**第** **9** **章**

**CSP和CryptoAPI**

**9.1 什么是CSP**

CSP 是Windows平台的加密服务提供者 (Cryptographic Service Provider),它是真正执 行密码运算的独立模块。物理上一个CSP 由两部分组成：一个是动态链接库，另一个是签名 文件。其中签名文件保证密码服务提供者经过了认证，以防出现攻击者冒充CSP。 若加密算 法由硬件实现，则CSP 还包括硬件装置。

Microsoft 通过捆绑RSA Base Provider(RST基础提供者),在操作系统中提供一个CSP, 使 用RSA 公司的公钥加密算法，更多的CSP 可以根据需要增加到应用中。Windows 2000以 后自带了多种不同的CSP。

CSP是Windows密码服务系统的底层实现，它通过统一的编程接口CryptoAPI面向用户， 提供编程调用服务。

 **CryptoAPI简介**

当前，有关加密的API国际标准有如下4类：

(1)GSS-API(Generic Security Services API)

(2)CDSA

(3)RSA PKCS#11

(4)微软CryptoAPI

在 Windows 领域，微软CryptoAPI 是重要的加密标准，所以必须学。微软的CryptoAPI 是Win32 平台下为应用程序开发者提供的数据加解密和证书服务的编程接口。CryptoAPI提供 了很多和信息安全相关的函数，如编码、解码、加密、解密、哈希、数字证书、证书管理、证 书存储等。CryptoAPI的编程模型与Windows系统的图形设备接口GDI 比较类似，其中加密 服务提供者(CSP) 等同于图形设备驱动程序，加密部件(可选)等同于图形部件，其上层的 应用程序也类似，都不需要同设备驱动程序和硬件直接打交道。

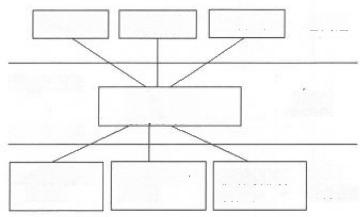


第 9 章 CSP 和 CryptoAPI



**CSP服务体系**

微软 Windows 加密系统由不同的元素组成。这三个可执行部分包括应用程序(包含 CryptoAPI)、操作系统(OS) 和加密服务提供者(CSP) 。应用程序通过加密API(CryptoAPI) 与OS 通信。操作系统通过加密服务提供程序接口(CryptoSPI) 与 CSP 通信。图9-1显示了 这些概念。



应用程序A 应用程序B 应用程序C 应用层

CryptoAPI

CryptoAPI 操作系统

CryptoSPI

微软RSA服务 CSP#1

防篡改服务 CSP#3

智能卡服务 CSP#2

服务 提供层

系统层

图9 - 1

从系统调用层次来看，分为相互独立的三层(参看上图的服务分层体系):

(1)最底层是加密服务提供层，即具体的一个CSP, 它是加密服务提供机构提供的独立 模块，担当真正的数据加密工作，包括使用不同的加密和签名算法产生密钥、交换密钥、进行 数据加密以及产生数据摘要、数字化签名。它独立于应用层和操作系统，并提供通用的 SPI 编程接口与操作系统层进行交互。有些CSP 还会使用硬件进行加密工作，以达到更为安全的 效果。

(2)中间层，即操作系统(OS) 层，在此是指具体的 Windows 操作系统平台，在 CSP 体系中，它为应用层提供统一的API 接口，为加密服务提供层提供SPI 接口，操作系统层为 应用层隔离了底层CSP 和具体的加密实现细节，用户可独立各个CSP 进行交互，它担当一定 的管理功能，包括定期验证CSP 等。

(3)应用层，也就是任意用户进程或线程具体通过调用操作系统层提供的CryptoAPI使 用加密服务的应用程序。

根据CSP 服务分层体系，应用程序不必关心底层CSP 的具体实现细节，利用统一的API 接口进行编程，而由操作系统通过统一的SPI 接口来与具体的加密服务提供者进行交互，由其 他的厂商根据服务编程接口SPI 实现加密、签名算法，有利于实现数字加密与数字签名。

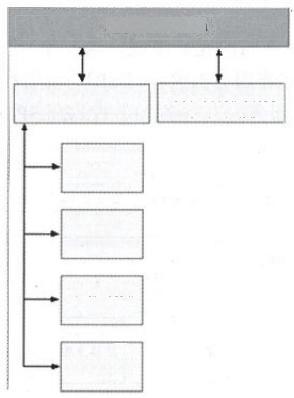
应用程序中要实现数字加密与数字签名时，一般是调用微软提供的应用程序编程接口 CryptoAPI。应用程序不能直接与CSP 通信，只能通过CryptoAPI操作系统界面过滤后，经过 CryptoSPI 系统服务接口与相应的CSP 通信。CSP 才是真正实现所有加密操作的独立模块。

CSP 是执行实际加密操作的独立单元(CryptoAPI只是面向用户的接口，真正的加解密运 算在CSP) 。CSP 通过Coredll.dll与应用程序通信。CSP 负责创建和销毁密钥，并使用它们执



**Windows C/C++加密解密实战**

行各种加密操作。每个CSP 都提供了CryptoAPI的不同实现。有些提供更强大的加密算法， 而另一些包含硬件加解密实现。图9-2显示了应用程序Coredll.dll和CSP 之间的关系。



Application

Protected store API or DPAPI

RSABASE CSP

RSAENH CSP

Snartcard CSP

Other CSP

CryptoAPI

(Coredl1.dl1)

图9- 2

CSP 至少由动态链接库 (DLL) 和签名文件组成。签名文件确保操作系统识别 CSP。操 作系统定期验证此签名，以验证CSP 未被篡改。

加密标准常被划分为不同的系列组，每个系列都包含一组数据格式和协议。即使使用相同 的算法，两个家族也经常使用不同的密码模式、密钥长度和默认模式。在CryptoAPI中，每个 CSP 类型代表一个不同的家族。默认情况下，当应用程序连接到特定类型的 CSP 时，每个 CryptoAPI函数都以与CSP 类型对应的系列指定的方式操作。表9-1显示了由应用程序选择的 CSP 类型指定的项。

**表9-** **1** **由应用程序选择的CSP 类型指定的项**

|  |  |
| --- | --- |
| **CSP类型属性** | **描** **述** |
| 密钥交换算法 | 指定一个密钥交换算法，特定类型的每个CSP都必须实现此算法，应用程序 指定密钥交换算法的唯一方法是选择适当的CSP类型 |
| 数字签名算法 | 这与密钥交换算法相同，每个CSP类型指定一个数字签名算法 |
| 密钥二进制大对象格式 | 指定导出键的格式。密钥可以从CSP导出为密钥二进制大对象格式，以增强 CSP之间传输时的安全性 |
| 数字签名格式 | 规定了特定的数字签名格式，这确保由CSP生成的签名可以由同一类型的任 何CSP验证 |
| 会话密钥派生方案 | 指定用于派生会话密钥的方法 |
| 密钥长度 | 指定密钥长度 |
| 默认模式 | 指定各种选项的默认模式，例如块加密密码模式或块加密填充方法 |



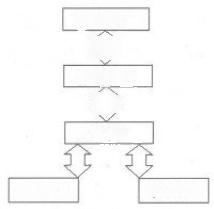
**第** **9** **章** **CSP 和** **CryptoAPI**

**9.4 CSP的组成**

CSP 为 Windows 平台上加解密运算的核心层实现，是真正执行加密工作的独立模块。CSP 与Windows的接口以 DLL 形式实现。

按照CSP的不同实现方法，可分为纯软件实现与带硬件的实现，其中带硬件的实现CSP

按照硬件芯片不同，可以分为使用智能卡芯片(内置加密算法)的加密型和不使用智能卡芯片 的存储型两种，与计算机的接口现在一般都用USB, 所以把CSP 的硬件部分称为USB Key。 物理上一个CSP 由两部分组成：动态链接库和签名文件。CSP 逻辑上的组成如图9-3所示。



SPI接口



算法实现



密钥库

密钥容器 密钥容器

图9-3

(1)微软提供的 SPI 接口函数实现。在微软提供的SPI 接口中共有23个基本密码系统函 数，由应用程序通过CAPI 调用，CSP 必须支持这些函数，这些函数提供了基本的功能。

(2)加密签名算法实现。如果是纯软件实现的CSP 与用存储型的USB Key 实现的CSP, 这些函数就在CSP 的 DLL 或辅助DLL 中实现；带硬件设备实现的CSP, 并且使用加密型的 USB Key,CSP的动态库就是一个框架，一般的函数实现在CSP 的动态库中，而主要函数的 核心在硬件中实现。在CSP 的动态库中只是函数的框架，如加/解密、散列数据、验证签名等， 这是因为私钥一般不导出，这些函数的实现主要在硬件设备中，保密性好。

(3)CSP 的密钥库及密钥容器。每一个加密服务提供程序都有一个独立的密钥库，它是 一个CSP 内部数据库，此数据库包含一个和多个分属于每个独立用户的容器，每个容器都用 一个独立的标识符进行标识。不同的密钥容器内存放不同用户的签名密钥对、交换密钥对以及 x.509数字证书。出于安全性考虑，私钥一般不可以被导出。带硬件实现的CSP,CSP 的密钥 库及密钥容器放在硬件存储器中，纯软件的CSP实现放在硬盘上的文件中。

 **CryptoAPI体系结构**

密码服务编程接口CryptoAPI体系架构由五大部分组成：

(1)基本加密函数：用于选择 CSP、建立CSP 连接、产生密钥、交换及传输密钥等操作。



**Windows C/C++加密解密实战**

(2)简单的消息函数：用于消息处理，比如消息编解码、消息加解密、数字签名及签名 验签等操作。它是把多个底层消息函数包装在一起以完成某个特定任务，方便用户使用。

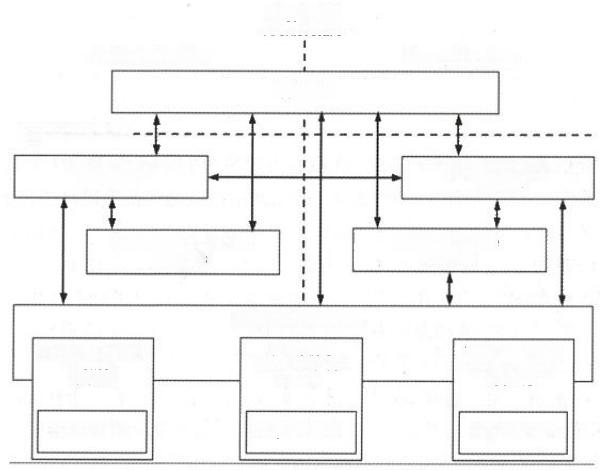
(3)底层消息函数：底层消息函数对传输的PKCS#7 数据进行编码，对接收到的PKCS#7 数据进行解码，并且对接收到的消息进行解码和验证。它可以实现简单消息函数可以实现的所 有功能，且提供更大的灵活性，但一般需要更多的函数调用。

(4)证书编解码函数：用于数据加密、解密、哈希等操作，创建和校验数字签名操作， 实现证书请求和证书扩展编码和解码操作。

(5)证书库管理函数：用于证书管理等操作。这组函数用于管理证书、证书撤销列表和 证书信任列表的使用、存储、获取等。

其中前三者可用于对敏感信息进行加密或签名处理，可保证网络传输信息交流中的私有性； 后两者通过对证书的使用，可保证网络信息交流中的认证性。

我们可以使用图9-4所示的CryptoAPI体系结构。



**CryptoAPI Architecture**

**Certifcate Functions**

Appication

CryptoAPI Interface

Certificate Store Functons

CertificateEncode/Decode Funcbons

Cryptographic Functons

Tamper Resistant

ServiceProvider CSP #3

Key Database

Microsoft RSA Base Provider

CSP#1

Smart Card

Service Provider

CSP#2

LowLevel Message Functions

Simplifed Message Functions

**Message Functions**

Key Database

Key Database

图9-4



**CryptoAPI 调用底层** **CSP服务方式**

微软CryptoAPI 从两方面保证安全通信：保密性和验证。

CryptoAPI函数调用底层CSP 函数时，首先使用函数CryptAcquireContext给出欲选择CSP 的名称参数和类型参数，该函数返回一个指向被选择的CSP 的句柄。CSP 有一个密钥库。密 钥库用于存储密钥，每个密钥库包括一个或多个密钥容器(Key Containers)。每个密钥容器



**第** **9** **章** **CSP 和CryptoAPI**

中包含属于一个特定用户的所有密钥对。每个密钥容器被赋予唯一的名字，以这个名字做函数 CryptAcquireContext参数，从而获得指向这个密钥容器的句柄。CSP 将永久保存密钥容器，包 括保存每个密钥容器中的公/私钥对(会话密钥除外)。在交换密钥时，或密钥需要离开 CSP (导出密钥)时，就存在选择什么样的数据结构存储密钥的问题。微软CryptoAPI采用KeyBlob 数据结构存储离开了CSP 内部的密钥。密钥总是在CSP 内部被安全地保存，应用程序只能通 过句柄访问密钥，而KeyBlob 例外。当使用CryptExportKey函数从CSP 中导出密钥时，KeyBlob 被创建。之后某一时间，使用CryptImportKey函数将密钥导入其他CSP 中(不同机器上的不

同 CSP)。 因此，KeyBlob 是在不同CSP 之间安全地传送密钥载体。KeyBlob 由一个标准信 息头和位于信息头之后一段表示密钥本身的数据组成。应用程序不访问 KeyBlob 内部，而是 把KeyBlob 当作一个透明对象。

由于公/私钥对的私钥部分需要绝对保密，因此私钥要用对称加密算法加密。加密 PrivateKeyBlob时，除了BlobHeader 之外，所有部分都要加密。但加密所用的算法和密钥(或 密钥参数)不与该KeyBlob存储在一起，应用程序负责管理这些信息。



**CrpytoAPI** **的基本功能**

利 用CryptoAPI, 开发者可以给基于Windows的应用程序添加安全服务，包括ANS.1编 码/解码、数据加解密、身份认证、数字证书管理，同时支持PKI、对称密码技术等。CrpytoAPI 基本功能有：

(1)密钥管理

在 CryptoAPI中，支持两种类型的密钥：会话密钥和公私钥对。会话密钥也称为对称密钥， 用于对称密钥算法。为了保证密钥的安全性，在CryptoAPI中，这些密钥都保存在CSP 内部， 用户可以通过 CryptExportKey 以加密密钥形式导出。公私钥用于非对称加密算法。非对称加 密算法主要用于加解密会话密钥和数字签名。在CryptoAPI中，一般来说，大多数 CSP 产生 的密钥容器包含两对密钥对，一对用于加密会话密钥，称为交换密钥对；另一对用于产生数字 签名，称为签名密钥对。在CryptoAPI中，所有的密钥都存储在 CSP 中 ，CSP 负责密钥的创 建、销毁、导入导出等操作。

(2)数据编解码

CryptoAPI 采用的编码方式为ASN.1, 编码规则为DER, 表示发送数据时先把数据抽象为 ASN.1 对象，然后使用DER 编码规则把 ASN.1 对象转化为可传输的0、1串；接收方接收到 数据后，利用DER 解码规则把0、1串转化为ASN.1 对象，然后把ASN.1对象转化为具体应 用支持的数据对象。

(3)数据加解密

在 CryptoAPI中约定加密较大的数据块时，采用对称密钥算法。通过其封装好的加解密函



**Windows C/C++加密解密实战**

数来实现数据加解密操作。

(4)哈希和数字签名

哈希和数字签名一般用于数据的完整性校验和身份鉴别。在CryptoAPI中，通过其封装好 的哈希与数字签名函数来实现相关操作。微软公司提供的CSP 产生的数字签名遵循RSA 标准 (PKCS#6)。

(5)数字证书管理

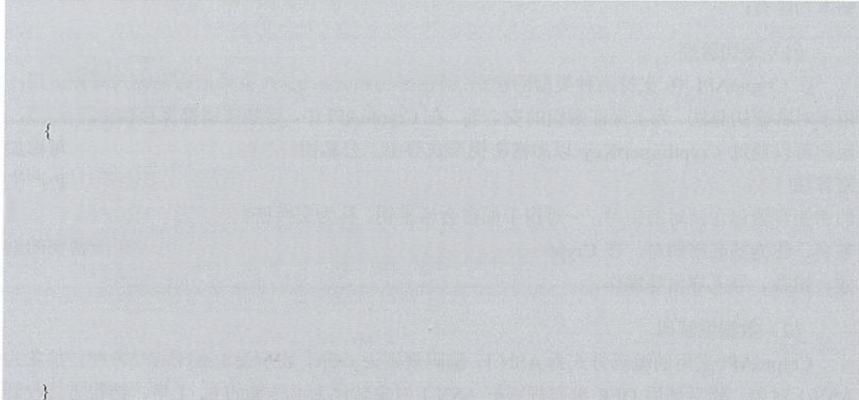
数字证书主要用于安全通信中的身份鉴别。在CryptoAPI中，对数字证书的使用管理函数 分为证书与证书库函数、证书验证函数两大部分。



**搭建CryptoAPI 开发环境**

本节我们使用VC2017 来开发CryptoAPI应用程序，因此只需在VC 工程中包含相关头文 件即可。下面我们来看一个例子，例子中只调用了一个CryptoAPI函数，如果调用成功，就说 明开发环境搭建成功。

【例9.1】第一个CryptoAPI 应用程序



(1)打开VC 2017,新建一个控制面板工程，工程名是test。

(2)在 test.cpp中输入代码如下：

#include "pch.h"

#include <iostream>

#include<windows.h>

#include <wincrypt.h> //这里面声明了CryptoAPI 的库函数

int main()

HCRYPTPROV hCryptProv;

if(!CryptAcquireContext(&hCryptProv,NULL,NULL,PROV\_RSA\_FULL,CRYPT\_SILENT| CRYPT\_VERIFYCONTEXT)

{

printf("CryptAcquireContext failed:0x%x",GetLastError()

return -1; }

std::cout<<"CryptAcquireContext OK\n" CryptReleaseContext(hCryptProv,0);

return 0;

值得注意的是，windows.h 必须在 wincrypt.h 前面包含。在代码中，我们调用了两个 CryptoAPI库函数CryptAcquireContext和CryptReleaseContext。其 中 ，CryptAcquireContext用 来连接CSP, 获得指定CSP的密钥容器的句柄；CryptReleaseContext释放由CryptAcquireContext 得到的句柄。



**第** **9** **章** **CSP 和CryptoAPI**

(3)保存工程并运行，运行结果如图9-5所示。



图9-5



**基本加密函数**

CSP 是真正实行加密的独立模块，它既可以由软件实现，又可以由硬件实现。但是它必 须符合CryptoAPI 接口的规范。每个CSP 都有一个名字和一个类型。每个CSP 的名字是唯一 的，这样便于CryptoAPI找到对应的CSP。目前已经有9种CSP 类型，并且还在增长。表9-2 列出了它们支持的密钥交换算法、签名算法、对称加密算法和Hash 算法。

**表9-2** **基本加密函数**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CSP类型** | **密钥交换算法** | **签名算法** | **对称加密算法** | **Hash算法** |
| PROV\_RSA\_FULL | RSA | RSA | RC2 RC4 | MD5 SHA |
| PROV\_RSA\_SIG | none | RSA | none | MD5 SHA |
| PROV\_RSA\_SCHANNEL | RSA | RSA | RC4 DES  Triple DES | MD5 SHA |
| PROV\_DSS | DSS | none | DSS | MD5 SHA |
| PROV\_DSS\_DH | DH | DSS | CYLINK\_MEK | MD5 SHA |
| PROV\_DH SCHANNEL | DH | DSS | DES  Triple DES | MD5 SHA |
| PROV\_FORTEZZA | KEA | DSS | Skipjack | SHA |
| PROV MS EXCHANGE | RSA | RSA | CAST | MD5 |
| PROV SSL | RSA | RSA | Varies | Varies |

基本加密函数为开发加密应用程序提供了足够灵活的空间。所有CSP 的通信都是通过这 些函数进行的。一个CSP 是实现所有加密操作的独立模块。在每一个应用程序中至少需要提 供一个CSP 来完成所需的加密操作。如果使用一个以上的CSP, 在加密函数调用中就要指定 所需的 CSP。微软基本加密提供者(Microsoft Base Cryptographic Provider)是默认绑定到 CryptoAPI 中的。如果没有指定其他 CSP, 这 个CSP 就是默认的。

每一个CSP 对CryptoAPI 提供了一套不同的实现。一些CSP提供了更加强大的加密算法；



**Windows** **C/C++加密解密实战**

另外一些CSP 包含对硬件的支持，比如智能卡；还有一些CSP 偶尔和使用者直接通信，比如 数字签名就使用了用户的签名私钥。

基本加密函数包含这几类：服务提供者函数、密钥的产生和交换函数、编码/解码函数、 数据加密/解密函数、哈希函数和数字签名函数。

**9.9.1** **服务提供者函数**

服务提供者函数是很重要的基本加密函数。应用程序使用服务提供者函数来连接和断开一 个CSP 。表9-3所示就是主要的服务提供者函数。

**表9-3** **主要的服务提供者函数**

|  |  |
| --- | --- |
| **主要的服务提供者函数** | **说** **明** |
| CryptAcquireContext | 获得指定CSP的密钥容器的句柄 |
| CryptContextAddRef | 增加一个应用计数 |
| CryptEnumProviders | 枚举当前计算机中的CSP |
| CryptEnumProviderTypes | 枚举CSP的类型 |
| CryptGetDefaultProvider | 对于指定CSP类型的默认CSP |
| CryptGetProvParam | 得到一个CSP的属性 |
| CryptInstallDefaultContext | 安装先前得到的HCRYPTPROV上下文作为当前默认的上下文 |
| CryptReleaseContext | 释放由CryptAcquireContext得到的句柄 |
| CryptSetProvider和CryptSetProviderEx | 为指定CSP类型指定一个默认的CSP |
| CryptSetProvParam | 指定一个CSP的属性 |
| CryptUninstallDefaultContext | 删除先前由CryptInstallDefaultContext安装的默认上下文 |

**1.连接CSP的函数CryptAcquireContext**

该函数用于获得指定CSP 的密钥容器的句柄。该函数声明如下：

BO0L CryptAcquireContext( HCRYPTPROV\*phProv,

pszContainer, pszProvider,

LPCTSTR LPCTSTR

DWORD dwProvType DWORD dwFlags)

● phProv:[out]所获取的CSP 的句柄指针。

● pszContainer:[in]指向密钥容器的字符串指针，用于指定在所要寻找的CSP 中所寻找 的密钥容器的名字。如果dwFlags 为 CRYPT\_VERIFYCONTEXT,pszContainer 就 必 须为NULL。

● pszProvider:[in]指定所寻找的CSP 的名字。

● dwProvType:[in]请求的CSP 的类型。

● dwFlags:[in]标记请求的CSP 的用途，该参数的可选值和含义如表9-4所示。



**第** **9** **章** **CSP 和** **CryptoAPI**

**表9-4** **dwFlags 参数的可选值和含义**

|  |  |
| --- | --- |
| **值** | **含** **义** |
| CRYPT\_VERIFYCONTEXT | 此选项指出应用程序不需要使用公钥/私钥对，如程序只执行哈希和对称 加密，只有程序需要创建签名和解密消息时才需要访问私钥 |
| CRYPT\_NEWKEYSET | 使用指定的密钥容器名称创建一个新的密钥容器。如果pszContainer为 NULL,密钥容器就使用默认的名称创建 |
| CRYPT MACHINE\_KEYSET | 由此标志创建的密钥容器只能由创建者本人或有系统管理员身份的人使用 |
| CRYPT\_DELETEKEYSET | 删除由pszContainer指定的密钥容器。如果pszContainer为NULL,默认 名称的容器就会被删除。此容器里的所有密钥对也会被删除 |
| CRYPT SLIENT | 应用程序要求CSP不显示任何用户界面 |

如果函数执行成功就返回TRUE, 否则返回FALSE, 此时可以用函数 GetLastError来获取 错误码。

这个函数用来取得指定CSP 密钥容器句柄，以后任何加密操作都是针对此CSI 句柄而言 的。函数首先查找由 dwprovtype 和 pszprovider 指定的CSP, 如果找到了CSP, 函数就查找由 此 CSP 指定的密钥容器。由适当的dwflags 标志，这个函数就可以创建和销毁密钥容器，如 果不要求访问私钥，也可以提供对CSP 临时密钥容器的访问，比如CryptAcquireContext(&hProv, NULL,NULL,PROV\_RSA\_FULL,0));。

**2.枚举CSP的函数** **CryptEnumProviders**

该函数用于枚举计算机上的所有 CSP。 此函数可以得到第一个或下一个可用的 CSP。如 果循环调用，可以得到计算机上所有可用的CSP。 函数声明如下：



BOOLWINAPI CryptEnumProviders(

DWORD dwIndex,

DWORD\*pdwReserved, DWORD dwFlags

DWORD\*pdwProvType, LPTSTRpszProvName,

DWORD\*pcbProvName );

● dwIndex:[in]枚举下一个CSP的索引。

● pdwReserved:[in]保留参数，必须为NULL。

● dwFlags:[in]为未来的保留参数，必须设为0。

● pdwProvType:[out]CSP的类型。

● pszProvName:[out] 指向接收CSP 名称的缓冲区字符串指针。此指针可为NULL, 用来得到字符串的大小。

● pcbProvName:[in/out]指明pszProvName 所指字符串的长度。

如果函数执行成功就返回TRUE, 否则返回FALSE, 此时可以用函数GetLastError 来获取 错误码。

下面来看一个例子，枚举本机上的所有CSP。



**Windows** **C/C++加密解密实战**

【例9.2】枚举计算机上的所有CSP

(1)打开VC2017, 新建一个控制面板工程，工程名是 test。

(2)在工程中打开test.cpp中，输入代码如下：

#include "pch.h"

#include<stdio.h>

#include<string.h>

#include<windows.h> #include<WinCrypt.h> int main()

{

DWORD dwIndex=0; DWORD dwType

DWORD cbNameLen LPWSTR pszName; DWORDi=0;

//查看机器中所有的CSP

pszName=(LPTSTR)LocalAlloc(LMEM\_ZEROINIT,256); cbNameLen=256;

while(CryptEnumProviders(dwIndex++,NULL,0,&dwType,pszName,&cbNameLen)) {

wprintf(L"dwType=%2.0d,len=%d CSPName=%s\n",dwType,cbNameLen,pszName);cbNameLen=

256://这里要加这个

LocalFree(pszName) return 0;

}

在代码中，我们通过循环调用CryptEnumProviders函数来获取当前计算机的所有CSP。

(3)保存工程并运行，运行结果如图9 - 6所示。



**Thierosoft Visoal stadio调试控训台** 口回×

Iype-1.len-86 CSPNome-Hicrosoft Base Gryptographic Provid日

r -13.len-122 CSPNHane-Hicrosoft Base DSS and Diffie-Hei

doIype=3.1en-84 CSPNane-Hicrosoft Basa DSS Gryptographic Pr doIype-1.1en-84 GSPHane=Hicrosoft Base Snart Card Grypto Pr doIype-18,len-90 GSPHane-Hicrosoft DH SChannel Gryptographic doIype=1.len=94 CSPHane -Microsoft Enhanced Gryptographic Pr doIype 13.len=130 CSPHane-Hicrosoft Enhanced DSS and Diffia-

Iv-en=188 GSPHane=Hicrosoft Enhanced RSA and AE8 Grg

dwIype-12.1en=92 CSPHane=Hicrosoft FSA 8Channol Cryptogpraphi doIype=1.1en=88 CSPNane-Hicrosoft Strong Gryptographic Prov



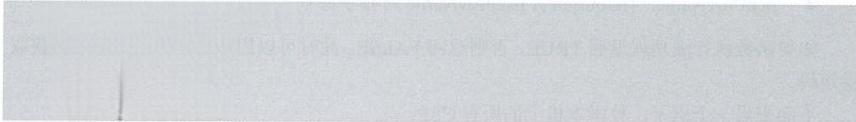


图9-6

**3.获取默认CSP** **的函数CryptGetDefaultProvider**

函数CryptGetDefaultProvider用于获取系统默认的CSP。该函数声明如下：

BO0L CryptGetDefaultProvider( DWORD dwProvType

DWORD\*pdwReserved.

DWORD dwFlags

LPSTRpszProvName,



**第** **9** **章** **CSP 和** **CryptoAPI**

DWORD \*pcbProvName);

● dwProvType:[in]要找的默认CSP 的类型。

● pdwReserved:[in]该参数保留，赋NULL即可。

● dwFlags:[in]标志位。

● pszProvName:[out]指向存放CSP 名称的缓冲区的字符串指针。

● pcbProvName:[in/out]输入时，表示 pszProvName 的大小；输出时，表示实际CSP 名称的大小。

当函数执行成功时返回TRUE, 否则返回FALSE, 此时可以用函数GetLastError来获取错 误码。

**4.设置默认** **CSP** **的函数CryptSetProvider**

函 数CryptSetProviderong 用来指定当前用户默认的加密服务提供程序。函数声明如下： BOOL CryptSetProvider(LPCTSTRpszProvName,DWORD dwProvType);

● pszProvName:[in]指向存放CSP 名称的字符串缓冲区。

● dwProvType:[in]表示CSP 类型。

如果函数执行成功就返回TRUE, 否则返回FALSE, 此时可以用函数GetLastError 来获取 错误码。

**5** **.** **获取CSP** **参数属性的函数CryptGetProvParam**

函数CryptGetProvParam用于获取CSP各种参数属性，函数声明如下：



BOOL CryptGetProvParam(HCRYPTPROVhProv,DWORD dwParam,BYTE\*pbData,DWORD\* pdwDataLen,DWORD dwFlags);

● hProv:[in]CSP 句柄。

● dwParam:[in]指定查询的参数，可选值如表9-5所示。

**表9-5** **dwParam可选值**

|  |  |
| --- | --- |
| **参数名** | **作** **用** |
| PP CONTAINER | 指向密钥名称的字符串 |
| PP\_ENUMALGS | 不断地读出CSP支持的所有算法 |
| PP\_ENUMALGS EX | 比PP\_ENUMALGS获得更多的算法信息 |
| PP ENUMCONTAINERS | 不断地读出CSP支持的密钥容器 |
| PP IMPTYPE | 指出CSP怎样实现 |
| PP NAME | 指向CSP名称的字符串 |
| PP VERSION | CSP的版本号 |
| PP\_KEYSIZE\_INC | AT\_SIGNATURE的位数 |
| PP KEYX KEYSIZE\_INC | AT KEYEXCHANGE的位数 |
| PP KEYSET\_SEC\_DESCR | 密钥的安全描述符 |
| PP\_UNIQUE\_CONTAINER | 当前密钥容器的唯一名称 |



**Windows** **C/C++加密解密实战**

**(续表)**

|  |  |
| --- | --- |
| **参数名** | **作用** |
| PP PROVTYPE | CSP类型 |
| PP ENUMALGS EX | 比PP\_ENUMALGS获得更多的算法信息 |
| PP ENUMCONTAINERS | 不断地读出CSP支持的密钥容器 |

● pbData:[out]指向接收数据的缓冲区指针。

● pdwDataLen:[in/out]指出pbData的数据长度。

● dwFlags:[in]如果指定PP\_ENUMCONTAINERS,就指定CRYPT\_MACHINE\_KEYSET。

如果函数执行成功就返回TRUE, 否则返回FALSE, 此时可以用函数 GetLastError来获取 错误码。

下面的代码演示了CryptGetProvParam的使用：

HCRYPTPROV hCryptProv

BYTE pbData[1000];

DWORD cbData //默认CSP 名称 cbData=1000;

if(CryptGetProvParam(hCryptProv,PP\_NAME,pbData,&cbData,0) {

printf("CryptGetProvParam succeeded.\n") printf("Provider name:%s\n",pbData);

}

else {

printf("Error reading CSP name.\n");

exit(1);

}

cbData=1000;

if(CryptGetProvParam(hCryptProv,PP\_CONTAINER,pbData,&cbData,0)) {

printf("CryptGetProvParam succeeded.\n") printf("Key Container name:%s\n",pbData);

了

else

{

print("Error reading key container name.\n") exit(1);

**6** **.设置CSP** **参数的函数** **CryptSetProvParam**

函数CryptSetProvParam用于设置CSP的各种参数，函数声明如下：

BO0L CryptSetProvParam(HCRYPTPROVhProv,DWORD dwParam,BYTE\*pbData,DWORD dwFlags)

● hProv:[in]CSP句柄。

● dwParam:[in]指定设置的参数，可选值如表9 - 6所示。



第 9 章 CSP 和 CryptoAPI

**表9-6** **dwParam可选值**

|  |  |
| --- | --- |
| **值** | **说** **明** |
| PP\_CLIENT\_HWND | 设置Windows句柄 |
| PP KEYSET SEC DESCR | 密钥的安全描述 |
| PP USE HARDWARE RNG | 指出硬件是否支持随机数发生器 |

● pbData:[in]指向设置数据的缓冲区指针。

● dwFlags:[in]标志位。

如果函数执行成功就返回TRUE, 否则返回FALSE, 此时可以用函数GetLastError 来获取 错误码。

**7.断开CSP的函数CryptReleaseContext**

函数CryptReleaseContext用于断开CSP并释放CSP句柄。该函数和函数CryptAcquireContext 对应起来使用。该函数声明如下：

BO0L CryptReleaseContext(HCRYPTPROVhProv,DWORD dwFlags);

● hProv:[in]CSP句柄。

● dwFlags:[in]标志位，保留参数，必须为0。

如果函数执行成功就返回TRUE, 否则返回FALSE, 此时可以用函数 GetLastError来获取 错误码。

**9.9.2** **密钥的产生和交换函数**

密钥产生函数用于创建、配置和销毁加密密钥。它们也用于和其他用户交换密钥。表9-7 所示就是一些主要的函数。

**表9-7** **主要的密钥产生和交换函数**

|  |  |
| --- | --- |
| CryptAcquireCertificatePrivateKey | 对于指定证书上下文得到一个HCRYPTPROV句柄和dwKeySpec |
| CryptDeriveKey | 从一个密码中派生一个密钥 |
| CryptDestoryKey | 销毁密钥 |
| CryptDuplicateKey | 制作一个密钥和密钥状态的精确复制 |
| CryptExportKey | 把CSP的密钥做成BLOB传送到应用程序的内存空间中 |
| CryptGenKey | 创建一个随机密钥 |
| CryptGenRandom | 产生一个随机数 |
| CryptGetKeyParam | 得到密钥的参数 |
| CryptGetUserKey | 得到一个密钥交换或签名密钥的句柄 |
| CryptImportKey | 把一个密钥BLOB传送到CSP中 |
| CryptSetKeyParam | 指定一个密钥的参数 |



**Windows C/C++加密解密实战**

**9.9.3** **编码/解码函数**

有一些编码/解码函数可以用来对证书、证书撤销列表、证书请求和证书扩展进行编码和 解码。表9-8所示就是这些编码/解码函数。

**表9-8编码/解码函数**

|  |  |
| --- | --- |
| **函** **数** | **说** **明** |
| CryptDecodeObject | 对lpszStructType结构进行解码 |
| CryptDecodeObjectEx | 对lpszStructType结构进行解码，此函数支持内存分配选项 |
| CryptEncodeObject | 对lpszStructType结构进行编码 |
| CyptEncodeObjectEx | 对lpszStructType结构进行编码，此函数支持内存分配选项 |

**9.9.4** **数据加密/解密函数**

这些函数支持数据的加密/解密操作。CryptEncrypt 和 CryptDecrypt 要求在被调用前指定 一个密钥。这个密钥可以由CryptGenKey 、CryptDeriveKey 或 CryptimportKey 产生。创建密 钥时要指定加密算法。CryptSetKeyParam 函数可以指定额外的加密参数，如表9-9所示。

**表9-9** **CryptSetKeyParam函数可以指定的额外的加密参数**

|  |  |
| --- | --- |
| **函** **数** | **说** **明** |
| CryptDecrypt | 使用指定加密密钥来解密一段密文 |
| CryptEncrypt | 使用指定加密密钥来加密一段明文 |
| CryptProtectData | 执行对DATA\_BLOB结构的加密 |
| CryptUnprotectData | 执行对DATA\_BLOB结构的完整性验证和解密 |

**9.9.5** **哈** **希** **和** **数** **字** **签** **名** **函** **数**

这些函数在应用程序中完成计算哈希、创建和校验数字签名，如表9-10所示。

**表9-10** **哈希和数字签名函数**

|  |  |
| --- | --- |
| 函 数 | 说 明 |
| CryptCreateHash | 创建一个空哈希对象 |
| CryptDestoryHash | 销毁一个哈希对象 |
| CryptDuplicateHash | 复制一个哈希对象 |
| CryptGetHashParam | 得到一个哈希对象参数 |
| CryptHashData | 对一块数据进行哈希，把它加到指定的哈希对象中 |
| CryptHashSessionKey | 对一个会话密钥进行哈希，把它加到指定的哈希对象中 |
| CryptSetHashParam | 设置一个哈希对象的参数 |
| CryptSignHash | 对一个哈希对象进行签名 |
| CryptVerifySignature | 校验一个数字签名 |



**第** **9** **章** **CSP 和** **CryptoAPI**

【例9.3】加解密一个文件

(1)打开VC 2017,新建一个控制面板工程，工程名是test。

( 2 ) 在D 盘新建一个文本文档，并输入一行文本，比如hello。接着，在VC 工程中打开 test.cpp,并输入代码如下：

#include "pch.h

#define \_CRT\_SECURE\_NO\_DEPRECATE

#define \_CRT\_SECURE\_NO\_WARNINGS //为了使用 fopen

#include<windows.h> #include<stdio.h>

#include <stdlib.h>

#include<wincrypt.h>

1/确定使用RC2 块编码或RC4 流式编码 #ifdefUSE\_BLOCK\_CIPHER

#define ENCRYPT\_ALGORITHM CALG\_RC2 #define ENCRYPT\_BLOCK\_SIZE 8

#else

#define ENCRYPT\_ALGORITHM CALG\_RC4 #define ENCRYPT\_BLOCK\_SIZE 1

#endif

void CAPIDecryptFile(PCHAR szSource,PCHAR szDestination,PCHAR szPassword); void CAPIEncryptFile(PCHAR szSource,PCHAR szDestination,PCHAR szPassword);

int main(int argc,char \*argv[]) {

CHAR szSource[]="d:\plain.txt";

CHAR szDestination[]="d:\en.dat";

CHAR szDeDestination[]="D:\check.txt"; CHAR szPassword[]="123";

CAPIEncryptFile(szSource,szDestination,szPassword)

CAPIDecryptFile(szDestination,szDeDestination,szPassword); return 0;

}

/\*szSource为要加密的文件名称，szDestination为加密过的文件名称，szPassword 为加密口令\*/

void CAPIEncryptFile(PCHAR szSource,PCHAR szDestination,PCHAR szPassword) {

FILE \*hSource=NULL;

FILE \*hDestination=NULL; INT eof=0;

HCRYPTPROV hProv=0;

HCRYPTKEY hKey=0;

HCRYPTKEY hXchgKey=0; HCRYPTHASH hHash=0;

PBYTE pbKeyBlob=NULL;

**Windows** **C/C++加** **密** **解** **密** **实** **战**

DWORD dwKeyBlobLen PBYTE pbBuffer=NULL; DWORD dwBlockLen=16 DWORD dwBufferLen;

DWORD dwCount;

hSource=fopen(szSource,"rb"); //打开源文件

if(IhSource) {

printf("fopen %s failed",szSource);

return }

hDestination=fopen(szDestination,"wb"); //打开目标文件

//连接默认的CSP

CryptAcquireContext(&hProv,NULL,NULL,PROV\_RSA\_FULL,0); if(szPassword=NULL)

{

/口令为空，使用随机产生的会话密钥加密 //产生随机会话密钥

CryptGenKey(hProv,ENCRYPT\_ALGORITHM,CRYPT\_EXPORTABLE,&hKey); //取得密钥交换对的公共密钥

CryptGetUserKey(hProv,AT\_KEYEXCHANGE,&hXchgKey); //计算隐码长度并分配缓冲区

CryptExportKey(hKey,hXchgKey,SIMPLEBLOB,0,NULL,&dwKeyBlobLen); pbKeyBlob=(PBYTE)malloc(dwKeyBlobLen)

//将会话密钥输出至隐码

CryptExportKey(hKey,hXchgKey,SIMPLEBLOB,0,pbKeyBlob,&dwKeyBlobLen)

//释放密钥交换对的句柄

CryptDestroyKey(hXchgKey); hXchgKey=0;

//将隐码长度写入目标文件

fwrite(&dwKeyBlobLen,sizeof(DWORD),1,hDestination) //将隐码长度写入目标文件

fwrite(pbKeyBlob,1,dwKeyBlobLen,hDestination) }

else {

//口令不为空，使用从口令派生出的密钥加密文件 CryptCreateHash(hProv,CALG\_MD5,0,0,&hHash);

//建立散列表 //散列口令

CryptHashData(hHash,(BYTE\*)szPassword,strlen(szPassword),0); //从散列表中派生密钥

CryptDeriveKey(hProv,ENCRYPT\_ALGORITHM,hHash,0,&hKey); //删除散列表

CryptDestroyHash(hHash); hHash=0

}

//计算一次加密的数据字节数，必须为ENCRYPT\_BLOCK\_SIZE 的整数倍，dwBlockLen=1000-1000% ENCRYPT\_BLOCK\_SIZE

//如果使用块编码，就需要额外空间

if (ENCRYPT\_BLOCK\_SIZE>1)

dwBufferLen=dwBlockLen+ENCRYPT BLOCK SIZE;

**第** **9** **章CSP** **和CryptoAPI**

else

dwBufferLen=dwBlockLen;

//分配缓冲区

pbBuffer=(PBYTE)malloc(dwBufferLen); //加密源文件并写入目标文件

do {

//从源文件中读出dwBlockLen 字节

dwCount =fread(pbBuffer,1,dwBlockLen,hSource); eof=feof(hSource)

//加密数据

CryptEncrypt(hKey,0,eof,0,pbBuffer,&dwCount,dwBufferLen); //将加密过的数据写入目标文件

fwrite(pbBuffer,1,dwCount,hDestination); }while(!feof(hSource))

//关闭文件、释放内存 fclose(hSource);

fclose(hDestination); if(pbKeyBlob)

{

free(pbKeyBlob):

pbKeyBlob=NULL;

}

print(" 加密成功\n"); }

void CAPIDecryptFile(PCHAR szSource,PCHAR szDestination,PCHAR szPassword) {

FILE\*hSource =NULL;

FILE \*hDestination=NULL; INT eof=0;

HCRYPTPROV hProv=0;

HCRYPTKEY hKey=0;

HCRYPTKEY hXchgKey=0;

HCRYPTHASH hHash=0; PBYTE pbKeyBlob=NULL; DWORD dwKeyBlobLen;

PBYTE pbBuffer=NULL; DWORD dwBlockLen;

DWORD dwBufferLen; DWORD dwCount

hSource=fopen(szSource,"rb") //打开源文件

if (!hSource) {

printf("fopen %s failed",szSource); return

}

hDestination=fopen(szDestination,"wb"); //打开目标文件

//变量声明、文件操作同文件加密程序

CryptAcquireContext(&hProv,NULL,NULL,PROV\_RSA\_FULL,0); if(szPassword ==NULL)

**Windows** **C/C++加密解密实战**

{

//口令为空，使用存储在加密文件中的会话密钥解密 //读隐码的长度并分配内存

fread(&dwKeyBlobLen,sizeof(DWORD),1,hSource) pbKeyBlob=(PBYTE)malloc(dwKeyBlobLen);

/从源文件中读隐码

fread(pbKeyBlob,1,dwKeyBlobLen,hSource); //将隐码输入CSP

CryptImportKey(hProv,pbKeyBlob,dwKeyBlobLen,0,0,&hKey);

1 f

else {

//口令不为空，使用从口令派生出的密钥解密文件 CryptCreateHash(hProv,CALG\_MD5,0,0,&hHash);

CryptHashData(hHash,(BYTE\*)szPassword,strlen(szPassword),0);

CryptDeriveKey(hProv,ENCRYPT\_ALGORITHM,hHash,0,&hKey); CryptDestroyHash(hHash);

hHash=0; }

dwBlockLen=1000-1000%ENCRYPT\_BLOCK\_SIZE if(ENCRYPT\_BLOCK\_SIZE>1)

dwBufferLen=dwBlockLen+ENCRYPT\_BLOCK\_SIZE else

dwBufferLen=dwBlockLen;

pbBuffer=(PBYTE)malloc(dwBufferLen); //解密源文件并写入目标文件

do {

dwCount=fread(pbBuffer,1,dwBlockLen,hSource); eof=feof(hSource)

//解密数据

CryptDecrypt(hKey,0,eof,0,pbBuffer,&dwCount); //将解密过的数据写入目标文件

fwrite(pbBuffer,1,dwCount,hDestination); }while(!feof(hSource));

//关闭文件、释放内存 fclose(hSource)

fclose(hDestination) if(pbKeyBlob)

{

free(pbKeyBlob);

pbKeyBlob=NULL;



printf("解密成功\n");