v1 和 v2 版中为了回应新的需求和克服存在的不足问题。X.509 格式证书已经在电子邮件安全的 S/MIME 中、网络安全支付系统的 SET 中得到了广泛的使用，X.509 数字证书的格式如图 4.4 所示。

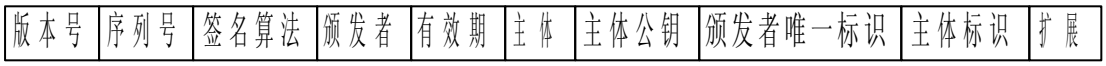

图 4.4 X.509 数字证书结构图

数字证书结构中的各个信息包含着证书的信息、证书颁发者信息、证书持有者信息，其具体含义如下：

（1）版本号：标识证书的版本（版本 1，版本 2，版本 3）。

（2）序列号：由 CA 赋予每个证书一个特殊整数编号，是证书的唯一标识。

（3）签名算法：签名算法标识符，用于说明本证书所用的签名算法。例如 SHA-1 和 RSA 的对象标识符就是用来说明该数字签名是利用 RSA 对 SHA-1 杂凑加密签名。

（4）颁发者（CA）：CA 身份名的可识别名（DN）。

（5）有效期：从起始时间到截止时间。本字段由“Not Valid Before”和“Not Valid After”两项组成。现在通用的证书一般采用 UTC 时间格式，它表示的时间范围为 1950 到 2049。

（6）主题：证书文件拥有者的可识别名（DN）。

（7）主题公钥信息：证书所有人的公开密钥和算法识别符，包括公钥算法、公钥的位字符串表示。

（8）颁发者唯一标识：颁发者的唯一标识，属于可选项。

（9）主题唯一标识：证书拥有者的唯一标识，属于可选项。

（10）扩展是构建证书的可选的标准和专用扩展，包括证书策略、策略映射、密钥标识符、主题及颁发者的别名和主题目录属性等。

4.4.2 数字证书的工作原理

数字证书采用公钥体制，就是加密和解密所使用的不是同一个密钥，通常有两个密钥，即“公钥”和“私钥”，它们两个必需配对使用，否则不能打开加密文件。这里的“公钥”是指可以对外公布的，“私钥”则不能，只能由持有人一个人知道。当发送一份保密文件时，发送方使用接收方的公钥对数据加密，而接收方则使用自己的私钥解密，这样信息就可以安全无误地到达目的地了。通过数字的手段保证加密过程是一个不可逆过程，即只有用私有密钥才能解密。

在公开密钥密码体制中，常用的一种是 RSA 体制。其数学原理是将一个大数分解成两个质数的乘积，加密和解密用的是两个不同的密钥。即使已知明文、密文和加密密钥（公开密钥），想要推导出解密密钥（私密密钥），在计算上是不可能的。按现在的计算机技术水平，要破解目前采用的 1024 位 RSA 密钥，需要上千年的计算时间。公开密钥技术解决了密钥发布的管理问题，商户可以公开其公开密钥，而保留其私有密钥。购物者可以用公开密钥对发送的信息进行加密，安全地传送给商户，然后商户用自己的私钥进行解密。

验证证书的过程就是迭代寻找证书链中下一个证书和他的上级 CA 的证书。在使用每一个证书前，必须检查相应的 CRL（Certificate Revocation List，证书吊销列表）[22]。用户检查证书的路径，是从最后一个证书所签发的证书有效性开始，检查每一个证书，一旦验证后，就提取该证书的公钥，用于检验下一个证书，直到验证完发送者的签名证书，并将该证书中包括的公钥用于验证签名。图 4.5 表示的是数字证书工作原理。

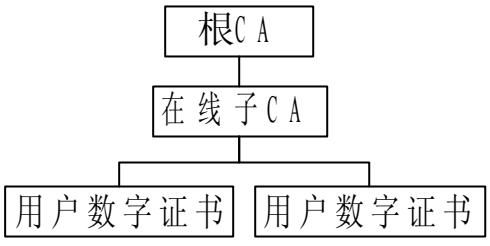

图 4.5 数字证书工作原理图

4.4.3 数字证书的申请

数字证书的申请流程如下：

（1）用户自己产生公钥私钥密钥对。

（2）用户将自己的公钥和相关信息发送给 CA，形成证书请求包，提出数字证书申请。

（3）CA 对用户的身份进行认证，认证通过后，CA 产生相应的证书，并用自己的私钥对用户证书进行数字签名，发送给用户。

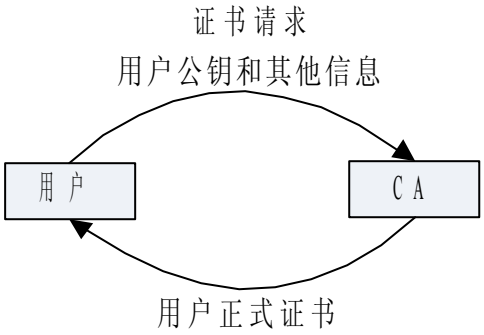

图 4.6 数字证书的申请

4.4.4 数字证书的获取

在验证信息的数字签名时，用户必须事先获取信息发送者的公钥证书，以对信息进行解密验证，同时还需要 CA 对发送者所发送的证书进行验证，以确定发送者身份的有效性。数字证书的获取有如下方式：

（1）发送者发送签名信息时，附加自己的证书。

（2）单独发送证书信息的通道。

（3）访问发布数字证书的目录服务器。

4.4.5 数字证书的存储

将数字证书进行加密，然后放入 SD 智能卡中进行存储而产生令牌，将数字证书和私钥存储在 SD 智能卡上进行认证；安全 SD 智能卡，有一定的存储空间，可以存储用户的私钥以及数字证书，利用 SD 卡内置的公钥算法实现对用户身份的认证。因为用户私钥保存在密码锁中，理论上使用任何方式都无法读取，因此保证了用户认证的安全性，用户在使用安全 SD 智能卡前也必须由静态的 PIN 码进行确认，优点是具有很高的安全性以及携带方便，缺点是 PIN 码是在客户的终端上输入容易被黑客获取，因此使用后应立即拔除。

4.4.6 数字证书的作废

当密钥泄漏时，被泄密的私钥所对应的公钥证书应被废止。对一般用户而言，私钥的泄密可能是 SD 智能卡丢失损坏或者遗失。还有就是证书中所含的证书持有者已终止或者与某组织的关系已经终止，则相应的公钥证书也应该作废。还可能是被新证书替代，原数字证书则过时而需要作废。但是，即使在删除数字证书时，也不会删除相应的私钥。同时要保证执行数字证书作废操作的执行权限。

数字证书的作废流程如下：

（l）用户自己向 CA 提出证书作废请求。

（2）CA 对用户的信息进行认证。

（3）验证用户身份通过后，CA 向用户返回请求签名的信息。

（4）用户对 CA 返回的信息进行签名。

（5）CA 验证用户签名信息无误，将该证书加入 CRL 中。

4.4.7 数字证书的更新

每个证书都有一个有效期。有效期过后，证书就不再被当作可接受的或可用的凭据。数字证书管理模块会在证书的有效期结束前更新当前数字证书。可以使用以前使用过的相同密钥更新证书，也可以使用新的密钥更新证书。取决于多个因素，包括证书的生存时间、现有或将来密钥的长度、受密钥对保护的数据值以及私钥已被恶意用户获取的可能性。

数字证书的更新流程如下：

（l）用户自己向 CA 提出证书更新请求。

（2）CA 对用户的信息进行认证。

（3）验证用户身份通过后，CA 向用户返回请求签名的信息。

（4）用户对 CA 返回的信息进行签名。

（5）CA 验证用户签名信息无误，更新用户证书，并将更新后的数字证书返回给用户。

4.4 身份认证管理

身份认证管理所包括两个重要方面的认证，一是实体认证，另一个是消息认证，分别表示验证信息的发送者的真实身份、验证以及鉴别信息的完整性，确保数据在发送的过程中未被篡改。实现身份认证的技术基础即是消息摘要和数字签名。下面来分别阐述它们。

4.4.1 消息摘要

消息摘要算法是一种加密算法，将需加密的消息明文“摘要”成一串固定长度的散列值，即消息摘要，不同的明文摘要成的散列值总是不相同，同样的明文其摘要必定一致，并且由消息摘要不能反推消息明文。数字摘要将原本可变的很长的消息明文进行压缩变成与消息明文惟一映射的符号序列，使得在此基础上进行数字签名高效快捷。

如果原始信息发生了变化，则需重新用消息摘要算法来加密，并产生新的签名。因此它可以有效的对信息的更改和伪造进行防止并保证其完整性。

ElPKI 数字认证机制中消息摘要算法采用 MD-5 和 SHA-1。它具有五个特征：1）防冲突性，即两个不同的信息是不会产生相同的消息摘要；2）防预见性，即散列函数不可能事先产生某些预先假设好的目标消息；3）防可逆性，即散列函数是单向的，不能反转；4）公开性，即它的算法是公开的不需要保密，而由于它能够产生单向散列值就确保了其安全性；5）固定性，即它的消息摘要的长度是固定不变的。

4.4.2 数字签名

本数字认证机制采用数字签名技术与加密技术相结合的方式，利用数字签名来保证信息的完整性及不可伪造性，同时对传输的信息进行加密，保证其保密性。

发送端 A 的工作流程如下：

（1）发送端 A 对发送消息明文 M 形成消息摘要 Mmd；

（2）用 A 的私钥 Ap 对消息摘要 Mmd 进行数字签名，得到签名文件 Mds；

（3）随机产生保密密钥 As；

（4）用 A 的保密密钥 As 对欲发送的消息明文 M 加密，得到消息密文 Mp；

（5）用接收者 B 的公钥 Bpub 对保密密钥 As 进行加密，得到密钥密文 Asp；

（6）然后将消息摘要 Mmd、签名文件 Mds、消息密文 Mp、密钥密文 Asp 通过安全通道发送给接收者 B。

接收端 B 在接收到发送端 A 发过来的消息摘要 Mmd、签名文件 Mds、消息密文 Mp、密钥密文 Asp 文件后，其工作流程如下：

（1）对签名文件 Mds 进行数字签名验证，如果通过则继续进行下面的步骤，否则，认为消息 M 的完整性不可信任。

（2）用 B 的私钥对密钥文件 Asp 进行解密，得到发送者 A 加密消息明文时用到的密钥，为区别原来的密钥，此处记为 As’；

（3）用 B 的保密密钥 As’对消息密文 Mp 进行解密，得到消息明文，为区别原来的消息明文，此处记为 M’；

（4）采用与 A 进行消息摘要时使用的消息摘要算法，对消息明文 M’形成消息摘要，为区别原来的消息摘要，此处记为 Mmd’；

（5）将 B 得到的消息摘要 Mmd’与 A 得到的消息摘要 Mmd 相对比，如果相同，则说明传输到 B 端的消息是没有篡改过的，是可信任的。否则，消息不可信任。

整个数字签名与验证的过程如图 4.7 所示。

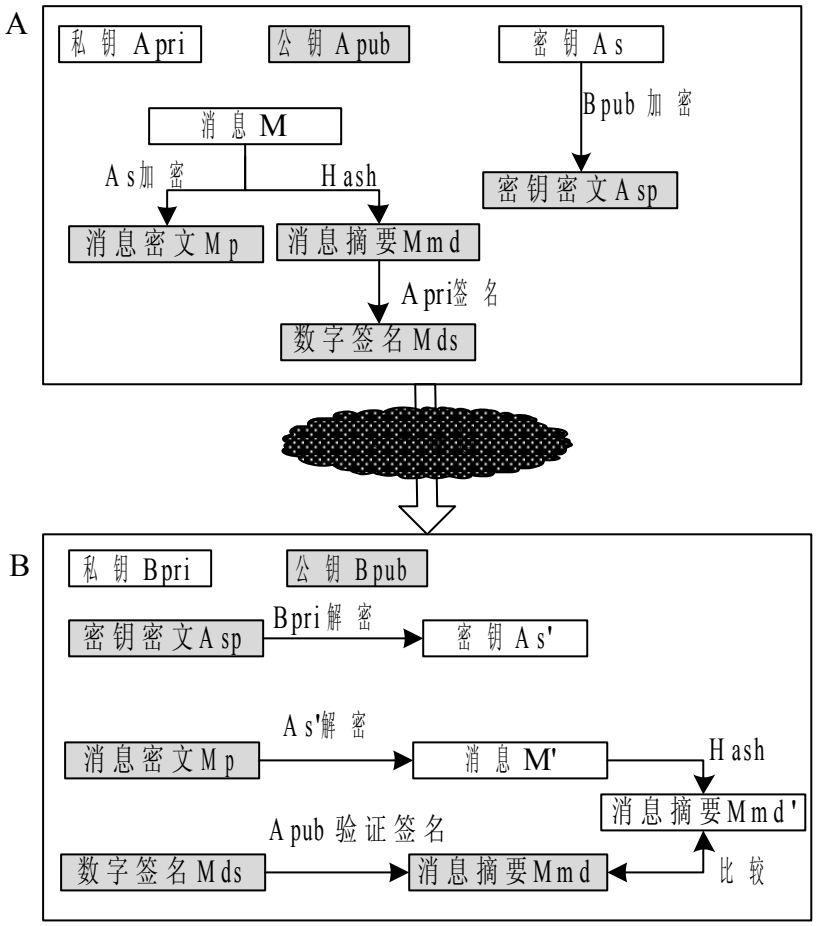

图 4.7 数字签名和验证过程

4.4.3 身份认证协议

本数字认证机制的身份认证采用 ISO 鉴别框架，也称为 X.509 协议。将移动公钥基础设施与 ISO 鉴别框架结合一起使用，可以提供很好的网间鉴别功能。

ISO 鉴别协议的具体过程如下：

当 A 与 B 通信时，A 首先查找 CA 的目录服务并得到一个从 A 到 B 的证书路径（Certification Path）和 B 的公开密钥。这时 A 可以使用单向、双向或三向鉴别协议。

单向协议是从 A 到 B 的单向通信。它建立了 A 和 B 双方身份的证明以及从 A 到 B 的任何通信信息的完整性。它还可以防止通信过程中的任何重放攻击。双向协议与单向协议类似，但它增加了来自 B 的应答。它保证是 B 而不是冒名者发送来的应答，它还保证双方通信的机密性并可防止重放攻击。单向和双向协议都使用了时间标记。单向协议增加了从 A 到 B 的另外消息，并避免了使用时间标记（用鉴别时间取代）。

单向协议包括（1）到（8）步，双向协议包括一个单向协议和一个从 B 到 A 的类似的单向协议。而三向协议完成双向协议的工作，但没有时间标记。T_A＝T_B＝0 时，从第(11)到第(15)步与双向协议相同。下面是三向协议的整个过程。

（1）A 产生一个随机数 R_A。

（2）A 构造一条消息，$M = (T_A, R_A, I_B, d)$，其中 T_A 是 A 的时间标记，I_B 是 B 的身份证明，d 为任意的一条数据信息。为安全起见，数据可用 B 的公开密钥 E_B 加密。

（3）A 将$(C_A, D_A (M))$ 发送给 B。其中 C_A 为 A 的证书，D_A 为 A 的私钥。

（4）B 确认 C_A 并得到 E_A，确认这些密钥没有过期。E_A 为 A 的公开密钥。

（5）B 用 E_A 解密 $D_A(M)$，验证了 A 的签名和所签发信息的完整性。

（6）B 检查 M 中的 I_B 以为保证准确

（7）B 检查 M 中的 T_A 以证实消息是刚发来的。

（8）B 对照旧随机数数据库来检查 M 中的 R_A，确保消息不是旧消息的重发。这一步是可选的。

（9）B 产生另一个随机数 R_B。

（10）B 构造一条消息，$M' = (T_B, R_B, I_A, R_A, d)$，其中 T_B 是 B 的时间标记，I_A 是 A 的身份，d 为任意的一条数据信息。为了保证安全，可用 A 的公开密钥对数据加密。R_A 是 A 在第（1）步中产生的随机数。

（11）B 将 $D_B (M')$发送给 A。

（12）A 用 E_B 解密 $D_B (M')$，以确认 B 的签名和消息的完整性。

（13）A 检查 M′中的 I_A 保证准确，。

（14）A 检查 M′中的 T_B，并证实消息是刚发来的。

（15）A 可检查 M′中的 R_B 来确保消息不是重发的旧消息，是可选的。

（16）A 对照第（3）步中她发送给 B 的 R_A，以检查接收到的 R_A。

（17）A 将 $D_A(R_B)$发送给 B。

（18）B 用 E_A 解密 $D_A(R_B)$。这样可证明 A 的签名和消息的完整性。

（19）B 对照他在第（10）步中发送给 A 的 R_B，以检查他接收到的 R_B。

上面的三向协议的认证过程详图如图 4.8 所示。

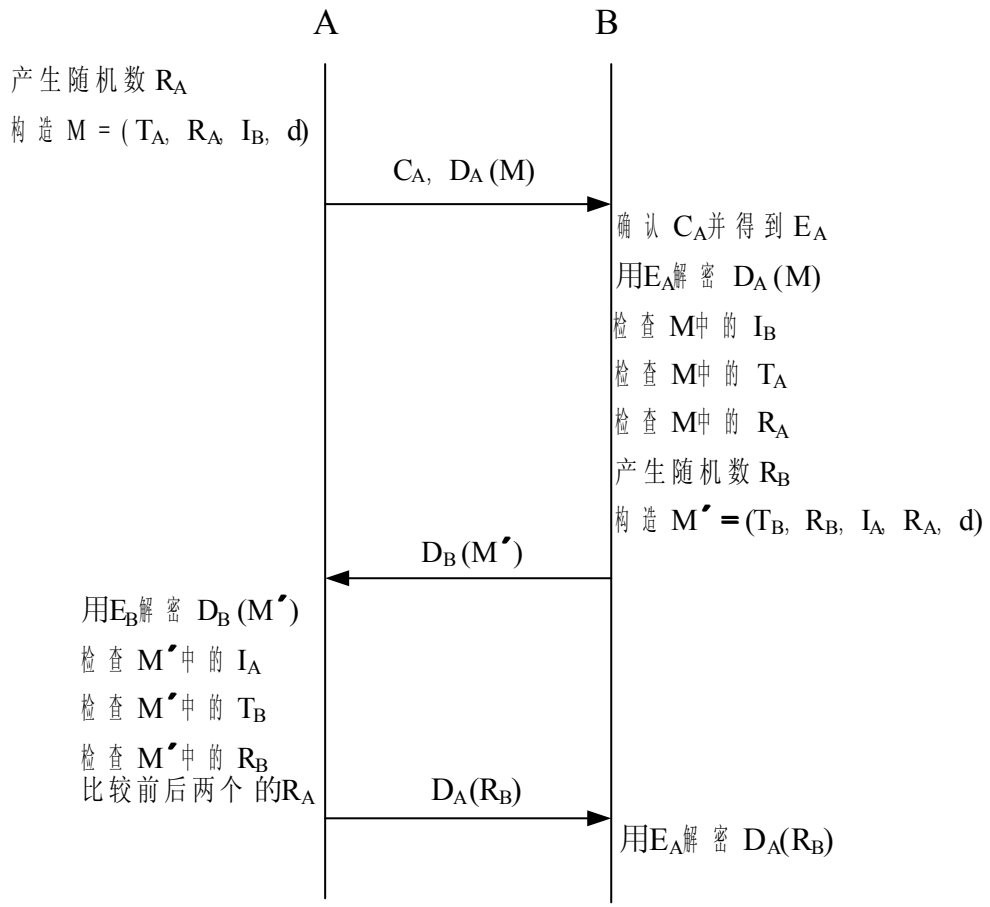

图 4.8 三向协议的认证过程图

4.5 安全传输管理

安全传输管理模块采用 SSL 安全协议建立安全传输通道，与外界传输数据建立 SSLSocket（安全套接字连接）。SSL 安全协议提供了数据加密、数字签名和建立完整的安全性连接三个主要功能。在这里，我们使用安全性连接功能，具体采用开源工程 OpenSSL 实现的其协议库[35]。数据加密和数字签名服务通过 PKCS#11 标准接口使用 SD 卡提供的相应服务。

此外，根据传输的数据类型的不同，以及便于数据的封装和解析，采用 XML 格式存储，然后在 SSL 的安全通道中进行传输。

4.5.1 安全传输协议

SSL 协议结构是一个分层的结构，位于传输层和应用层之间，包括 SSL 握手协议、SSL 更改密码协议、SSL 报警协议和 SSL 记录层协议[36]。图 4.9 表示的

是它的层次结构。

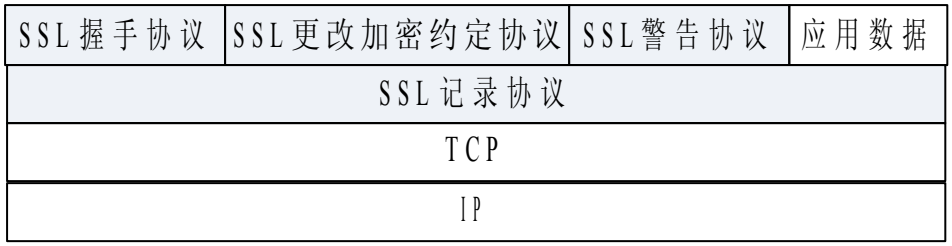

图 4.9 SSL 协议结构

SSL 记录协议和握手协议是 SSL 协议体系中两个主要的协议。SSL 记录协议确定数据安全传输的模式，SSL 握手协议用于客户机和服务器建立起安全连接之前交换一系列的安全信息，这些安全信息主要包括客户机确定服务器的身份、允许客户机和服务器选择双方共同支持的一系列加密算法、服务器确定客户机身份、通过非对称密码技术产生双方共同的密钥、建立 SSL 的加密安全通道。SSL 握手协议执行客户端和服务器 SSL 会话的建立过程。SSL 更改密码协议负责协商会话用的密码套接字。SSL 报警协议负责在客户端和服务器间传递 SSL 错误信息。

SSL 采用 TCP 作为传输协议提供数据的可靠传输[37]，其中 SSL 记录协议位于可靠的传输层协议（如 TCP/IP）之上，用来将数据流分割成一系列的数据段，然后分别对每个数据段单独进行数据保护和传输。在接收方，对每条记录单独进行解密和验证，这样的设计使数据一经准备好就可以从连接的一端传到另一端，而接受端也可以立即对数据处理。

对应用数据进行加解密的加密方法和密钥是通过握手协议产生的。当客户端和服务器首次进行通信时，他们要对 SSL 协议版本号、加密算法、产生共享密钥的公钥、加密算法以及双方是否进行相互认证进行协商。这一过程为建立会话连接，示意图如图 4.10 所示。

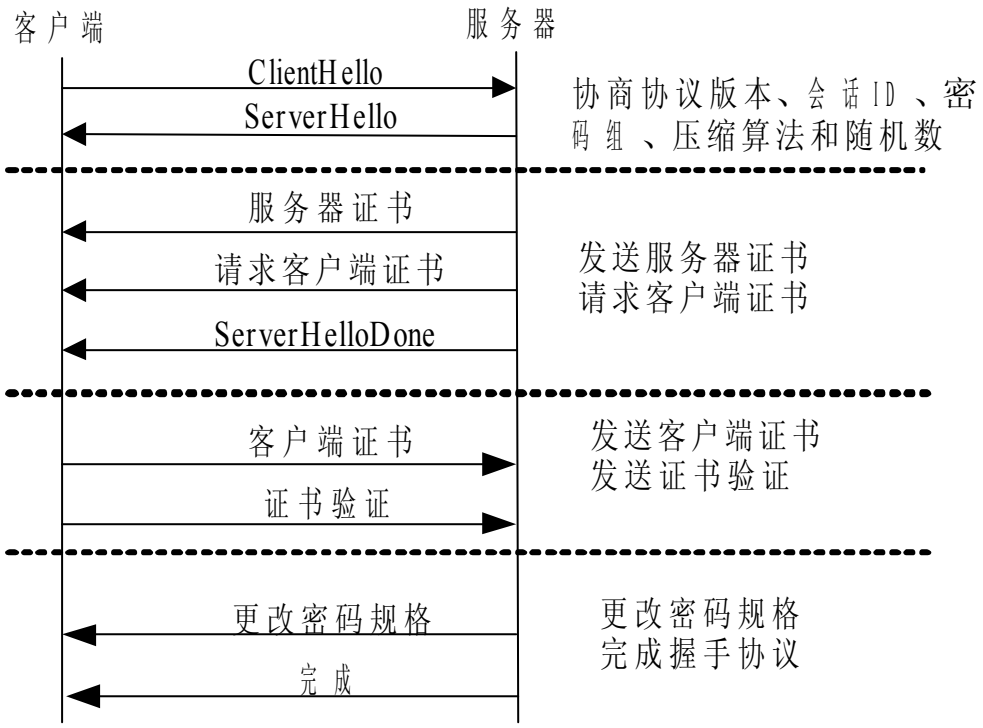

图 4.10 握手次序图

第一阶段：客户端发送一个 ClientHello 消息，包括版本、随机数、会话 ID、支持的密码算法列表、压缩方法列表等参数。然后，客户端等待服务器的 ServerHello 消息。服务器发送 ServerHello 消息，包括客户端建议的最低版本以及服务器支持的最高版本、服务器产生的随机数、会话 D、服务器选中的由客户端建议的一套密码算法、服务器选中的由客户端建议的一个压缩方法等参数。

第二阶段（服务器认证和密钥交换阶段）：首先，服务器发送自己的证书，消息包括一个 X.509 证书或者一条证书链。然后，对于非匿名服务器发送请求客户端证书消息，向客户端请求一个证书（包括证书类型和可接受的 CA）。最后，服务器发送 ServerHelloDnoe 消息等待应答。

第三阶段（客户认证和密钥交换阶段）：客户端收到 ServerHelloDone 消息后，根据需要验证服务器提供的证书，并判断 ServerHelloDone 的参数是否可以接受，如果都没有问题的话，发送响应消息给服务器。

第四阶段，服务器同样发送更改密码规格消息和完成消息。

此时，SSL 握手过程完成，为客户端和服务器建立起了一个通道级别的安全连接。

在安全连接通道中传输的数据是经过 SSL 记录协议封装的，这些数据包括应用层协议数据和普通数据。SSL 记录协议将这些数据分割为若干数据段，这些数据可以是任意长度的非空数据块，然后对这些数据块进行压缩、应用 MAC、

加密、增加首部，然后把密文交给下一层网络传输协议处理。对收到的数据，处理过程与上相反，即解密、验证、解压缩、重新拼装，然后发送到更高层的用户。如图 4.11 表示了 SSL 记录协议的整个执行过程。

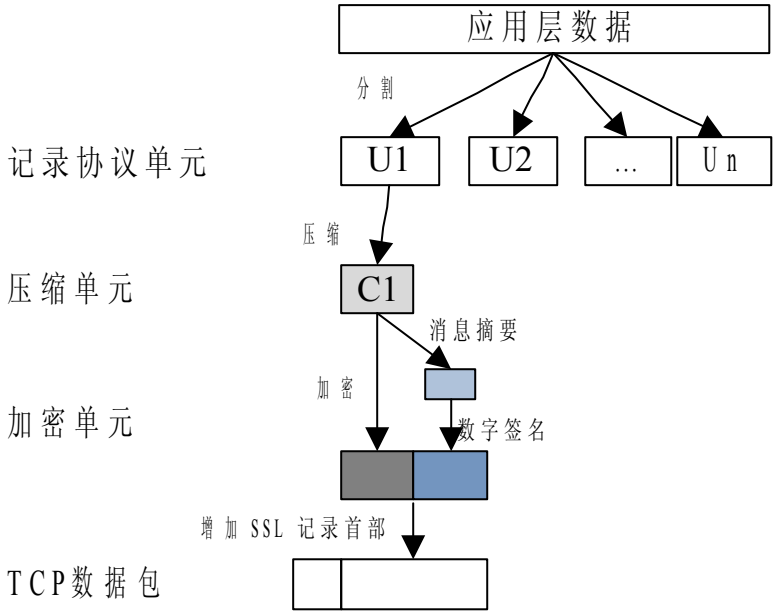

图 4.11 SSL 记录协议执行过程

4.5.2 XML 数据传输格式

在安全通道中需要传输的数据包括消息密文、消息摘要、数字签名、随机密钥密文。为了便于对这些数据进行封装和解析，本数字认证机制采用 XML[38][39] 格式存储它们。存储格式如下列代码所示。

```
<?xml version="1.0" encoding="utf-8"?>
<message>
    <messagecrypto>消息密文数据</messagecrypto>
    <messagedigest>消息摘要数据</messagedigest>
    <digitalsign>数字签名数据</digitalsign>
    <keycrypto>随机加密密钥密文数据</keycrypto>
</message>
```

<message>是根结点，所有实体数据包含在<message>结点下，<messagecrypto>结点表示是消息密文，<messagedigest>结点表示是由消息密文经过散列函数产生的消息摘要，<digitalsign>结点表示的是由数据摘要经过签名产生的数字签名数据文件，<keycrypto>结点表示的是随机密钥密文，它是在对消息明文进行加密时用到的对称密钥，经过接收者的公钥加密后得到的。

此外，考虑到扩展性，新增加的数据可以以同样的方式封装到<message>结

点中。

4.6 运行时安全检查模块

传输的数据或分享的服务可以是普通数据文件，也可以是具有计算能力的CAR 构件。后者是可以在本地运行的，参与本地计算，可以调用本地系统资源，所以需要进行运行时安全检查，即在 CAR 构件 A 被动态加载时，检查它的数字签名，此外，如果这个 CAR 构件 A 需要动态加载其他 CAR 构件 B，需要验证 CAR 构件 A 是否有权限请求。

运行安全检查的目的是保证 CAR 构件运行时环境的安全性，方法是验证 Elastos 系统中原生 CAR 构件，尤其是安全构件，和参与本地计算的第三方 CAR 构件的真实身份和数据完整性，以及第三方CAR 构件请求其他CAR 构件的权限。

Elastos 系统中运行的 CAR 构件服务可能是系统原生的服务构件，也可能是第三方开发的服务构件。CAR 构件虚拟机的必须保证其能安全的运行，所谓安全就是指对虚拟机的执行环境、宿主计算机平台和其它 CAR 构件的执行来说是安全的[40][41]。所以 CAR 构件虚拟机支持 CAR 构件运行时安全检查。

首先，制定CAR构件认证机制，保证在参与本地计算的所有CAR构件是安全的。CAR构件提供者提供的CAR构件服务在发布之前，都必须经过Elastos的数字签名机制进行安全认证，保证这些CAR构件的安全性和真实性。

然后，在加载CAR构件时，对它进行安全检查，包括数字签名，保证该CAR构件的真实身份。如果通过检查，则正常加载，进入下一步的代码运行时安全检查；如果不通过检查，则不加载，并返回错误代码。

CAR构件虚拟机采用域（Domain）的机制来对CAR构件进行运行时安全管理，它的模型如图4.12所示。CAR构件虚拟机通过加载器将CAR构件中的代码加载到虚拟机的域中，然后通过域来管理这些CAR构件。CAR构件A和CAR构件B运行在域1中，而CAR构件C运行在域2中，CAR构件A、B和C的交互通过CAR构件虚拟机内部的接口来实现，它们之间是相互不可见的，可以有效隔离CAR构件A、B和C，从而保证它们之间的安全性。

引入域安全机制后，为保证CAR构件的运行，CAR构件虚拟机的执行引擎提供一些域间通信等的支持。

采用Applet作为CAR构件虚拟机的一个域，在加载CAR构件前，先创建一个Applet，然后将CAR构件加载到这个Applet中。最后由CAR构件虚拟机的执行引擎来管理这些已创建好的Applet，例如Applet的加载、以及它们之间的通信等。

或者采用进程作为CAR构件虚拟机的一个域，在需要域的时候，CAR构件虚

拟机就创建一个进程，在这个进程中运行CAR构件实体，域的管理也就是进程的管理，采用调用宿主计算机平台进程的管理相关接口的方式就可以实现对进程的运行和通信等的管理。

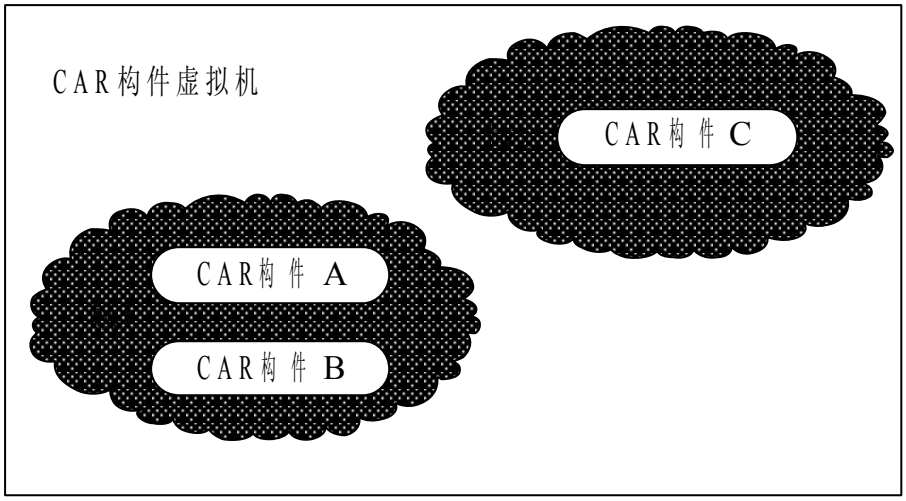

图4.12 CAR构件虚拟机中域机制

CAR 构件内部的代码采用的是 CPU 指令序列，这些代码在运行时，是直接在宿主计算机平台上运行的。所以，CAR 构件虚拟机中要提供保证 CAR 构件运行时的安全，就需要充分使用宿主计算机平台的安全机制，例如，CAR 构件的执行中的错误处理采用异常机制。

此外，CAR 构件虚拟机的适配器调用宿主计算机平台接口前，使用额外的安全控制。例如，对堆内存的分配和使用之前，在适配器中记录一些额外的安全信息。这样，CAR 构件在使用堆内存之前，虚拟机就可以对其进行越界判断等处理。

第5章 认证机制的实现

本章详细分析了认证机制的实现，它的作用是调用者与安全 SD 卡交互的衔接层，主要是基础安全构件 ElCryptoCAR 的接口定义和实现，以及为了将基础安全构件的应用环境扩展到 Android 计算模型的 Java 编程环境中，而详细定义了 ElCryptoJava。此外，还分析了涉及到的关键数据结构、错误管理和接口参数规则。

5.1 数据结构

5.1.1 数据类型

ElCryptoCAR 构件中的数据类型包括基本数据类型和扩展数据类型，其中基本数据类型采用 CAR 构件编程规范要求的基本数据类型。但是为了更方便地使用它们，对它们进行了重新声明。例如字节类型为 8 位无符号整数类型和相应的指针类型的声明如下：

```
typedef   Byte              ElPKI_BYTE;
typedef   ElPKI_BYTE *      ElPKI_BYTE_PTR;
```

同样，还有如下的基本数据类型，分别列举如下：

ElPKI_CHAR 和 ElPKI_CHAR_PTR：分别表示单字节字符类型和相应的指针类型；

ElPKI_INT8、ElPKI_INT8_PTR 和 ElPKI_UINT8、ElPKI_UINT8_PTR：分别表示 8 位有符号和无符号整型和相应的指针类型；

ElPKI_INT16、ElPKI_INT16_PTR 和 ElPKI_UINT16、ElPKI_UINT16_PTR：分别表示 16 位有符号和无符号整型和相应的指针类型；

ElPKI_INT32、ElPKI_INT32_PTR 和 ElPKI_UINT32、ElPKI_UINT32_PTR：分别表示 32 位有符号和无符号整型和相应的指针类型；

ElPKI_INT64、ElPKI_INT64_PTR 和 ElPKI_UINT64、ElPKI_UINT64_PTR：分别表示 64 位有符号和无符号整型和相应的指针类型；

ElPKI_LONG、ElPKI_LONG_PTR 和 ElPKI_ULONG、ElPKI_ULONG_PTR：分别表示有符号和无符号长整型和相应的指针类型；

ElPKI_FLOAT 和 ElPKI_FLOAT_PTR：分别表示单精度浮点数类型和相应的指针类型；

ElPKI_DOUBLE 和 ElPKI_DOUBLE_PTR：分别表示双精度浮点数类型和相应的指针类型；

ElPKI_VOID 和 ElPKI_VOID_PTR：分别表示空类型和相应的指针类型；

为了方便调用安全构件的函数接口和见名知义，所以重新定义了扩展数据类型，包括会话类型、对象类型、密钥类型和机制类型，以及安全连接上下文类型、SSL 库函数列表类型。现详细列举如下。

ElPKI_SESSION_HANDLE 和 ElPKI_SESSION_HANDLE_PTR 分别表示会话句柄类型和指向会话句柄类型的指针，用来识别当前打开的会话。定义如下：

```
typedef   ElPKI_ULONG                ElPKI_SESSION_HANDLE;
typedef   ElPKI_SESSION_HANDLE*      ElPKI_SESSION_HANDLE_PTR;
```

ElPKI_MECHANISM 和 ElPKI_MECHANISM_PTR 分别表示一个确定加密解密机制以及所需要的参数的结构体类型和指向这样的结构的指针。用来表示当前的密码操作采用哪种模式，实际上是封装的 Cryptoki 库中的 ElPKI_MECHANISM 结构体类型，其定义如下：

```
typedef   ElPKI_MECHANISM            ElPKI_MECHANISM;
typedef   ElPKI_MECHANISM *          ElPKI_MECHANISM_PTR;
```

ElPKI_OBJECT_HANDLE 和 ElPKI_OBJECT_HANDLE_PTR 分别表示一个对象句柄类型和指向对象句柄类型的指针，用来表示令牌上特定的对象。当在令牌上存在一个对象时，或者创建或者找到，为了使会话能够访问这个对象，就为它分配一个对象句柄。实际上是封装的 Cryptoki 库中的 ElPKI_OBJECT_HANDLE 结构体类型，其定义如下：

```
typedef   ElPKI_OBJECT_HANDLE        ElPKI_OBJECT_HANDLE;
typedef   ElPKI_OBJECT_HANDLE*       ElPKI_OBJECT_HANDLE_PTR;
```

ElPKI_SSLCTX 和 ElPKI_SSLCTX_PTR 分别表示的是安全连接上下文类型及指向它的指针，是建立安全连接的通信双方需要创建的全局上下文结构体，在程序的生命周期中只有一个，包含之后创建 SSL 连接对象所需的默认值。

```
typedef   SSLCTX                     ElPKI_SSLCTX;
typedef   ElPKI_SSLCTX*              ElPKI_SSLCTX_PTR;
```

ElPKI_SSL 和 ElPKI_SSL_PTR 表示的是建立安全连接的 SSL 套接字类型以及指向这个类型的指针，需要在通信双方在建立的安全连接中创建。它实际上是封装的 OpenSSL 库中的 SSL 结构体类型，其定义如下：

```
typedef   SSL                    ElPKI_SSL;
typedef   ElPKI_SSL*             ElPKI_SSL_PTR;
```

ElPKI_SSLMETHOD 和 ElPKI_SSLMETHOD_PTR 表示的是与安全连接通道相关的 SSL 库函数列表以及指向这个类型的指针，这些函数分别实现了各种协议版本，创建 SSLCTX 对象时必须提供的参数类型。它的定义如下：

```
typedef   SSL_METHOD             ElPKI_SSLMethod;
typedef   ElPKI_SSLMethod *      ElPKI_SSLMethod_PTR;
```

5.1.2 错误管理

调用基础安全构件和 Java 层接口时会出现错误，采用统一的处理方式，即通过函数的返回值的不同来判断函数执行成功与否。这些函数接口，需要将函数返回值类型声明为 ECode 类型。函数的成功调用的返回值均是返回 ElPKI_OK，其他不成功的情况，会返回不出的返回值，用宏定义来定义这些不同的返回值。例如 ElPKI_OK 的定义如下：

```
#define ElPKI_OK                    0x00000000
```

下面列举了一些常用的返回值的定义。

关于不明原因导致错误返回的返回值为 ElPKI _ERROR。

```
#define ElPKI _ERROR                0x00010001
```

由于设备而产生的各种返回值，如设备错误返回值 ElPKI_DEVICE_ERROR 和设备已经移除返回值 ElPKI_DEVICE_REMOVED。

```
#define ElPKI_DEVICE_ERROR          0x00020001
#define ElPKI_DEVICE_REMOVED        0x00020002
```

关于密钥的各种返回值，如密钥句柄无效、密钥大小错误和密钥已经改变等返回值。

```
#define ElPKI_KEY_HANDLE_INVALID    0x00030001
#define ElPKI_KEY_SIZE_RANGE        0x00030002
#define ElPKI_KEY_CHANGED           0x00030003
```

当某一对象类型句柄无效时，就会返回对象类型句柄无效错误码 ElPKI_OBJECT_HANDLE_INVALID。定义如下：

```
#define ElPKI_OBJECT_HANDLE_INVALID      0x00040001
```

在用户登录时需要提供登录密码 PIN，会出现密码错误、密钥无效、密码长度错误、密码过期错误等各种返回值，定义如下：

```
#define ElPKI_PIN_INCORRECT              0x00050001
#define ElPKI_PIN_INVALID                0x00050002
#define ElPKI_PIN_LEN_RANGE              0x00050003
#define ElPKI_PIN_EXPIRED                0x00050004
```

在数字签名验证时，提供的数字签名可能会出现签名无效、签名长度无效等错误，其对应的返回值定义如下：

```
#define ElPKI_SIGNATURE_INVALID          0x00060001
#define ElPKI_SIGNATURE_LEN_RANGE        0x00060002
```

由于加密 SD 卡是可移除设备，所以也会由于这一特性而出现的错误，比如设备当前不可用、无法识别 SD 卡、SD 卡写保护等，其相关的返回值定义如下：

```
#define ElPKI_TOKEN_NOT_PRESENT          0x00070001
#define ElPKI_TOKEN_NOT_RECOGNIZED       0x00070002
#define ElPKI_TOKEN_WRITE_PROTECTED      0x00070003
```

5.1.3 接口参数规则

采用 CAR 构件编程规范对接口参数进行处理，也就是将接口参数和返回参数都看作是函数的参数。其中，输入参数用[in]来表示，输出参数用[out]来表示。接口函数不负责处理数据的实际存储空间的分配和释放，而需要调用者来管理这些存储空间。

5.1 ElCryptoCAR构件设计与实现

基础安全构件 ElCryptoCAR 为上层应用程序和其他构件提供加密解密相关服务，是 CAR 构件运行时安全检查的基础，在加载 CAR 构件时验证构件的真实性和完整性。由于 ElCryptoCAR 也是系统构件，也经过了系统的数字签名，所以在加载时也包括自身的身份真实性检查。ElCryptoCAR 满足了 Elastos 计算模型对安全服务的需求。

5.1.1 密钥管理接口

密钥管理接口涉及到密钥或密钥对的生成、打包和解包密钥、派生密钥，下面来分别说明它们的主要功能和用法。

ElPKI_GenerateKey 的作用是生成一个保密密钥，创建新的密钥对象。其中，参数 session 是应用程序与令牌的会话句柄，为传入参数； mechanism 是指向密钥生成机制的指针，为传入参数；template 是指向密钥模板的指针，为传入参数；count 是 template 指向的模板中属性个数，为传入参数； key 是指向生成的密钥对象的句柄的指针，为传出参数。返回值类型为 ECode，如果调用成功，则返回 0，其他返回值意义，请参见错误管理小节。以后出现的返回值也是这样的。

```
ECode ElPKI_GenerateKey(
    /* [in] */ ElPKI_SESSION_HANDLE session
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_ATTRIBUTE_PTR template,
    /* [in] */ ElPKI_ULONG count,
    /* [out] */ ElPKI_OBJECT_HANDLE_PTR key);
```

ElPKI_GenerateKeyPair 的作用生成公钥私钥对，创建新的密钥对象。其中，参数 session 是应用程序与令牌的会话句柄，为传入参数；mechanism 是指向密钥生成机制的指针，为传入参数；publicKeyTemplate 和 privateKeyTemplate 分别是指向公钥模板和私钥模板的指针，为传入参数；publicKeyAttributeCount 和 privateKeyAttributeCount 分别是公钥模板和私钥模板的属性个数，为传入参数；publicKey 和 privateKey 分别是指向生成的公钥对象、私钥对象的句柄的指针，为传出参数。

```
ECode ElPKI_GenerateKeyPair(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_ATTRIBUTE_PTR publicKeyTemplate,
    /* [in] */ ElPKI_ULONG publicKeyAttributeCount,
    /* [in] */ ElPKI_ATTRIBUTE_PTR privateKeyTemplate,
    /* [in] */ ElPKI_ULONG privateKeyAttributeCount,
    /* [out] */ ElPKI_OBJECT_HANDLE_PTR publicKey,
    /* [out] */ ElPKI_OBJECT_HANDLE_PTR privateKey);
```

ElPKI_WrapKey 的作用是将一个密钥打包，即加密。其中，参数 session 是会话句柄，为传入参数；mechanism 是指向打包机制结构的指针，为传入参数；wrappingKey 是打包密钥的句柄，为传入参数；key 是指向被打包的密钥的句柄，为传入参数；wrappedKey 是指向打包好的密钥的指针，为传出参数；wrappedKeyLen 是 wrappedKey 的长度，为传入参数。

```
ECode ElPKI_WrapKey(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ElPKI_OBJECT_HANDLE wrappingKey,
    /* [in] */ ElPKI_OBJECT_HANDLE key,
    /* [out] */ ElPKI_BYTE_PTR wrappedKey,
    /* [in] */ ElPKI_ULONG_PTR wrappedKeyLen);
```

ElPKI_UnwrapKey 的作用是将打包的密钥解出来，功能与 ElPKI_WrapKey 相反，需要创建一个新的密钥对象。其中，参数 session 是会话句柄，为传入参数；mechanism 是指向解包机制结构的指针，为传入参数；unwrappingKey 是解包密钥的句柄，为传入参数；wrappedKey 指向打包好密钥的指针，为传入参数；wrappedKeyLen 是密钥 wrappedKey 的长度，为传入参数； template 指向新密钥的模块，为传入参数；attributeCount 是模块中的属性数，为传入参数；key 指向解包后密钥的句柄的指针，为传出参数。

```
ECode ECode ElPKI_UnwrapKey(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_OBJECT_HANDLE unwrappingKey,
    /* [in] */ ElPKI_BYTE_PTR wrappedKey,
    /* [in] */ ElPKI_ULONG wrappedKeyLen,
    /* [in] */ ElPKI_ATTRIBUTE_PTR template,
    /* [in] */ ElPKI_ULONG attributeCount,
    /* [out] */ ElPKI_OBJECT_HANDLE_PTR key);
```

ElPKI_DeriveKey 的作用是从基础密钥对象派生一个密钥对象。其中，参数 session 是会话句柄，为传入参数；mechanism 是指向密钥派生机制结构的指针，为传入参数；baseKey 是基础密钥对象句柄，为传入参数； template 指向新密

钥模板的指针，为传入参数；attributeCount 是模板 template 中的属性个数，为传入参数；key 指向接收派生密钥的句柄的指针，为传出参数。

```
ElPKI_DeriveKey(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_OBJECT_HANDLE baseKey,
    /* [in] */ ElPKI_ATTRIBUTE_PTR template,
    /* [in] */ ElPKI_ULONG attributeCount,
    /* [out] */ ElPKI_OBJECT_HANDLE_PTR key);
```

5.1.2 数字证书管理接口

数字证书管理接口主要涉及到如何申请、获取、存储、删除和更新数字证书，方便操作数字证书，并通过数字证书对象句柄来完成这些操作，对用户来说是透明的，以保证数字证书的安全性。下面来分别讲解各个接口函数的作用。

ElPKI_ApplyCertificate 的作用是申请数字证书。其中，参数 session 是会话句柄，为传入参数；参数 certificate 中包含着数字证书所需的信息，为传入参数；ca 表示是 CA 中心证书申请地址。如果成功，则返回 0，否则，详细错误码含义参见错误管理小节。

```
ECode ElPKI_ApplyCertificate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_OBJECT_HANDLE_PTR certificate,
    /* [in] */ ElPKI_CHAR_PTR ca);
```

ElPKI_GetCertificate 的作用是获取通信对方的数字证书，并将其保存在数字证书对象中。其中，参数 session 是会话句柄，为传入参数；参数 certificate 是指向数字证书对象句柄的指针，指向数字证书对象句柄，通过句柄来访问真正的证书对象，为传出参数；peer 表示是通信对方的数字证书存放地址，为传入参数。

```
ECode ElPKI_GetCertificate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [out] */ ElPKI_OBJECT_HANDLE_PTR certificate,
    /* [in] */ ElPKI_CHAR_PTR peer);
```

ElPKI_StoreCertificate 的作用是将数字证书存储在外部安全 SD 卡上，持久化保存。其中，参数 session 是会话句柄，为传入参数；certificate 是指向数字证

书对象句柄的指针，通过该句柄来间接访问真正的证书对象，并将其数据加密处理后保存到安全 SD 卡，为传入参数。

```
ECode ElPKI_StoreCertificate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_OBJECT_HANDLE_PTR certificate);
```

ElPKI_DeleteCertificate 的作用是删除本地存储的数字证书文件。其中，参数 session 是会话句柄，为传入参数；参数 certificate 是指向数字证书对象句柄的指针，指向数字证书对象句柄，通过句柄在安全 SD 卡上查找存储的证书文件，并将其删除，之后，该指为 NULL，为传入参数。

```
ECode ElPKI_DeleteCertificate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_OBJECT_HANDLE_PTR certificate);
```

ElPKI_UpdateCertificate 的作用是当本地数字证书文件临近过期时，更新本地存储的数字证书文件，并将新获取到的保存起来。功能相当于 ElPKI_GetCertificate 和 ElPKI_StoreCertificate。其中，参数 session 是会话句柄，为传入参数；

```
ECode ElPKI_UpdateCertificate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_OBJECT_HANDLE_PTR certificate);
```

5.1.3 身份认证管理接口

身份认证管理接口包括两大类，一类是消息摘要函数接口，另外一类是数字签名和验证函数接口。下面来分别分析这些接口函数。

ElPKI_DigestInit 的作用是初始化消息摘要操作。其中，参数 session 是会话句柄，为传入参数；mechanism 是指向摘要机制结构的指针，按照其指定的模式进行初始化，为传入参数。

ElPKI_DigestInit 调用成功后，可以进一步调用 ElPKI_Digest 进行对单一部分数据进行摘要操作，或者先调用零次或多次 ElPKI_DigestUpdate，然后调用 ElPKI_DigestFinal 对多部分数据进行摘要操作。最终得到原消息的摘要数据。如果要处理额外数据，必须再次调用 ElPKI_DigestInit，然后重复前面的操作。

```
ECode ElPKI_DigestInit(
    /* [in] */ ElPKI_SESSION_HANDLE session,
```

```
    /* [in] */ ElPKI_MECHANISM_PTR mechanism);
```

ElPKI_Digest 的作用是对单一部分数据进行摘要操作。其中，参数 session 是会话句柄，data 是需要摘要的数据；dataLen 是数据的长度；digest 是指向接收消息摘要的单元的指针，为传出参数；digestLen 是 digest 的长度。

该函数调用前必须调用 ElPKI_DigestInit 初始化。调用 ElPKI_Digest 总是会终止现用的摘要操作，它相当于调用一系列 ElPKI_DigestUpdate，然后调用 ElPKI_DigestFinal。

```
ECode ElPKI_Digest(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR data,
    /* [in] */ ElPKI_ULONG dataLen,
    /* [out] */ ElPKI_BYTE_PTR digest,
    /* [in] */ ElPKI_ULONG_PTR digestLen);
```

ElPKI_DigestUpdate 的作用是对多部分数据进行摘要操作，处理另一个数据部分。其中，session 是会话句柄；part 是指向数据部分的指针；partLen 是数据部分 part 的长度。

```
ECode ElPKI_DigestUpdate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR part,
    /* [in] */ ElPKI_ULONG partLen);
```

ElPKI_DigestFinal 的作用是终止多部分消息的摘要操作，并返回消息摘要数据。其中，参数 session 是会话句柄；digest 是指向接收消息摘要的存储单元的指针，为传出参数；digestLen 指向消息摘要 digest 长度的存储单元的指针。

```
ECode ElPKI_DigestFinal(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR digest,
    /* [in] */ ElPKI_ULONG_PTR digestLen);
```

ElPKI_SignInit 的作用是初始化签名操作。其中，参数 session 是会话句柄；mechanism 是指向签名机制结构的指针；key 是签名时使用的密钥的句柄。

ElPKI_SignInit 调用成功后，可以进一步调用 ElPKI_Sign 进行对单一部分数据签名，或者先调用零次或多次 ElPKI_SignUpdate，然后调用 ElPKI_SignFinal

对多部分数据进行签名，最终得到签名数据。如果要处理额外数据，必须再次调用 ElPKI _SignInit，然后重复前面的操作。

```
ECode ElPKI_SignInit(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_OBJECT_HANDLE key);
```

ElPKI_Sign 的作用是对单一部分数据进行签名操作。其中，参数 session 是会话句柄；data 是需要签名的数据；dataLen 是数据长度；signature 是指向接收签名的存储单元的指针；signatureLen 是 signature 的长度。

该函数调用前必须调用 ElPKI_SignInit 初始化。调用 ElPKI_ Sign 总是会终止现用的签名操作，它相当于调用一系列 ElPKI_SignUpdate，然后调用 ElPKI_SignFinal。

```
ECode ElPKI_Sign(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR data,
    /* [in] */ ElPKI_ULONG dataLen,
    /* [out] */ ElPKI_BYTE_PTR signature,
    /* [in] */ ElPKI_ULONG_PTR signatureLen);
```

ElPKI_SignUpdate 的作用是对多部分数据继续进行签名操作。其中，session 是会话句柄；part 是指向数据部分的指针；partLen 是数据部分 part 的长度。

```
ECode ElPKI_SignUpdate(
    /* [in] */ ElPKI_SESSION_HANDLE hSession,
    /* [in] */ ElPKI_BYTE_PTR pPart,
    /* [in] */ ElPKI_ULONG ulPartLen);
```

ElPKI_SignFinal 的作用是终止多部分数据的签名操作，并返回签名数据。其中，参数 session 是会话句柄；signature 是指向接收签名数据存储单元的指针，为传出参数；signatureLen 指向签名数据 signature 长度的存储单元的指针。

```
ECode ElPKI_SignFinal(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR signature,
    /* [in] */ ElPKI_ULONG_PTR signatureLen);
```

ElPKI_VerifyInit 的作用是初始化数字签名验证操作。其中，参数 session 是会话句柄；mechanism 是指向数字签名验证机制结构的指针；hKey 是签名验证时使用的密钥对象的句柄。

ElPKI_VerifyInit 调用成功后，可以进一步调用 ElPKI_Verify 对单一部分数据签名进行验证，或者先调用零次或多次 ElPKI_VerifyUpdate，然后调用 ElPKI_VerifyFinal 对多部分数据签名进行验证。如果要验证其他数据签名，必须再次调用 ElPKI _VerifyInit，然后重复前面的操作。

```
ECode ElPKI_VerifyInit(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_OBJECT_HANDLE key);
```

ElPKI_Verify 的作用是验证单一部分的数字签名。其中，参数 session 是会话句柄；data 指向数据；dataLen 是数据的长度；signature 指向签名；signatureLen 是签名的长度。如果调用成功则返回 ElPKIR_OK 值，说明签名是有效的，如果返回 ElPKIR_SIGNATURE_INVALID 值，则说明签名是无效的。

该函数调用前必须调用 ElPKI_VerifyInit 初始化。调用 ElPKI_Verify 总会终止现用的签名验证操作，它相当于调用一系列 ElPKI_VerifyUpdate，然后调用 ElPKI_SignFinal。

```
ECode ElPKI_Verify(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR data,
    /* [in] */ ElPKI_ULONG dataLen,
    /* [in] */ ElPKI_BYTE_PTR signature,
    /* [in] */ ElPKI_ULONG signatureLen);
```

ElPKI_VerifyUpdate 的作用是对多部分签名继续进行的验证操作，然后接着验证下一部分签名。其中，session 是会话句柄，part 是指向数据签名部分的指针；partLen 是 part 的长度。

```
ECode ElPKI_VerifyUpdate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR part,
    /* [in] */ ElPKI_ULONG partLen);
```

ElPKI_VerifyFinal 的作用是结束一次多部分验证操作，核查数字签名。其中，session 是会话句柄；signature 是指向签名数据的指针；signatureLen 是 signature 的长度。如果调用成功则返回 ElPKIR_OK 值，说明签名是有效的，如果返回 ElPKIR_SIGNATURE_INVALID 值，则说明签名是无效的。

该函数调用前必须调用 ElPKI_VerifyInit 初始化。调用 ElPKI_VerifyFinal 总是会结束现有的校验操作。

```
ECode ElPKI_VerifyFinal(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR signature,
    /* [in] */ ElPKI_ULONG signatureLen);
```

5.1.4 安全传输接口

安全传输接口提供了安全传输管理模块中要求的功能接口，以实现采用 SSL 安全协议与通信对方建立 SSLSocket 安全传输通道，以便传输需要的数据。下面来详细分析各个函数接口的定义和使用方法。

ElPKI_SSLLibraryInit 的作用是初始化 OpenSSL 库，无参数，调用成功的时候返回 ElPKI_OK。在使用 OpenSSL 库时首先要初始化它，然后才可以使用其他的函数接口。

```
ECode ElPKI_SSLLibraryInit (ElPKI_VOID);
```

ElPKI_ SSLCTXNew 的作用是创建 SSL 上下文。其中，参数 method 是指向 ElPKI_SSLMethod 结构体的指针，它包含着若干操作 SSLv3.0 的函数指针；参数 context 是指向返回 ElPKI_SSLCTX 的指针，表示的是 SSL 上下文环境，是传出参数类型。

```
ECode ElPKI_SSLCTXNew(
    /* [in] */ ElPKI_SSLMethod_PTR method,
    /* [out] */ ElPKI_SSLCTX_PTR context);
```

ElPKI_SSLCTXFree 的作用是释放不再使用的 SSL 上下文。其中，参数 context 是通过 ElPKI_SSLCTXNew 返回的指针，是传入参数类型。

```
ECode ElPKI_SSLCTXFree(
    /* [in] */ ElPKI_SSLCTX_PTR context);
```

ElPKI_SSLNew 的作用是在 SSL 上下文的基础上创建一个 SSL 套接字对象，然后所有的读写数据操作将在这个套接字上进行。其中，参数 context 是通过

ElPKI_SSLCTXNew 返回的 SSL 上下文指针，是传入参数；ssl 是 SSL 套接字结构体指针，是传出参数类型。

```
ECode ElPKI_SSLNew(
    /* [in] */ ElPKI_SSLCTX_PTR context,
    /* [out] */ ElPKI_SSL_PTR ssl);
```

ElPKI_SSLSetFileDescriptor 的作用是为 SSL 套接字绑定读写数据文件描述符，其中，参数 ssl 是 SSL 套接字结构体指针，是传入参数类型；参数 fd 是读写数据流的文件描述符，是传入参数类型。

```
ECode ElPKI_SSLSetFileDescriptor (
    /* [in] */ ElPKI_SSL_PTR ssl,
    /* [int] */ ElPKI_INT32 fileDescriptor);
```

ElPKI_SSLAccept 的作用是服务端监听客户端的请求，并建立二者之间的安全连接，其中，参数参数 ssl 是 SSL 套接字结构体指针，是传入参数类型。

```
ECode ElPKI_SSLAccept (
    /* [in] */ ElPKI_SSL_PTR ssl);
```

ElPKI_SSLConnect 的作用是客户端完成与服务器端的握手过程，建立起安全连接。其中，参数 ssl 是 SSL 套接字结构体指针，是传入参数类型。

```
ECode ElPKI_SSLConnect(
    /* [in] */ ElPKI_SSL_PTR ssl);
```

当建立起安全连接通道后，就可以使用如下的接口进行数据传输了。ElPKI_SSLWrite 的作用是将数据写入安全连接通道中，其中，参数 ssl 是 SSL 套接字结构体指针，是传入参数类型；参数 buffer 是指向要传输的数据的指针，是传入参数；参数 num 表示的是 buffer 所指向的数据的字节数，是传入参数。

```
ECode ElPKI_SSLWrite (
    /* [in] */ ElPKI_SSL_PTR ssl,
    /* [in] */ ElPKI_VOID_PTR buffer,
    /* [in] */ ElPKI_INT32 num);
```

ElPKI_SSLRead 的作用是将数据从安全连接通道中读取出来，其中，参数 ssl 是 SSL 套接字结构体指针，是传入参数类型；参数 buffer 是指向要接收的数据的指针，是传出参数；参数 num 表示的是 buffer 所指向的数据存储单位最大能接收的字节数，是传入参数。

```
ECode ElPKI_SSLRead (
    /* [in] */ ElPKI_SSL_PTR ssl,
    /* [out] */ ElPKI_VOID_PTR buffer,
    /* [in] */ ElPKI_INT32 num);
```

5.1.5 加密服务接口

加密服务主要是关于数据的加密与解密操作的函数，下面是各个加密函数和解密函数的定义和使用方法。

ElPKI_EncryptInit 的作用是初始化加密操作。其中，参数 session 是会话句柄；mechanism 是指向加密机制结构的指针；key 是加密密钥的句柄。它们都是传入参数。

ElPKI_EncryptInit 调用成功后，可以进一步调用 ElPKI_Encrypt 对单一部分数据进行加密，或者先调用零次或多次 ElPKI_EncryptUpdate，然后调用 ElPKI_EncryptFinal 对多部分数据进行加密。如果要加密其他数据，必须重新调用 ElPKI_EncryptInit，然后重复前面的操作。

```
ECode ElPKI_EncryptInit(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_OBJECT_HANDLE key);
```

ElPKI_Encrypt 的作用是对单一部分数据进行加密。其中，参数 session 是会话句柄，为传入参数；data 是指向数据存储单元的指针，为传入参数；dataLen 是加密数据的长度，为传入参数；encryptedData 是指向接收加密后的数据的存储单元的指针，为传出参数；encryptedDatedLen 是指向加密数据长度的存储单元的指针。如果调用成功则返回 ElPKIR_OK。

该函数调用前必须调用 ElPKI_EncryptInit 初始化。调用 ElPKI_Encrypt 总会完成当前加密操作，它相当于调用一系列 ElPKI_EncryptUpdate，然后调用 ElPKI_EncryptFinal。

```
ECode ElPKI_Encrypt(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR data,
    /* [in] */ ElPKI_ULONG dataLen,
    /* [out] */ ElPKI_BYTE_PTR encryptedData,
```

```
    /* [in] */ ElPKI_ULONG_PTR encryptedDataLen);
```

ElPKI_EncryptUpdate 的作用是对多部分数据继续进行加密操作。其中，session 是会话句柄；part 是指向数据部分的指针；partLen 是数据部分 part 的长度。encryptedPart 是指向接收加密后数据的存储单元的指针，为传出参数；encryptedPartLen 是指向加密数据长度的存储单元的指针。调用成功则返回 ElPKIR_OK。该函数调用前必须调用 ElPKI_EncryptInit 初始化。

```
ECode ElPKI_EncryptUpdate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR part,
    /* [in] */ ElPKI_ULONG partLen,
    /* [out] */ ElPKI_BYTE_PTR encryptedPart,
    /* [in] */ ElPKI_ULONG_PTR encryptedPartLen);
```

ElPKI_EncryptFinal 的作用是终止多部分数据加密操作，同时返回最后加密后的部分数据。其中，参数 session 是会话句柄；lastEncryptedPart 是指向接收最后一个加密数据部分的存储单元的指针；lastEncryptedPartLen 是指向包含最后加密数据部分长度的单元。该函数调用前必须调用 ElPKI_EncryptInit 初始化。

```
ECode ElPKI_EncryptFinal(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [out] */ ElPKI_BYTE_PTR lastEncryptedPart,
    /* [in] */ ElPKI_ULONG_PTR lastEncryptedPartLen);
```

ElPKI_DecryptInit 的作用是初始化解密操作。其中，参数 session 是会话句柄；mechanism 是指向解密机制结构的指针；key 是解密时使用的密钥的句柄。

ElPKI_DecryptInit 调用成功后，可以进一步调用 ElPKI_Decrypt 对单一部分数据进行解密，或者先调用零次或多次 ElPKI_DecryptUpdate，然后调用 ElPKI_DecryptFinal 对多部分数据进行解密，最终得到解密出来的数据。如果要解密其他数据，必须重新调用 ElPKI_DecryptInit，然后重复前面的操作。

```
ECode ElPKI_DecryptInit(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_MECHANISM_PTR mechanism,
    /* [in] */ ElPKI_OBJECT_HANDLE key);
```

ElPKI_Decrypt 的作用是对单一部分的加密数据进行解密。其中，参数 session 是会话句柄；encryptedData 是指向加密数据的指针；encryptedDataLen 是加密数

据的长度；data 是指向接收解密后数据的存储单元的指针，为传出参数；dataLen 是指向解密数据长度的指针。如果调用成功则返回 ElPKIR_OK。

该函数调用前必须调用 ElPKI_DecrypttInit 初始化。调用 ElPKI_Decrypt 总会完成当前解密操作，它相当于调用一系列 ElPKI_DecryptUpdate，然后调用 ElPKI_DecryptFinal。

```
ECode ElPKI_Decrypt(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR encryptedData,
    /* [in] */ ElPKI_ULONG encryptedDataLen,
    /* [out] */ ElPKI_BYTE_PTR data,
    /* [in] */ ElPKI_ULONG_PTR dataLen);
```

ElPKI_DecryptUpdate 的作用是对多部分加密数据进行解密操作，其中，参数 session 是会话句柄；encryptedPart 是指向加密数据的指针；encryptedPartLen 是加密数据的长度；part 是指向接收解密数据存储单元的指针；partLen 指向解密出的数据的长度。调用成功则返回 ElPKIR_OK。该函数调用前必须调用 ElPKI_DecrypttInit 初始化。

```
ECode ElPKI_DecryptUpdate(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [in] */ ElPKI_BYTE_PTR encryptedPart,
    /* [in] */ ElPKI_ULONG encryptedPartLen,
    /* [out] */ ElPKI_BYTE_PTR part,
    /* [in] */ ElPKI_ULONG_PTR partLen);
```

ElPKI_DecryptFinal 的作用是终止多部分加密数据的解密操作。其中，参数 session 是会话句柄;lastPart 是指向接收最终解密出的数据部分存储单元的指针，是传出参数；lastPartLen 是指向解密数据部分的长度。如果调用成功则返回 ElPKIR_OK。该函数调用前必须调用 ElPKI_DecryptInit 初始化。

```
ECode ElPKI_DecryptFinal(
    /* [in] */ ElPKI_SESSION_HANDLE session,
    /* [out] */ ElPKI_BYTE_PTR lastPart,
    /* [in] */ ElPKI_ULONG_PTR lastPartLen);
```

5.2 ElCryptoJava设计与实现

ElCryptoCAR 构件提供了基础的安全服务能力，满足了 Elastos 中应用程序和 CAR 构件的要求。为了能在 Java 应用中同样使用 ElCryptoCAR 提供的安全相关服务，就需要依靠 Java 与 CAR 互操作机制，在 Java 层面上定义相应的接口，即 ElCryptoJava。在调用 ElCryptoJava 的时候，依靠 Java&CAR 虚拟机将 ElCryptoCAR 构件加载起来，并绑定到刚创建的 Java 对象上，实现 Java 对象和 CAR 对象的互相访问。进而也满足了 Android 模型中 Java 应用程序对安全服务的需求。

5.2.1 Java 与 CAR 互操作规则

ElCryptoJava 类中定义了 Java 端的安全服务函数，其中每一个函数对应于基础安全构件 ElCryptoCAR 构件中的函数，两者一一对应。实际上，ElCryptoJava 中的函数调用的是 ElCryptoCAR 中相应的函数。

后面几小节中定义的函数都是 ElCryptoJava 中的函数，ElCryptoJava 类的定义形式如下。在 Java 应用程序中需要使用安全密码服务时，只需要像使用一般 Java 类一样创建 ElCryptoJava 对象，并调用相应的函数就可以了。

```
Import dalvik.annotation.ElastosClass;
@ElastosClass(Module=”ElCryptoCAR.eco”, Class=” ElCryptoCAR”)
Class ElCryptoJava {
   public native ElPKIObjectHandle generateKeyElPKI();
   public native ElPKIObjectHandle generateKeyPairElPKI();
   …….
}
```

然后，需要定义 ElCryptoJava 中的 annotation，即 ElastosClass，用来让 Java&CAR 虚拟机识别和加载 ElCryptoCAR 构件。其中详细代码如下：

```
package dalvik.annotation;
import java.lang.annotation.*;
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ElastosClass {
   String Module();
   String Class();
}
```

5.1.1 密钥管理接口

ElCryptoJava 的密钥管理接口同样涉及到密钥或密钥对的生成、打包和解包密钥、派生密钥，下面来分别说明它们的主要功能和用法。

generateKeyElPKI 的作用是生成一个密钥对象。其中，各参数的含义与 ElCryptoCAR 中的含义是一样的，类型是一致的。返回的是密钥对象引用。

```
public native ElPKIObjectHandle generateKeyElPKI (
    long session,  ElPKIMechanism mechanism,
    ElPKIAttribute template,  long count);
```

generateKeyPairElPKI 的作用生成一对公钥私钥对。其中，各参数的含义参见 ElCryptoCAR 中相应函数参数的含义，类型是一致的。

```
public native ElPKIObjectHandle generateKeyPairElPKI (
    long session,  ElPKIMechanism mechanism,
    ElPKIAttribute publicTemplate,  long publicCount,
    ElPKIAttribute privateTemplate,  long privateCount);
```

wrapKeyElPKI 的作用是将一个密钥打包，即将密钥加密后返回其引用。其中，各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_WrapKey 参数的含义。

```
public native ElPKIObjectHandle wrapKeyElPKI (
    long session,  ElPKIMechanism mechanism,
    ElPKIObjectHandle wrappingKey,  ElPKIObjectHandle key,
    long wrappedKeyLen);
```

unwrapKeyElPKI 的作用是将打包的密钥解出来，功能与 ElPKI_WrapKey 相反，需要创建一个新的密钥对象。各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_UnwrapKey 参数的含义。

```
public native ElPKIObjectHandle unwrapKeyElPKI (
    long session,  ElPKIMechanism mechanism,
    ElPKIObjectHandle unwrappingKey,  byte wrappedKey,
    long wrappedKeyLen,  ElPKIAttribute template,  long attributeCount);
```

deriveKeyElPKI 的作用是从基础密钥对象派生一个密钥对象，并返回其引用。各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_DeriveKey 参数的含义。

```
public native ElPKIObjectHandle deriveKeyElPKI (
    long session,  ElPKIMechanism mechanism,  ElPKIObjectHandle baseKey,
```

```
    ElPKIAttribute template,    long attributeCount);
```

5.1.2 数字证书管理接口

ElCryptoJava 的数字证书管理接口主要涉及到如何申请、获取、存储、删除和更新数字证书，方便操作数字证书。下面来分别讲解各个接口函数的作用。

applyCertificateElPKI 的作用是申请数字证书。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_ApplyCertificate 参数的含义。

```
public native void applyCertificateElPKI (
    long session, ElPKIObjectHandle certificate, char[] ca);
```

getCertificateElPKI 的作用是获取通信对方的数字证书，并将其保存在数字证书对象中。各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_GetCertificate 参数的含义。

```
public native ElPKIObjectHandle getCertificateElPKI (
    long session, char[] peer);
```

storeCertificateElPKI 的作用是将数字证书存储在外部安全 SD 卡上，持久化保存。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_StoreCertificate 参数的含义。删除成功返回 true，否则返回 false。

```
public native boolean storeCertificateElPKI (
    long session, ElPKIObjectHandle certificate);
```

deleteCertificateElPKI 的作用是删除本地存储的数字证书文件。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_DeleteCertificate 参数的含义。删除成功返回 true，否则返回 false。

```
public native boolean deleteCertificateElPKI (
    long session, ElPKIObjectHandle certificate);
```

updateCertificateElPKI 的作用是当本地数字证书文件临近过期时，更新本地存储的数字证书文件，并将新获取到的保存起来。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_UpdateCertificate 参数的含义。删除成功返回 true，否则返回 false。

```
public native boolean updateCertificateElPKI (
    long session, ElPKIObjectHandle certificate);
```

5.1.3 身份认证管理接口

ElCryptoJava 的身份认证管理接口同基础安全构件的接口，包括消息摘要函数接口、数字签名和验证函数接口。下面来分别分析这些接口函数。

digestInitElPKI 的作用是在使用消息摘要操作之间进行初始化操作。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_DigestInit 参数的含义。digestInitElPKI 调用成功后，可以进一步调用 digestElPKI 进行对单一部分数据进行摘要操作，或者先调用零次或多次 digestUpdateElPKI，然后调用 digestFinalElPKI 对多部分数据进行摘要操作。

```
public native void digestInitElPKI (
  long session, ElPKIMechanism mechanism);
```

digestElPKI 的作用是对单一部分数据进行摘要操作。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_Digest 参数的含义。

```
public native byte[] digestElPKI (
  long session, byte[] data, long dataLen);
```

digestUpdateElPKI 的作用是对多部分数据进行摘要操作，处理另一个数据部分。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_DigestUpdate 参数的含义。

```
public native byte[] digestUpdateElPKI (
  long session, byte[] data, long dataLen);
```

digestFinalElPKI 的作用是终止多部分消息的摘要操作，并返回消息摘要数据。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_DigestFinal 参数的含义。

```
public native byte[] digestFinalElPKI ( long session);
```

signInitElPKI 的作用是在进行数字签名之前要进行初始化签名操作。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_SignInit 参数的含义。

调用成功后，可以进一步调用 signElPKI 进行对单一部分数据签名，或者先调用零次或多次 signUpdateElPKI，然后调用 signFinalElPKI 对多部分数据进行签名，最终得到签名数据。如果要处理额外数据，必须再次调用本函数，然后重复前面的操作。

```
public native void signInitElPKI (
  long session, ElPKIMechanism mechanism, ElPKIObjectHandle key);
```

signElPKI 的作用是对单一部分数据进行签名操作，返回签名数据。其中，参数 session 是会话句柄；data 是需要签名的数据；dataLen 是数据长度。该函数调用前必须调用 signInitElPKI 初始化。

```
public native byte[] ElPKI_Sign(
  long session, byte[] data, long dataLen);
```

signUpdateElPKI 的作用是对多部分数据继续进行签名操作。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_SignUpdate 参数的含义。该函数调用前必须调用 signInitElPKI 初始化。

```
public native void signUpdateElPKI (
  long session, byte[] data, long dataLen);
```

signFinalElPKI 的作用是终止多部分数据的签名操作，并返回签名数据。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_SignFinal 参数的含义。该函数调用前必须调用 signInitElPKI 初始化。

```
public native byte[] signFinalElPKI (
  long session, byte[] data, long dataLen );
```

verifyInitElPKI 的作用是在进行验证数字签名前进行的初始化操作。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_VerifyInit 参数的含义。该函数调用成功后，可以进一步调用 verifyElPKI 对单一部分数据签名进行验证，或者先调用零次或多次 verifyUpdateElPKI，然后调用 verifyFinalElPKI 对多部分数据签名进行验证。

```
public native void verifyInitElPKI (
  long session, ElPKIMechanism mechanism,
  ElPKIObjectHandle key);
```

verifyElPKI 的作用是验证单一部分的数字签名。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_Verify 参数的含义。该函数调用前必须调用 verifyInitElPKI 初始化。调用 verifyElPKI 总会终止现用的签名验证操作，它相当于调用一系列 verifyUpdateElPKI，然后调用 verifyFinalElPKI 函数。

```
public native byte[] verifyElPKI (
  long session,   byte[] data,   long dataLen);
```

verifyUpdateElPKI 的作用是对多部分签名继续进行的验证操作，然后接着验证下一部分签名数据。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_VerifyUpdate 参数的含义。

```
public native void verifyUpdateElPKI (
    long session,  byte[] part,  long partLen);
```

verifyFinalElPKI 的作用是结束多部分验证操作以验证数字签名。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_VerifyFinal 参数的含义。如果调用成功则返回 true，说明签名是有效的，如果返回 false，则说明签名是无效的。该函数调用前必须调用 verifyInitElPKI 初始化。

```
public native boolean verifyFinalElPKI (
    long session,  byte[]signature,  long signatureLen);
```

5.1.4 安全传输接口

ElCryptoJava 的安全传输接口同样提供了安全传输管理模块中要求的功能接口，以实现采用 SSL 安全协议与通信对方建立 SSLSocket 安全传输通道，以传输需要的数据。下面来详细分析各个函数接口的定义。

initSSLLibraryElPKI 的作用是初始化 OpenSSL 库，无参数，调用成功的时候返回 true。在使用 OpenSSL 库时首先要初始化它，然后才可以使用其他的函数接口。

```
public native boolean initSSLLibraryElPKI ();
```

newSSLCTXElPKI 的作用是创建 SSL 上下文。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_SSLCTXNew 参数的含义。

```
public native ElPKISSLCTX newSSLCTXElPKI (
    ElPKISSLMethod method);
```

freeSSLCTXElPKI 的作用是释放不再使用的 SSL 上下文。实际释放操作是在 ElCryptoCAR 中相应的接口函数在做的。其中各参数的含义参见 ElCryptoCAR 中相应函数 ElPKI_SSLCTXFree 参数的含义。

```
public native void freeSSLCTXElPKI (
    ElPKISSLCTX context);
```

newElPKISSL 的作用是在 SSL 上下文的基础上创建一个 SSL 套接字对象，然后所有的读写数据操作将在这个套接字上进行。

```
public native ElPKISSL newElPKISSL (
    ElPKISSLCTX context);
```

ECode ElPKI_SSLSetFileDescriptor 的作用是为 SSL 套接字绑定读写数据文件描述符，其中，参数 ssl 是 SSL 套接字引用；参数 fileDescriptor 是读写数据流的文件描述符。如果调用成功则返回 true，否则返回 false。

```
public native boolean setFileDescriptorElPKISSL (
    ElPKISSL ssl,    int fileDescriptor);
```

acceptElPKISSL 的作用是服务端监听客户端的请求，并建立二者之间的安全连接，其中，参数参数 ssl 是 SSL 套接字引用。详细含义可以参考 ElCryptoCAR 中相应函数 ElPKI_SSL_accept 参数的含义。

```
public native boolean acceptElPKISSL (
    ElPKISSL ssl);
```

connectElPKISSL 的作用是客户端完成与服务器端的握手过程，建立起安全连接。其中，参数 ssl 是 SSL 套接字引用。

```
public native boolean connectElPKISSL (
    ElPKISSL ssl);
```

当建立起安全连接通道后，就可以使用如下的接口进行数据传输了。writeElPKISSL 的作用是将数据写入安全连接通道中，其中，参数 ssl 是 SSL 套接字引用；参数 buffer 是指向要传输的数据。

```
public native void writeElPKISSL (
    ElPKISSL ssl,    byte[] buffer);
```

readElPKISSL 的作用是将数据从安全连接通道中读取出来，其中，参数 ssl 是 SSL 套接字引用，函数返回的是指向要接收的数据。详细含义可以参考 ElCryptoCAR 中相应函数 ElPKI_SSL_Read 参数的含义。

```
public native byte[] readElPKISSL (
    ElPKISSL ssl);
```

5.1.5 加密服务接口

ElCryptoJava 的加密服务同样是关于数据的加密与解密操作的函数，下面是各个加密函数和解密函数的定义和使用方法。

encryptInitElPKI 的作用是在使用加密解密服务函数之间进行的初始化操作。其中，参数 key 是加密时需要的密钥。调用 encryptInitElPKI 成功后，可以进一步调用 encryptElPKI 对单一部分数据进行加密，或者先调用零次或多次

encryptUpdateElPKI，然后调用 encryptFinalElPKI 对多部分数据进行加密。如果要加密其他数据，必须重新调用 encryptInitElPKI，然后重复前面的操作。

```
public native void encryptInitElPKI (
        long session,    ElPKIMechanism mechanism,
        ElPKIObjectHandle key);
```

encryptElPKI 的作用是对单一部分数据进行加密。参数 data 表示是需要加密的数据。函数调用成功后返回加密后的数据。该函数调用前必须调用 encryptInitElPKI 进行初始化。

```
public native byte[] ElPKI_Encrypt(
    long session,      byte[] data);
```

encryptUpdateElPKI 的作用是对多部分数据继续进行加密操作。其中参数 part 表示是需要加密的数据；函数调用成功后返回加密后的数据。该函数调用前也必须调用 encryptInitElPKI 进行初始化。

```
public native byte[] encryptUpdateElPKI (
    long session,      byte[] part);
```

encryptFinalElPKI 的作用是终止多部分数据加密操作，同时返回最后加密后的部分数据。该函数调用前必须调用 encryptInitElPKI 进行初始化。

```
public native byte[] encryptFinalElPKI (
    long session);
```

decryptInitElPKI 的作用是初始化解密操作。其中参数 key 表示的是解密时使用的密钥。decryptInitElPKI 调用成功后，可以进一步调用 decryptElPKI 对单一部分数据进行解密，或者先调用零次或多次 decryptUpdateElPKI，然后调用 decryptFinalElPKI 对多部分数据进行解密，最终得到解密出来的数据。如果要解密其他数据，必须重新调用 decryptInitElPKI，然后重复前面的操作。

```
public native void decryptInitElPKI (
    long session,    ElPKIMechanism mechanism,
    ElPKIObjectHandle key);
```

decryptElPKI 的作用是对单一部分的加密数据进行解密。参数 encryptedData 表示的是需要解密的加密数据，即密文。函数调用成功返回解密后数据。该函数调用前必须调用 decryptInitElPKI 初始化。调用 decryptElPKI 总会完成当前解密操作，它相当于调用一系列 decryptUpdateElPKI，然后调用 decryptFinalElPKI。

```
public native byte[] decryptElPKI (
  long session,     byte[] encryptedData);
```

decryptUpdateElPKI 的作用是对多部分加密数据进行解密操作，参数 encryptedData 表示的是需要解密的加密数据，即密文。函数调用成功返回解密后数据。该函数调用前必须调用 decryptInitElPKI 初始化。

```
public native byte[] decryptUpdateElPKI (
  long session,     byte[] encryptedData);
```

decryptFinalElPKI 的作用是终止多部分加密数据的解密操作。函数调用成功返回解密后数据。该函数调用前必须调用 decryptInitElPKI 初始化。

```
public native byte[] decryptFinalElPKI (
  long session);
```

第6章 总结与展望

经过一年多时间的研究工作和努力，本课题已经取得了阶段性的成果，本文的撰写也将近尾声。在移动互联网的深入发展的背景下，在Elastos上开展着各种各样的业务，对安全的要求日益突出，尤其是安全相关业务，要求对传输的数据进行加密，保证敏感数据的保密性，同时还必须保证通信双方的真实身份。此外，各种业务应用的运行需要Elastos具有一个安全稳定的运行环境，提供运行时安全检查。

本文详细研究和讨论了基于安全 SD 卡的 Elastos 数字认证机制，根据移动电子商务等安全相关业务应用的安全需求提出了它的具体设计目标，分析了它的体系结构和特点，提供数据加密解密、数字证书管理、身份认证管理和安全数据传输服务。按照构件技术的设计思想设计了基础安全构件 ElCryptoCAR，为应用程序和其他 CAR 构件提供统一的调用接口。同时，依靠 Java 与 CAR 互操作技术设计了 Java 层接口 ElCryptoJava。这样，不仅为 Elastos 普通应用程序和 CAR 构件提供安全服务，还为 Android 系统中的 Java 应用程序提供高效的安全服务，同时满足了 Elastos 和 Android 两种计算模型对安全的需要。

此外，为了保证数字认证机制的稳定正常运行，设计 CAR 构件运行时安全检查模块，用于动态加载构件时的数据完整性检查和身份认证，以及构件间请求时是否拥有足够的权限，以达到动态地检查系统 CAR 构件自身安全性和第三方 CAR 合法性。

关于 Elastos 数字认证机制课题的研究工作取得了阶段性成果，但是后续的研究工作还是有很多的，还需要更加深入地研究。主要有如下几个方面：

（1）通过多方面的测试和实践，发现该认证机制存在的不足，制订详细的安全需求和问题报告，加入到后续研究工作中，不断完善该认证机制；

（2）操作安全 SD 卡的函数由于 PKCS#11 库来提供的，它是具有数字签名的，具有合法的加载与运行权限，但如果通过非法伪造的这个库去操作安全 SD 卡，获取密钥和数字证书等信息，是非常严重的，这种情况如何来避免。

（3）如何保证在多平台上运行的高效率等问题。

（4）还应该提供更多的认证方式，并实现细粒度化访问资源。

致谢

时光荏苒，现在已经是硕士研究生学习和生活的尾声，回首整个研究生学习生涯，收获很多，感慨也颇多。在这里，我不仅学到了宝贵的计算机理论知识，丰富了自己的头脑，还学到了做人的道理，让自己真正懂得那些深有感触的人生哲学。

首先，我衷心地感谢导师陈榕教授，感谢他在研究生期间给予我悉心的指导和帮助，他严谨的治学态度和执着的毅力深深地影响着我，为我真正竖立起了榜样。还有他对于计算机专业技术的深刻理解，让我懂得理论知识必须牢固掌握，而且还必须活学活用、学以致用，达到融会贯通，有做一番成就的信心和决定。

感谢顾伟楠老师和裴喜龙老师。顾老师的为人谦逊和治学严谨对我产生了积极的作用，在学习和论文撰写方面，他更是给予我无微不至的关心和指导。在实习期间和研究生的日常生活方面，裴老师不仅给我传授理论知识，更多的是教我学术研究方和做人道理，指出我的不足并帮助我及时改正它们，他的谆谆教诲我将牢记于心。

感谢上海科泰世纪有限公司的实习机会，感谢我的同学和我实习公司的同事在工作、学习和生活上给予我的帮助。

感谢父母在背后给我的关怀和支持。

参考文献

[1] 洪国彬，范月娇. 电子商务安全与管理. 北京：电子工业出版社，2006

[2] 张爱菊. 电子商务安全技术. 北京：清华大学出版社，2006.

[3] 杨帆. USBKEY体系研究与技术实现：[硕士论文]. 湖南：武汉大学，2004

[4] 罗建超. 基于PKCS#11规范的公共安全平台的研究与实现：[硕士论文]. 四川：电子科技大学，2003

[5] 王玉柱，吕述望，王挺. Intel的CDSA和Microsoft的CAPI. 计算机应用与研究，2001(5)

[6] 谭安芬，姜建国. 从CDSA看安全体系架构. 信息网络安全，2003(12)

[7] Taylor R. A comparison of CDSA to Cryptoki, Proceedings 22nd National information System Security Conference pp.270-281 Vol.1

[8] RSA Laboratories. PKCS #11: Cryptographic Token Interface Standard. V2.20. June 2004

[9] 郜宪林. DGSA、SSAF和CDSA安全体系结构比较与分析. 计算机工程与应用，2002(3)

[10] 上海科泰世纪有限公司. Elastos资料大全. 2010

[11] 上海科泰世纪有限公司. Elastos CAR构件与编程模型技术文档. 2006.11

[12] 上海科泰世纪有限公司. Java&CAR程序设计技术. 2010.10

[13] 潘爱民. COM原理与应用. 北京：清华大学出版社，1999

[14] Bill Venners. Inside the Java Virtual Machine, Second Edition. McGraw-Hill. 2000

[15] Oracle. The Java Virtual Machine Specification. http://java.sun.com/docs/books/jvms/

[16] Sheng Liang. The Java Native Interface Programmer's Guide and Specification. Addison Wesley. 1999

[17] Google. Android NDK Development Guide. http://www.android.com

[18] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. IEEE Computer Society. 2009, Vol 7 (1): 50～57

[19] Open Mobile Alliance. Enabler Release Definition for Wireless Public Key Infrastructure. 2004

[20] Carlisle Adams, Steve Lioyd. Understanding Public-key Infrastructure: Concepts, Standards, and Deployment Considerations, 2002.3

[21] M Myers, R Ankney, A Malpani et al. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol- OCSR .RFC2560. 1999

[22] R. Housley, W. Polk. RFC3280 Internet X .RFC3280. 509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile, 2008

[23] Wireless Application Protocol Forum. Wireless Transport Layer Security. 2001.

[24] Oracle. Oracle Solaris Security for Developers Guide. 2010

[25] 佘堃，郑方伟. PKI原理与技术. 西安：电子科技大学出版社，2007.

[26] 赵长超. WPKI在移动电子商务中的研究和实现：[硕士论文]. 山东：山东大学，2008

[27] Wolfgang Rankl. Smart Card Handbook. Wiley, John & Sons, Incorporated, 2002

[28] ISO. ISO/IEC7816. Identification cards. 2005

[29] 王爱英. 智能卡技术. 北京：清华大学出版社，2000

[30] 李翔. 智能卡研发技术与工程实践. 北京：人民邮电出版社，2003

[31] 黄美林基于PKCS#11的数据安全管理框架设计及其关键技术研究：[硕士论文]. 湖南：国防科学技术大学，2006

[32] 沈仟. 基于PKCS#11协议的安全平台关键技术研究与实现：[硕士论文]. 四川：电子科技大学，2004

[33] 陈阳，杨林，PKCS#11会话机制的研究与实现，信息安全与通信保密，2005(3)

[34] Schneier Bruce. Applied Cryptography. John Wiley & Sons Inc. 2005

[35] OpenSSL org. OpenSSL. www.openssl.org

[36] P. Chandra, M. Messier, J. Viega. Network Security with OpenSSL. California:O'Reilly. 2002

[37] ERIC Rescorla著. 崔凯译. SSL与TSL，中国电力出版社，2002.

[38] W3C org. Extensible Markup Language (XML). http://www.w3.org/TR/xml/

[39] XML Encryption Syntax and Processing. http://www.w3.org/TR/xmlenc-core/

[40] 陈俞飞. CAR构件虚拟机（Bonsai）的研究与实现：[硕士论文]. 上海：同济大学，2010

[41] 陆刚. Linux平台上和欣虚拟机的研究与实现：[硕士论文]. 上海：同济大学，2008

[42] 王成志. 移动电子商务安全体系研究：[硕士论文]. 上海：同济大学，2008

个人简历、在读期间发表的学术论文与研究成果

个人简历：

申波，男，1984年7月生。

2003年09月至2007年07月，济南大学，计算科学与技术专业，获学士学位

2008年09月至今，同济大学，计算机应用技术专业，硕士研究生

已发表论文：

申波，陈榕，王保卫. 基于服务模型的移动设备系统软件架构研究. 电脑知识与技术.2010，10