飞天诚信
我们构筑安全

ePass3000 开发指南

1.2 版

www.FTsafe.com

北京飞天诚信科技有限公司（以下简称“飞天公司”）尽最大努力使这篇文档中的内容完善且正确。飞天公司对于由这篇文档导致的任何形式的直接或间接损失不负有责任。这篇文档的内容会跟随产品的升级而有所变化。

修改记录：

日期	版本	修改
2005 年 6 月 15 日	1.0	第一版
2007 年 8 月 13 日	1.1	第一版第一次修订
2009 年 6 月 2 日	1.2	第一版第二次修订

北京飞天诚信科技有限公司

软件开发协议

北京飞天诚信科技有限公司（以下简称“飞天”）的所有产品，包括但不限于：开发工具包，磁盘，光盘，硬件设备和文档，以及未来的所有定单都受本协议的制约。如果您不愿接受这些条款，请在收到后的 7 天内将开发工具包寄回飞天，预付邮资和保险。我们会把货款退还给您，但要扣除运费和适当的手续费。

1. 许可使用

您可以将本软件合并、连接到您的计算机程序中，但其目的只是如开发指南中描述的那样保护该程序。您可以以存档为目的复制合理数量的拷贝。

2. 禁止使用

除在条款 1 中特别允许的之外，不得复制、反向工程、反汇编、反编译、修改、增加、改进软件、ePass 系列硬件和产品的其它部分。禁止对软件和产品的任何部分进行反向工程，或企图推导软件的源代码。禁止使用产品中的磁性或光学介质来传递、存储非本产品的原始程序或由飞天提供的产品升级的任何数据。禁止将软件放在服务器上传播。

3. 有限担保

飞天保证在自产品交给您之日起的 12 个月内，在正常的使用情况下，ePass 系列硬件和软件存储介质没有重大的工艺和材料上的缺陷。

4. 修理限度

当根据本协议提出索赔时，飞天唯一的责任就是根据飞天的选择，免费进行替换或维修。飞天对更换后的任何产品部件都享有所有权。

保修索赔单必须在担保期内写好，或发生故障 14 天内连同令人信服的证据交给飞天。当将产品返还给飞天或飞天的授权代理商时，须预付运费和保险。

除了在本协议中保证的担保之外，飞天不再提供特别的或隐含的担保，也不再对本协议中所描述的产品负责，包括它们的质量，性能和对某一特定目的的适应性。

5. 责任限度

不管因为什么原因，不管是因合同中的规定还是由于刑事的原因，包括疏忽的原因，而使您及任何一方受到了损失，由我方产品所造成的损失或该产品是起诉的原因或与起诉有间接关系，飞天对您及任何一方所承担的全部责任不超出您购买该产品所支付的货款。在任何情况下，飞天对于由于您不履行责任所导致的损失，或对于数据、利润、储蓄或其它的后续的和偶然的损失，即使飞天被建议有这种损失的可能性，或您根据第 3 方的索赔而提出的任何索赔均不负责任。

6. 协议终止

当您不能遵守本协议所规定的条款时，将终止您的许可和本协议。但条款 2，3，4，5 将继续有效。

CE Attestation of Conformity

The equipment complies with the principal protection requirement of the EMC Directive (Directive 89/336/EEC relating to electromagnetic compatibility) based on a voluntary test.

This attestation applies only to the particular sample of the product and its technical documentation provided for testing and certification. The detailed test results and all standards used as well as the operation mode are listed in

Test report No. 70407310011
Test standards: EN 55022/1998 EN 55024/1998

After preparation of the necessary technical documentation as well as the conformity declaration the CE marking as shown below can be affixed on the equipment as stipulated in Article 10.1 of the Directive. Other relevant Directives have to be observed.

FCC certificate of approval

This Device is conformance with Part 15 of the FCC Rules and Regulations for Information Technology Equipment.

USB

This equipment is USB based.

WEEE

Dispose in separate collection.

缩略语及术语

缩略语及术语	解释
PKCS#11 接口	由 RSA(www.rsasecurity.com)实验室推出的程序设计接口，将密码设备抽象成一种通用的逻辑视图即密码令牌（Cryptographic Token）提供给上层应用，做到设备无关性和资源共享。
CryptoAPI 接口(简称 CAPI)	由微软公司提供的密码(cryptography)操作接口，提供设备无关的或软件实现的密码算法封装，很容易使开发者能够开发出用于数据加解密、使用数字证书的身份认证、代码签名等的 Windows 平台上的 PKI 应用程序。
Token	密码设备的统称，可以是智能卡，也可以是具有密码和证书存储功能的任何设备。
USB Token	具有 USB 接口的密码设备，其携带方便，操作简单。
ePass3000	飞天公司推出的将智能卡和 USB 接口结合的便携式设备，具有智能卡的优点，又有携带方便的好处。支持 PKI 应用。
ePassNG (ePass Next Generation)	飞天公司推出的新一代的中间件框架产品，支持 ePass 系列等产品，并能够非常方便的增加被支持的硬件。支持 PKI 应用。

目 录

第一章 开发综述

本章讲述有关开发 ePass3000 应用程序的问题，涉及的内容包括 ePass3000 支持的各种开发接口和针对不同接口的开发方法。

- ePass3000 的应用开发接口
- 使用 MS CryptoAPI 接口开发 ePass3000 应用
- 使用 PKCS#11 接口开发 ePass3000 应用

1.1 ePass3000 的应用开发接口

ePass3000 的应用程序开发可大体分为两类：一类是 PKI 应用的开发，另一类是智能卡应用的开发。对于针对 PKI 应用开发的接口，ePass3000 提供了 PKCS#11 和 CSP for Microsoft CryptoAPI 2.0 两种主流的应用接口。这两个接口分别遵循 RSA 公司开发的 PKCS#11 标准和微软公司制定的 MS CryptoAPI 接口标准。由于这两个接口标准同时也被其他很多软硬件厂商所支持，因此，ePass3000 可以不需要二次开发就与符合这两种接口规范的应用集成使用。ePass3000 的另一类接口是针对智能卡应用的 PC/SC 接口。

ePass3000 的 PKI 应用接口本身就是建立在 PC/SC 接口之上的。应用程序开发者可以针对自己项目的需求采用一种或多种接口对 ePass3000 进行开发。

1.2 使用MS Crypto API接口开发ePass3000 应用

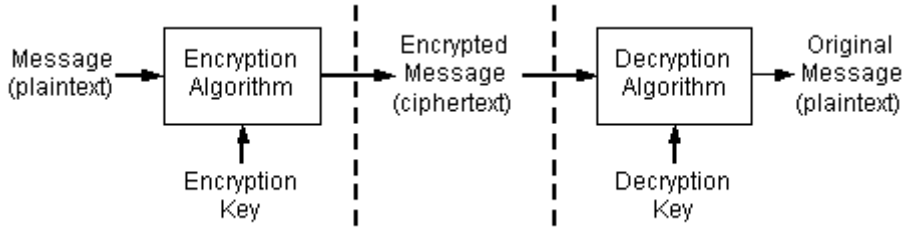
微软的 CryptoAPI 是 Win32 平台下为应用程序开发者提供的数据加密和安全的编程接口。CryptoAPI 函数集包含了基本的 ASN.1 编码、解码、散列、数据加密和解密、数字证书管理等重要的密码学应用功能。数据的加、解密支持对称算法和公开密钥算法两类算法。CryptoAPI 是所有微软的 Win32 应用程序及很多第三方厂商应用程序使用的数据加密接口，诸如 Internet Explorer 和 Outlook 等应用都是基于 CryptoAPI 开发的。

在不安全的网络上进行安全的数据传输涉及三个方面的要求：信息隐藏，身份鉴别和完整性检测。CryptoAPI 除了提供上述三个功能外还提供标准的 ASN.1 编码、解码、信息解密、数字证书和证书存储区的管理、证书信任列表、吊销列表和证书有效性检测等功能。

1.2.1 信息隐藏

信息隐藏的意义就是保证信息内容只能被特定的人获得。信息隐藏通常都要使用某种形式的密码学方法。数据加密算法能保证信息的安全隐藏和传输。数据加密算法将明文数据经过变化使其看上去像一组毫无规律的数据。在没有加密密钥的情况下，对于好的加密算法想从密文强制推导出明文是极其困难的。被加密的数据可以是任意 ASCII 码文本文件、数据库文件或其他任何需要进行安全传输的数据。在这里，“信息”这一术语用来表示任意一段数据，“明文”则指任意一段没有被加密的数据，“密文”则指任意一段被加密的数据。

被加密的数据可以通过不安全的介质或网络进行传输而不损害其安全性。之后，密文可被还原成明文，如下图所示：



数据加密和解密的概念非常简单，对数据加密的时候需要一个加密密钥，这里的密钥就相当于普通的门钥匙一样。解密的时候，需要使用一个解密密钥来解开数据。加、解密密钥可以相同也可以不同。
加密密钥必须小心保存，给其他用户时也必须通过安全的渠道传递。对解密密钥的访问权限必须小心控制，因为拥有解密密钥意味着可以解开所有用相应加密密钥加密的信息。

1.2.2 身份鉴别

安全通讯的前提是通讯的双方必须明确知道对方的身份。身份鉴别的任务就是鉴定一个用户或实体的真实身份。标识用户身份的文档通常被称为信任状或凭证。当兑现一张支票时，用户可以用身份证或驾照来表示自己的身份。身份证和驾照就是鉴别用户身份的有效凭证。护照是另外一个例子，海关官员通过护照来确定护照持有人的真实身份。海关官员信任发放护照的政府部门对持有人身份的鉴别。在上述两个例子中，用户身份的凭证存在于物理的文档中。

身份鉴别有时也用来判定接收到的数据就是被发送的数据。如果 A 向 B 发送了一段数据，B 需要鉴别这段数据就是 A 发出的，而不是其他人冒充 A 发出的。为了满足这类验证的要求，CryptoAPI 提供了数字签名和校验函数，来对信息进行鉴别。

因为在计算机网络上传输的数据与用户之间并没有物理的联系，因此对数据进行鉴别的凭证也必须能够在网络上进行传输。这种凭证必须由受信任的凭证发行机构发行。

数字证书也就是通常所说的证书就是这样一种凭证，是一种在计算机网络上进行身份验证的有效凭证。

数字证书是由一个被称为证书机构的被信任组织或实体颁发的凭证。它包含与证书对应的用户公钥以及其他一些记录证书主题和用户信息的数据。证书机构只有在验证了证书主题和证书对应的用户公钥的有效性后才会签发证书。

证书申请者和证书机构之间交互签发证书信息可以使用物理介质，比如软盘，进行传输。通常，这种信息交换都是在计算机网络上完成的。证书机构使用被信任的服务程序处理用户的请求和证书的签发工作。

1.2.3 完整性检测

任何通过不安全介质传输的信息都可能被意外或蓄意的修改。在现实世界中，印章就是用来提供和证明信息完整性的工具。例如一瓶阿司匹林使用不可还原的包装和完好的封印来证明它出厂后瓶里的药片没有变化。

同理，信息的接收者不但需要确定信息是由谁发送的，还需要确定自己收到的信息就是发送者发出的信息，而没有任何变化。要建立数据的完整性检测机制，不仅需要发送信息本身，还需要发送用来校验数据的信息，这一信息通常被称作哈希值。数据和验证信息都可以与数字签名一起发送来证明其完整性。

1.2.4 CSP与加密处理

Crypto API 函数使用“加密服务提供程序”（CSPs）完成数据加密、解密以及密钥的存储管理。所有的 CSPs 都是相互独立的模块。理论上，CSPs 应该是独立于特定的应用程序的，也就是说所有的应用程序都可以使用任何一个 CSP。但是，实际上有些应用程序只能与特定的 CSP 协作。CSP 与应用程序之间的关系就类似于 Windows GDI 模型。CSP 就类似于图形硬件驱动程序。

密钥存储的安全性完全取决于 CSP 的具体实现而与操作系统没有关系。这就使得应用程序无需修改就可以运行于多种安全环境之下。

应用程序与加密模块之间的访问必须受到严格的控制。只有这样才能保证应用的安全性和移植性。下面是三条应用原则：

- 应用程序不能直接访问密钥的内容。因为所有的密钥都是在 CSP 内部产生的，应用程序通过不透明的句柄对密钥进行访问。这就避免了应用程序和其关联的动态链接库泄漏密钥或使用不好的随机数产生密钥的可能。
- 应用程序不能指定加密操作的细节。CSP 接口允许应用程序选择进行加密或签名操作使用的算法类型，但是实际的操作完全由 CSP 内部进行控制。
- 应用程序不处理用户的信任凭证或其他身份鉴别数据。用户身份的鉴别是由 CSP 完成的。因此，对于未来可能出现的身份验证方式应用程序无需修改其身份验证模型。

最简单的 CSP 提供形式是由一个 Win32 动态链接库文件（DLL）和一个签名文件组成。必须提供正确的签名文件 CSP 才能被 CryptoAPI 识别和使用。CryptoAPI 对 CSP 的签名要经常进行周期性的检查，以防止 CSP 程序被篡改。

有些 CSP 模块通过本地 RPC 调用或硬件驱动程序在独立的地址空间中实现其关键的加密操作功能。将密钥和相关的加密操作放到具有独立地址空间的服务程序或硬件内可以保证密钥数据不被应用程序地址空间内的程序随意修改。

应用程序依靠某一 CSP 的特殊特性是很不明智的做法。例如，Microsoft Base Cryptographic Provider 现在提供 40 位的会话密钥和 512 位的公钥。应用程序应避免以此假设存储密钥数据所需的空间大小，因为一旦使用一个不同的 CSP 这一大小就可能不适用。好的应用程序应能与不同的 CSP 协同工作。

1.2.5 CSP上下文

应用程序调用的第一个 CryptoAPI 函数必定是 CryptAcquireContex。这个函数返回包含指定了特定密钥容器的 CSP 操作句柄。密钥容器的选择可以特别指定也可以使用登录用户缺省的容器。CryptAcquireContext 也可用来创建新的密钥容器。

CSP 模块本身是具有名称和类型的。例如，Windows 操作系统缺省安装的 CSP 程序的名字是：Microsoft Base Cryptographic Provider，其类型是 PROV_RSA_FULL。每个 CSP 的名称都必须是不同的，而类型可以相同。

当应用程序调用 CryptAcquireContext 函数获取一个 CSP 操作句柄时，可以指定 CSP 的类型和名字。如果指定了类型和名字则只有这两个属性匹配的 CSP 模块会被调用。调用成功后，函数返回 CSP 的操作句柄，之后应用程序就可以通过句柄访问 CSP 和 CSP 中的密钥容器了。

1.2.6 CryptoAPI体系架构

CryptoAPI 体系架构由五个主要部分组成：

- 基本加密函数

用来连接和建立 CSP 操作句柄的函数。这组函数允许应用程序通过指定名称和类型来选择特定类型的 CSP 模块。

➢ 密钥生成函数

用来生成和保存加密密钥。其功能包括修改加密模式，初始化加密向量等加密特性。

➢ 密钥交换函数

用来交换和传输密钥。

- 数字证书编码与解码函数

这组函数用来加密和解密数据。其功能还包括数据散列计算的支持。

- 数字证书存储函数

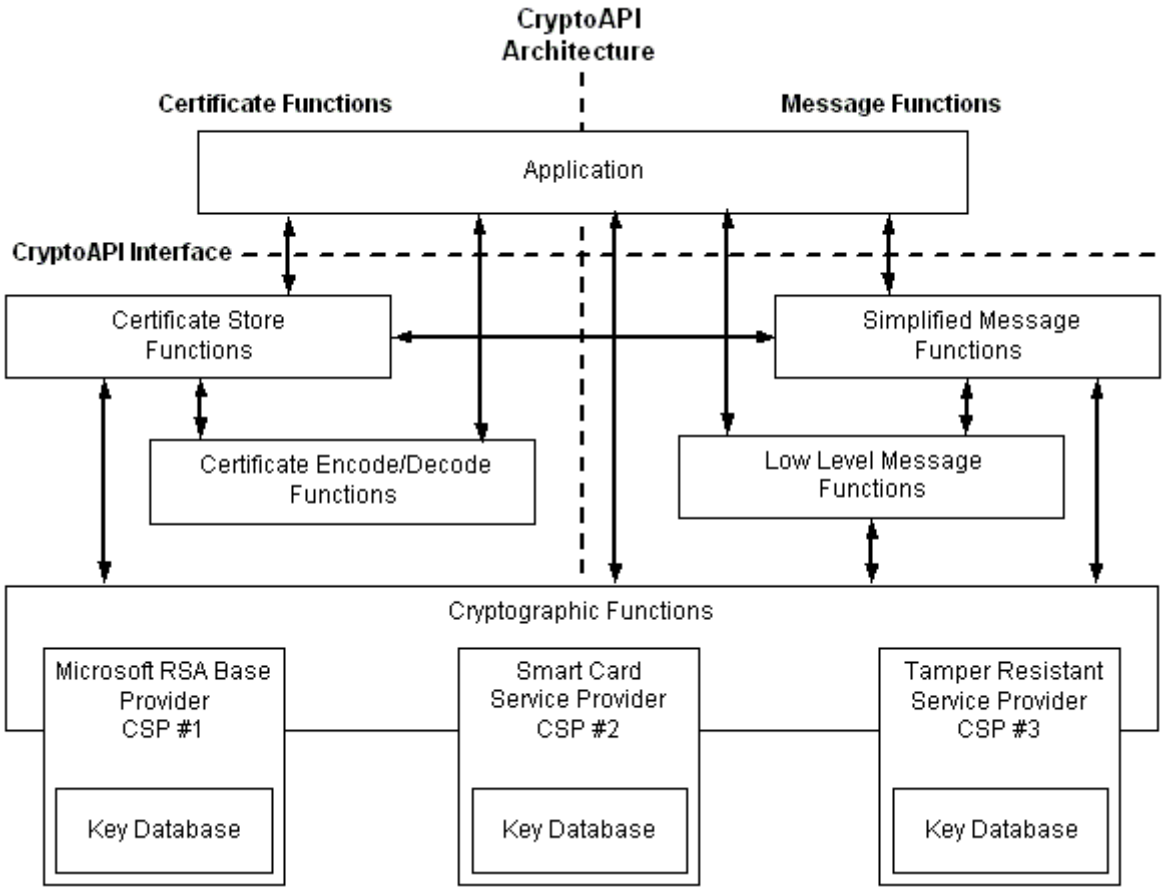
这组函数用来管理数字证书集合。

- 简化信息处理函数

这组函数用来加密和解密消息及数据，对消息和数据进行签名，验证消息及数据签名的有效性。

- 底层信息处理函数

这组函数是简化消息处理函数的实际实现函数。它提供了对消息进行各种操作的更为细致的控制。



每类函数的命名前缀都有约定，具体如下：

函数分类	前缀约定

基本加密函数	Crypt
数字证书编码与解码函数	Crypt
数字证书存储函数	Store
简化信息处理函数	Message
底层信息处理函数	Msg

1.3 使用PKCS#11 接口开发ePass3000 应用

由于 Internet 全球范围内的爆炸式增长，应用程序对公众网领域中的安全事务和通讯的要求也日益迫切。而加密安全产品的迅猛增长也导致了对应用程序之间交互性的需求。因此 RSA 公司创立了公开密钥加密标准(PKCS)来满足这一要求。

PKCS#11 是 PKCS 系列标准中的一个。PKCS#11 标准（也称为“Cryptoki”）被设计用来解决不同厂商与开发者的公开密钥应用之间交互与兼容的问题。它定义了一个通用的编程接口模型 Cryptoki tokens，ePass3000 的 PKCS#11 接口符合 2.11 版本的 PKCS#11 规范。

在使用ePass3000 的PKCS#11 接口开发应用程序前，开发人员应当熟悉PKCS#11 标准。这个标准的文档可以在RSA公司的网站上自由下载，下载网址是：http://www.rsa.com/rsalabs/node.asp?id=2133

第二章 ePass3000 的CSP模块

本章讲述 ePass3000 所支持 Crypto API 接口开发的相关情况，涉及的内容包括 ePass3000 的 CSP 接口名称及支持的函数和算法实现情况。

- ePass3000 的 CSP 模块描述
- ePass3000 的 CSP 函数的算法实现描述
- ePass3000 的 CSP 函数实现描述

2.1 ePass3000 的CSP模块描述

ePass3000 通过提供标准的 CSP 模块来实现与 Crypto API 应用程序的无缝集成。ePass3000 的 CSP 模块遵从微软的 Crypto Service Provider 编程规范编写，可以兼容现有的和将来的 Crypto API 应用。

2.1.1 基本情况

类型：PROV_RSA_FULL

这种类型的 CSP 支持数字签名和数据加解密，是一种通用类型的 CSP。所有的公钥运算都是使用 RSA 算法。

名称：FEITIAN ePassNG RSA Cryptographic Service Provider

2.1.2 特征

ePass3000 的 CSP 具有以下几个特点：

- 提供了安全的 RSA 密钥对存储容器
- 提供多种分组加密算法和哈希算法
- 支持硬件实现的 RSA 运算，长度可达 2048 位
- 支持硬件随机数生成
- 支持多线程访问，多设备管理
- 支持多证书应用
- 和 PKCS#11 数据格式兼容
- 真正支持双证，即一个容器中可以有两个密钥对（AT_KEYEXCHANGE 和 AT_SIGNATURE）和对应证书
- 支持所有 Windows 平台及 Linux、Mac OS
- 无缝兼容现有 Windows 平台应用，如 Office 加解密、Internet Explorer 的 WEB 网页登录和 SSL 网站登录、Outlook（Express）的安全电子邮件等

2.2 ePass3000 的CSP所支持的算法

下表列出了 ePass3000 的 CSP 模块所支持的算法情况。

算法	默认长度(bit)	最小长度(bit)	最大长度(bit)	用途类别
CALG_RC2	40	8	1024	加解密
CALG_RC4	40	8	2048	
CALG_DES	56	56	56	
CALG_3DES	192	192	192	
CALG_SHA1	160	160	160	散列
CALG_MD2	128	128	128	
CALG_MD5	128	128	128	
CALG_SSL3_SHAMD5	288	288	288	
CALG_RSA_SIGN 或 AT_SIGNATURE	1024	512	2048	签名验证
CALG_RSA_KEYX 或 AT_KEYEXCHANGE	1024	512	2048	加解密、签名验证
CALG_SSF33*	128	128	128	加解密
CALG_SCB2*	256	256	256	加解密

2.3 ePass3000 所支持的函数实现

下表中列出了 CSP 接口各函数的支持和实现情况，“未实现”表示在 CSP 模块中有该接口，但是并没有实现，“未支持”表示在 CSP 模块中没有该函数入口。

由于 ePass3000 的 CSP 类型为 PROV_RSA_FULL，不支持下表中所列出的标有“未支持”的函数是很正常的。标示为“未实现”的函数都返回 FALSE 并置 ErrorCode 为 E_NOTIMPL。这些函数是 CSP 接口，Crypto API 应用程序不需要直接调用这些接口。

名称	描述	支持情况
连接函数		
CPAcquireContext	为应用程序创建一个上下文	已实现
CPGetProvParam	返回 CSP 相关的信息	已实现
CPReleaseContext	释放 CPAcquireContext.创建的上下文	已实现
CPSetProvParam	设置 CSP 的参数操作.	已实现
密钥生成和交换函数		
CPDeriveKey	从一个数据散列中生成一个会话密钥，它保证生成的密钥互不相同	已实现
CPDestroyKey	释放一个密钥句柄（释放后，句柄将无效，密钥将无法再被访问）	已实现
CPDuplicateKey	创建密钥的一个拷贝	未支持
CPExportKey	从 CSP 容器中导出密钥	已实现

CPGenKey	用来生成密钥或密钥对	已实现
CPGenRandom	使用随机数填充一个缓冲	已实现
CPGetKeyParam	用来得到加密操作密钥的属性	已实现
CPGetUserKey	用来获取 CSP 容器中的持久密钥对	已实现
CPImportKey	从一个 blob 中导入密钥到 CSP 容器中	已实现
CPSetKeyParam	设置密钥的属性	已实现
数据加密函数		
CPDecrypt	用来解密先前被加密的数据	已实现
CPEncrypt	用来加密明文	已实现
散列和数字签名函数		
CPCreateHash	初始化并散列输入数据	已实现
CPDestroyHash	删除一个散列对象句柄	已实现
CPDuplicateHash	创建一个散列对象的拷贝	未支持
CPGetHashParam	获取散列对象的计算结果	已实现
CPHashData	散列输入的数据	已实现
CPHashSessionKey	散列一个会话密钥而不向应用程序暴露密钥的值	未实现
CPSetHashParam	定制一个散列对象的属性	已实现
CPSignHash	签名一个散列对象	已实现
CPVerifySignature	校验一个数字签名	已实现

微软的 CSP 规范中还定义了 OffloadModExpo 这个 CSP 函数，这个函数目前不被 ePass3000 的 CSP 模块支持。

2.4 CSP各函数所支持的参数

2.4.1 CPAcquireContext

——dwFlags

支持以下值：CRYPT_VERIFYCONTEXT、CRYPT_NEWKEYSET、CRYPT_DELETEKEYSET、CRYPT_SILENT；没有处理 CRYPT_MACHINE_KEYSET 的情况；

——pszContainer

根据 dwFlags 的不同值，*pszContainer* 可以为 NULL 和“”，也可以为带有 Reader Name 的字符串，字符串长度不能超过 MAX_PATH。

2.4.2 CPGetProvParam

——dwParam

只支持以下值：PP_CONTAINER、PP_ENUMALGS、PP_ENUMALGS_EX、PP_ENUMCONTAINERS、PP_IMPTYPE、PP_NAME、PP_VERSION、PP_UNIQUE_CONTAINER、PP_PROVTYPE、

PP_SIG_KEYSIZE_INC 、 PP_KEYX_KEYSIZE_INC 、 PP_KEYSPEC ； 不 支 持 以 下 值 ： PP_KEYSET_SEC_DESCR、PP_USE_HARDWARE_RNG、等。

——dwFlags

根据对微软 CSP 的分析，当 *dwParam* 的值为 PP_ENUMALGS、PP_ENUMALGS_EX 时，dwFlags 为 CRYPT_FIRST 开始枚举，为其它值时（包括 0 和 CRYPT_NEXT）枚举下一个。当 *dwParam* 的值为 PP_ENUMCONTAINERS 时，dwFlags 为 CRYPT_FIRST（1）或（CRYPT_FIRST|CRYPT_NEXT）（3）时开始枚举，为 0 或 CRYPT_NEXT（2）时枚举下一个。dwFlags 不支持 CRYPT_MACHINE_KEYSET。当 dwParam 的值为其他情况时，不检查 dwFlags 的值。

2.4.3 CPReleaseContext

——dwFlags

该值必须为 0。

2.4.4 CPSetProvParam

——dwParam

支持以下参数值： PP_KEYEXCHANGE_PIN、PP_SIGNATURE_PIN，此时当 pbData 为 NULL 时为退出登录。

不支持其他值。

——dwFlags

不检查该值。

2.4.5 CPDeriveKey

——Algid

只支持以下算法:CALG_RC2、CALG_RC4、CALG_DES、CALG_3DES、CALG_SSF33*、CALG_SCB2*。

——dwFlags

对 于 以 下 情 况 报 错 ：（ CRYPT_CREATE_SALT | CRYPT_NO_SALT ）、 CRYPT_PREGEN 、 CRYPT_USER_PROTECTED，不支持其他值的情况。

2.4.6 CPDestroyKey

无需进一步说明。

2.4.7 CPDuplicateKey

不支持该接口。

2.4.8 CPExportKey

——dwBlobType

只支持 PUBLICKEYBLOB、SIMPLEBLOB，不支持 PRIVATEKEYBLOB、OPAQUEKEYBLOB、PLAINTEXTKEYBLOB 等情况。

——dwFlags

当 dwBlobType 为 PUBLICKEYBLOB、SIMPLEBLOB 时，dwFlags 必须为 0，其它情况忽略该参数的值。

2.4.9 CPGenKey

——Algid

支持以下算法类型值：CALG_RSA_KEYX、CALG_RSA_SIGN、AT_KEYEXCHANGE、AT_SIGNATURE、CALG_DES、CALG_RC2、CALG_RC4、（CALG_3DES_112 下一版支持）、CALG_3DES、CALG_SSF33*、CALG_SCB2*。

——dwFlags

不支持 CSP 将返回错误信息：CRYPT_CREATE_SALT、CRYPT_NO_SALT、CRYPT_PREGEN。其他情况将该参数的前两个字节的值作为要产生的密钥长度（前两个字节的值为 0 表示产生默认长度的密钥）。忽略该参数的后两个字节的值。

2.4.10 CPGenRandom

无需进一步说明

2.4.11 CPGetKeyParam

本函数只支持 CALG_RSA_KEYX、CALG_RSA_SIGN、AT_KEYEXCHANGE、AT_SIGNATURE、CALG_DES、CALG_RC2、CALG_RC4、CALG_3DES、CALG_SSF33*、CALG_SCB2*等密钥类型。

——dwParam

对于 CALG_RSA_KEYX、CALG_RSA_SIGN、AT_KEYEXCHANGE、AT_SIGNATURE 等密钥类型，该参数的值可以为：KP_PERMISSIONS、KP_CERTIFICATE、KP_BLOCKLEN、KP_KEYLEN、KP_ALGID；对于 CALG_RC2 密钥类型，该参数的值可以为：KP_BLOCKLEN、KP_EFFECTIVE_KEYLEN、KP_KEYLEN、KP_ALGID、KP_SALT；对于 CALG_RC4 密钥类型，该参数的值可以为：KP_BLOCKLEN（返回值为 0）、KP_KEYLEN、KP_ALGID、KP_SALT；对于 CALG_3DES、CALG_DES、CALG_SSF33*和 CALG_SCB2*密钥类型，该参数的值可以为：KP_BLOCKLEN、KP_KEYLEN、KP_ALGID。

——dwFlags

必须为 0。

2.4.12 CPGetUserKey

——dwParam

支持以下参数的值：AT_KEYEXCHANGE、AT_SIGNATURE、（AT_KEYEXCHANGE | AT_SIGNATURE）

2.4.13 CPImportKey

——pbData

该 keyBlob 支持 SIMPLEBLOB、PUBLICKEYBLOB、PRIVATEKEYBLOB 这三种类型。

——dwFlags

忽略该参数。

2.4.14 CPSetKeyParam

——dwParam

KP_IV 参数值适用于 CALG_RC2、CALG_DES、CALG_3DES 类型密钥，KP_EFFECTIVE_KEYLEN 参数值适用于 CALG_RC2 类型密钥，KP_SALT 和 KP_SALT_EX 适用于 CALG_RC2 和 CALG_RC4 类型密钥，KP_CERTIFICATE 适用于 CALG_RSA_KEYX、CALG_RSA_SIGN、AT_KEYEXCHANGE、AT_SIGNATURE 等类型的密钥。

——dwFlags

必须为 0。

2.4.15 CPDecrypt

支持 CALG_RSA_KEYX、AT_KEYEXCHANGE、CALG_RC2、CALG_DES、CALG_3DES、CALG_RC4、CALG_SSF33*和 CALG_SCB2*等类型的密钥。

——dwFlags

必须为 0。

2.4.16 CPEncrypt

支持 CALG_RSA_KEYX、AT_KEYEXCHANGE、CALG_RC2、CALG_DES、CALG_3DES、CALG_RC4、CALG_SSF33*和 CALG_SCB2*等类型的密钥。

——dwFlags

必须为 0。

2.4.17 CPCreateHash

——Algid

支持 CALG_MD2、CALG_MD5、CALG_SHA1 和 CALG_SSL3_SHAMD5 算法。

——dwFlags

必须为 0。

2.4.18 CPDestroyHash

无需进一步说明。

2.4.19 CPDuplicateHash

不支持该接口。

2.4.20 CPGetHashParam

——dwParam

支持 HP_ALGID、HP_HASHSIZE、HP_HASHVAL 等参数值。

——dwFlags

必须为 0。

2.4.21 CPHashData

——dwFlags

必须为 0，不支持 CRYPT_USERDATA 的参数值。

2.4.22 CPHashSessionKey

没有实现，返回 FALSE，并置 ErrorCode 为 E_NOTIMPL。

2.4.23 CPSetHashParam

——dwParam

仅支持 HP_HASHVAL 参数值。

——dwFlags

必须为 0。

2.4.24 CPSignHash

——sDescription

忽略该参数。

——dwFlags

支持 CRYPT_NOHASHOID 参数值，其它值忽略。

2.4.25 CPVerifySignature

——sDescription

忽略该参数。

——dwFlags

不支持任何值。

2.5 函数调用说明

2.5.1 基本说明

CPAcquireContext 函数是所有 CSP 函数中最先被调用的函数。上层应用通过调用这个函数来指定操作哪一个密钥容器。每一个密钥容器中同时只能保存同一种类型的一对 RSA 密钥对，和任意多个会话密钥。RSA 密钥对是可以持久保存的对象，而会话密钥则只在运行时存在。如果应用程序需要访问密钥容器中的 RSA 私钥，则 ePass3000 的 CSP 模块会要求验证用户的身份。如果应用程序希望避免弹出这个对话框则应当设置 CRYPT_SILENT 标志，但此时由于 ePass3000 不支持使用 CPSetProvParam 来设置用户身份识别信息，所以所有访问私钥和被保护数据的地方都会失败。

2.5.2 开发示例

开发人员可以在 SDK 包中的 Samples\CryptAPI 目录下找到使用 ePass3000 的 CryptAPI 接口的示例程序并可以编译调试。

有些示例程序您可能需要安装微软的 Platform SDK。

第三章 ePass3000 的PKCS#11 模块

本章讲述 ePass3000 所支持的 PKCS#11 接口开发的相关情况，涉及的内容包括 ePass3000 的 PKCS#11 接口名称及支持的函数和算法实现情况。

- ePass3000 的 PKCS#11 模块描述
- ePass3000 的 PKCS#11 函数的算法实现描述
- ePass3000 的 PKCS#11 函数实现描述

3.1 ePass3000 的PKCS#11 模块描述

ePass3000 的 PKCS#11 接口以 Win32 动态链接库(DLL)的方式提供。开发者可以静态（使用 lib 文件）和动态链接两种方式进行访问。下面列出了 ePass3000 的 PKCS#11 接口相关的文件：

文件	SDK 路径
pkcs11.h	\Include\pkcs11（由 RSA 公司提供）
pkcs11f.h	\Include\pkcs11（由 RSA 公司提供）
pkcs11t.h	\Include\pkcs11（由 RSA 公司提供）
cryptoki.h	\Include\pkcs11（由 RSA 公司提供，内有前三个头文件在 Windows 平台下所需的类型定义）
Cryptoki_ft.h	\Include\pkcs11（内有 ePass3000 扩展的算法和返回值）
auxiliary.h	\Include\pkcs11（内有 ePass3000 相关扩展函数的定义）
ngp11v211.lib	\Lib（PKCS#11 接口库的 lib 文件）

ngp11v211.dll 是 ePass3000 的核心库文件，安装后运行库位于系统目录下，它实现了 RSA PKCS#11 标准中定义的所有接口函数。如果开发人员需要使用这个接口，并且所访问的都是 PKCS#11 规范规定的标准接口和定义，则必须在工程项目中包含 cryptoki.h 头文件。如果用到 ePass3000 专有的扩展函数和算法，则只需包含 cryptoki_ft.h 即可，该头文件内部会将其它头文件包含在内。如果不想使用 LoadLibrary 方式调用 ngp11v211.dll，可以在项目中包含 ngp11v211.lib 这个文件。

3.2 ePass3000 支持的PKCS#11 对象

ePass3000 的 PKCS#11 模块支持创建和使用下列类型的对象：

对象	描述
CKO_DATA	应用程序定义的对象。对象的数据结构可由应用程序任意定义，但是数据意义的解释由应用程序负责
CKO_SECRET_KEY	对称加密算法使用的密钥

CKO_CERTIFICATE	X.509 数字证书对象
CKO_PUBLIC_KEY	RSA 公钥对象
CKO_PRIVATE_KEY	RSA 私钥对象

所有的类对象都可根据生命期长短的不同分成两大类。一类是持久存储的类对象，这类对象被保存在 ePass3000 的安全存储区域当中，直到应用程序主动删除这些对象；另一类是会话对象，这类对象只存在于运行时建立的特定会话当中，一旦会话结束，这类对象也跟着被删除。决定类对象生命期的模板属性是 CKA_TOKEN，这是个布尔值，所有的类对象都有这一属性。开发人员应当根据 ePass3000 有限的存储空间来决定如何处理对象存储的策略。只有必须持续保存的关键对象才有必要在 ePass3000 的内部存储器进行保存。

PKCS#11 的类对象除了生命期长短有分别之外，在访问权限上也有限制。所有的类对象都可根据访问权限的不同分成两大类：一类是公开对象，这类对象是任何用户都可以访问的；另一类是私有对象，这一类对象只有身份被验证的用户才有权访问。决定对象的访问限制类型的模板属性是 CKA_PRIVATE。这是个布尔值，所有的类对象都有这一属性。应用程序可根据需要决定对象应为私有对象还是公开对象。需要注意的是，私有对象的存储区域和公开对象的存储区域都是有容量限制的，而且是相互独立的。应用程序必须衡量好这两种存储区域容量的比例分配。这一比例在 ePass3000 的初始化时已经设置，不能更改了。

3.3 ePass3000 的PKCS#11 支持的算法

下表列出了所有 ePass3000 的 PKCS#11 模块支持的密码学算法：

算法	加解密	签名校验	散列	密钥对生成
CKM_RSA_PKCS_KEY_PAIR_GEN				√
CKM_RSA_PKCS	√	√		
CKM_MD2_RSA_PKCS	√	√		
CKM_MD5_RSA_PKCS	√	√		
CKM_SHA1_RSA_PKCS	√	√		
CKM_RC2_KEY_GEN				√
CKM_RC2_ECB	√			
CKM_RC2_CBC	√			
CKM_RC4_KEY_GEN				√
CKM_RC4	√			
CKM_DES_KEY_GEN				√
CKM_DES_ECB	√			
CKM_DES_CBC	√			
CKM_DES3_KEY_GEN				√
CKM_DES3_ECB	√			
CKM_DES3_CBC	√			
CKM_MD2			√	
CKM_MD5			√	

CKM_SHA_1			√	
CKM_DH_PKCS_KEY_PAIR_GEN				√
CKM_DSA_KEY_PAIR_GEN				√
CKM_DSA		√		
CKM_SSF33_KEY_GEN*				√
CKM_SSF33_CBC*	√			
CKM_SSF33_ECB*	√			
CKM_SSF33_CBC_PAD*	√			
CKM_SCB2_KEY_GEN*				√
CKM_SCB2_CBC*	√			
CKM_SCB2_ECB*	√			
CKM_SCB2_CBC_PAD*	√			

下表显示了 ePass3000 的 PKCS#11 库支持的密钥的长度：

算法	密钥长度
CKM_RSA_KEY_PAIR_GEN	512, 1024, 2048bits
CKM_RC2_KEY_GEN	1-128bytes
CKM_RC4_KEY_GEN	1-256bytes
CKM_DES_KEY_GEN	8bytes
CKM_DES3_KEY_GEN	24bytes
CKM_DH_PKCS_KEY_PAIR_GEN	128-2048bits
CKM_DSA_KEY_PAIR_GEN	512, 1024bits
CKM_SSF33_KEY_GEN*	16bytes
CKM_SCB2_KEY_GEN*	32bytes

3.4 ePass3000 支持的PKCS#11 接口函数

PKCS#11 是针对 Cryptoki 硬件的通用模型定义，不同厂商的 PKCS#11 实现会有一些细节上的差别。ePass3000 的 PKCS#11 接口模块也有一些不同于规范的地方：

有一些 PKCS#11 标准中定义的函数没有被实现，但是它们也被导出了。只是当这些函数被调用的时候，应用程序将得到返回值 CKR_FUNCTION_NOT_SUPPORT。

注：ePass3000 就相当于 PKCS#11 标注中所指的“Token”。

PKCS#11 标准中将读卡器称为“Slot”，但是由于 ePass3000 在使用的过程中并不需要读卡器，因此在 ePass3000 的实现中，slot 只是一个虚拟的设备，但对于应用程序来说，并没有什么差别。

下表列出了所有 PKCS#11 2.11 标准定义的接口函数：

名称	描述	支持情况
一般功能函数		

C_Initialize	这个函数初始化库。在调用其它库函数前必须调用此函数。唯一的例外是 C_GetFunctionList 函数	已实现
C_Finalize	当应用程序结束对库的访问时应调用此函数	已实现
C_GetInfo	得到 cryptoki 库的信息	已实现
C_GetFunctionList	得到库导出函数的指针列表	已实现
Slot 和 Token 管理函数		
C_GetSlotList	得到 slot 列表	已实现
C_GetSlotInfo	获取 slot 的信息	已实现
C_GetTokenInfo	获取 slot 中 Token 的信息	已实现
C_WaitForSlotEvent	等待 slot 事件的发生，如 Token 被插入或移除	已实现
C_GetMechanismList	获取库支持算法的列表	已实现
C_GetMechanismInfo	获取算法详细信息	已实现
C_InitToken	初始化 Token	已实现
C_InitPIN	初始化 USER PIN	已实现
C_SetPIN	修改当前登录用户的 PIN 码	已实现
会话管理函数		
C_OpenSession	在应用程序和 Token 之间建立会话	已实现
C_CloseSession	关闭会话	已实现
C_CloseAllSessions	关闭应用程序打开的所有会话	已实现
C_GetSessionInfo	获取会话的信息	已实现
C_GetOperationState	获取当前加密操作的状态	未实现
C_SetOperationState	使用从 C_GetOperationState 调用功能返回的状态恢复库的操作状态	未实现
C_Login	登录用户到 Token	已实现
C_Logout	登出用户	已实现
对象管理函数		
C_CreateObject	创建新的 Cryptoki 对象	已实现
C_CopyObject	创建对象的拷贝	未实现
C_DestroyObject	删除一个对象	已实现
C_GetObjectSize	获取对象的大小	未实现
C_GetAttributeValue	获取对象一个或多个属性	已实现
C_SetAttributeValue	修改对象的一个或多个属性	已实现
C_FindObjectsInit	初始化一次对象查找操作	已实现
C_FindObjects	继续一次对象查找操作	已实现

C_FindObjectsFinal	结束一次对象查找操作	已实现
加密函数		
C_EncryptInit	初始化一次加密操作	已实现
C_Encrypt	加密数据	已实现
C_EncryptUpdate	继续加密数据	已实现
C_EncryptFinal	结束数据加密操作	已实现
解密函数		
C_DecryptInit	初始化一次解密操作	已实现
C_Decrypt	解密输入数据	已实现
C_DecryptUpdate	继续解密操作	已实现
C_DecryptFinal	结束一次解密操作	已实现
消息散列函数		
C_DigestInit	初始化一次散列操作	已实现
C_Digest	散列输入的数据	已实现
C_DigestUpdate	继续散列操作	已实现
C_DigestKey	继续散列一个密钥	未实现
C_DigestFinal	结束散列操作	已实现
签名与消息鉴别函数		
C_SignInit	初始化一次签名操作	已实现
C_Sign	签名输入数据	已实现
C_SignUpdate	继续一次数据签名操作	已实现
C_SignFinal	结束数据签名操作	已实现
C_SignRecoverInit	初始化一次数据可恢复的签名操作	已实现
C_SignRecover	继续签名操作	已实现
校验签名和消息鉴别函数		
C_VerifyInit	初始化一次校验操作	已实现
C_Verify	校验一个签名	已实现
C_VerifyUpdate	继续校验签名	已实现
C_VerifyFinal	结束一次校验操作	已实现
C_VerifyRecoverInit	初始化一次数据可恢复校验操作	已实现
C_VerifyRecover	校验数据可恢复的签名	已实现
双功能加密函数		
C_DigestEncryptUpdate	继续一次散列并加密操作	未实现
C_DecryptDigestUpdate	继续一次解密并散列操作	未实现
C_SignEncryptUpdate	继续一次签名并加密操作	未实现
C_DecryptVerifyUpdate	继续一次解密并校验操作	未实现
密钥管理函数		
C_GenerateKey	生成密钥并创建新的密钥对象	已实现

C_GenerateKeyPair	生成密钥对并创建新的公私钥对象	已实现
C_DeriveKey	从基础密钥中衍生出一个密钥(目前只支持 CKM_DH_PKCS_DERIVE)	已实现
C_WrapKey	包装一个私钥或秘密密钥（目前支持算法 CKM_RSA_PKCS，CKM_RSA_X_509）	已实现
C_UnwrapKey	解包一个私钥或秘密密钥（目前支持算法 CKM_RSA_PKCS，CKM_RSA_X_509）	已实现
随机数生成函数		
C_SeedRandom	加入随机种子到生成器中	已实现
C_GenerateRandom	生成随机数	已实现
并行功能管理函数		
C_GetFunctionStatus	这个函数已经废弃	未实现
C_CancelFunction	这个函数已经废弃	未实现

注：带*号的算法是国密办指定的自定义算法。