HYPERSECU®

HYP2003 Developer Guide

10/19/2020

HSTE-NB0058-RV 1.2

HYPERSECU INFORMATION SYSTEMS INC

200-6191 Westminster Hwy, Richmond BC, V7C 4V4 Canada
1-604-279-2000 | hypersecu.com

Table of Contents

Document History

Version	Release Date	Description of Changes	Document Owner	Approved By
1.0	2017-11-15	Original document	NB	JL
1.1	2020-01-10	Revised document	NB	JL

1 Overview

This section describes how to develop HYP2003 applications, including the development interfaces supported by HYP2003 and how to develop applications based on these interfaces. This section covers the following topics:

1.1 HYP2003 Application Development Interfaces

HYP2003 application divide into two categories: development of PKI application and development of smart card application. In allusion to the interfaces of PKI application, HYP2003 provides two application interfaces PKCS#11 and CSP for Microsoft CryptoAPI 2.0, these two interfaces respectively followed PKCS#11 standard of RSA and MS CryptoAPI standard.

They can also be supported by other software/hardware manufacturers, so HYP2003 can be directly integrated into the application which accord with these two interfaces without customization. Another category is the PC/SC interface of smart card application.

The PKI application interface of HYP2003 is based on PC/SC interface. Developers can do customization using one or more interfaces according to the project requirement.

1.2 Application Development Using PC/SC Interfaces

The smart card subsystem on the Win32 platform is designed according to the PC/SC specifications. (For information on the specifications, visit http://www.pcscworkgroup.com). It includes:

- A Smart Card Resource Manager using Win32 system programming interfaces
- A User Interface working with the smart card resource manager
- A set of COM components providing smart card services

Figure 1 illustrates the architecture of the smart card subsystem under the Win32 platform:

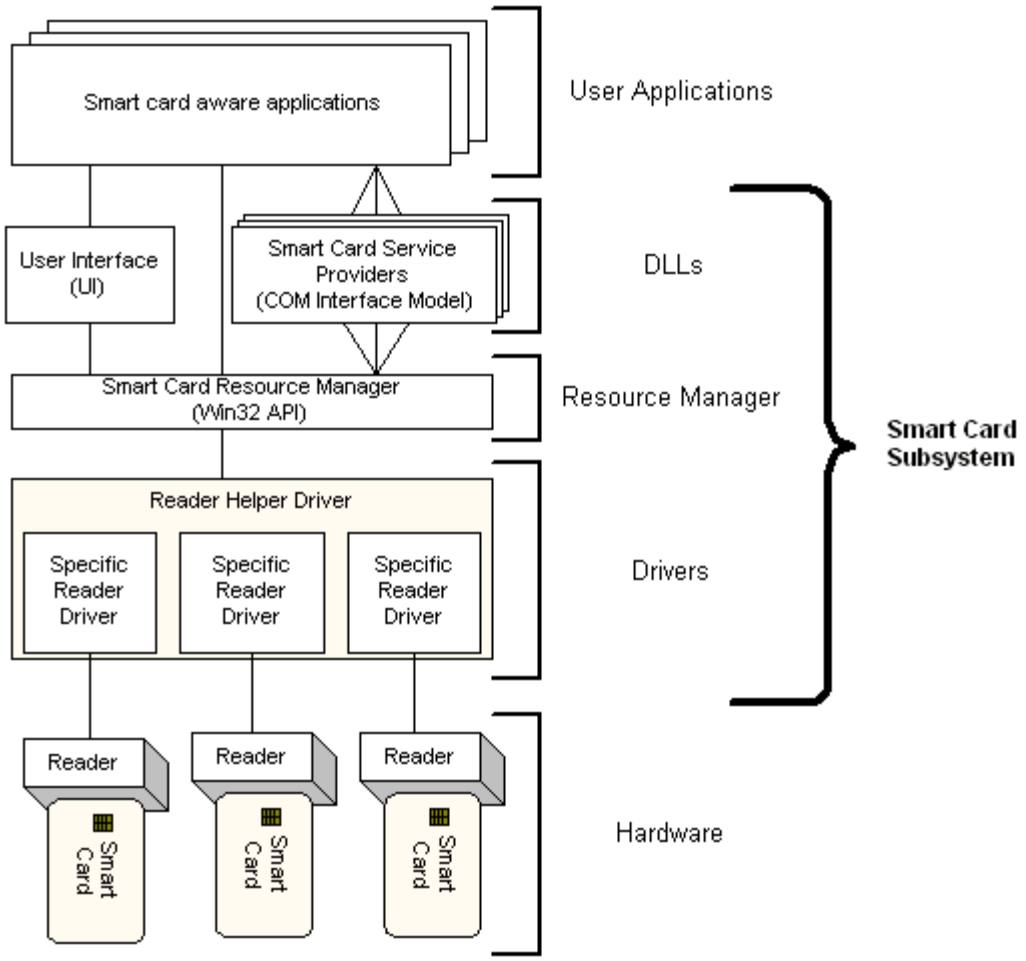

Figure 1 - Smart card subsystem architecture

As demonstrated above, the APIs provided by smart card manufacturers are separated from the interfaces used by smart card applications. In other words, the smart card applications use the smart card subset of standard Win32 functions only for accessing smart cards. For smart card manufacturers, the programming interfaces are unified. Changes or updates to the interfaces provided by smart card manufacturers does not affect the upper-level smart card applications.

The Smart Card Resource Manager acts as an intermediate layer. The Smart Card Resource Manager set of functions includes smart card database query functions, smart card database management functions, resource manager handle functions, resource manager tool functions, smart card monitoring functions, smart card and reader accessing functions, and direct card access functions.

1.2.1 Smart Card Database Query Functions

These functions can be used to search for the list of smart card types of a specific system user, the application service interfaces of a specific smart card, the grouping list of smart card readers, and the list of all smart card readers of a group.

When using these functions, the search scope can be the whole smart card resource database. You can also refine your search in the resource manager context by specifying some matching information. The function used for changing the smart card resource manager context is ScardEstablishContext. For some information, access may be denied for security reasons if you do not specify a specific context.

Function	Description
SCardGetProviderId	Obtain the identifier of the interface service of a specific smart card (GUID).
SCardListCards	Obtain the list of the smart card types accessible to a specific system user.
SCardListInterfaces	Obtain the unique identifier of the interface service component of a specific smart card (GUID).
SCardListReaderGroups	Obtain the list of the smart card groups.
SCardListReaders	Obtain the list of all smart card types of a specific smart card group.

1.2.2 Smart Card Database Management Functions

These functions can be used to manage the smart card resource database and update the location of the database with a specified resource context.

Function	Description
SCardAddReaderToGroup	Add a smart card reader to a specific smart card group.
SCardForgetCardType	Delete a smart card type.
ScardForgetReader	Delete a smart card reader.
ScardForgetReaderGroup	Delete a smart card reader group.
ScardIntroduceCardType	Add a new smart card type.
ScardIntroduceReader	Add a new reader type.
SCardIntroduceReaderGroup	Add a new reader group.

SCardRemoveReaderFromGroup	Delete a reader type from a specified reader group.

1.2.3 Resource Manager Handle Functions

These functions can be used to create or release the smart card operation context handles used by the smart card resource manager query or management functions.

Function	Description
ScardEstablishContext	Create a context handle for accessing the smart card database.
ScardReleaseContext	Close the context handle for accessing the smart card database.

1.2.4 Resource Manager Tool Function

This function can be used to release the memory area allocated automatically by the system function when flag SCARD_AUTOALLOCATE is specified.

Function	Description
ScardFreeMemory	Release the memory area allocated by the system function when flag SCARD_AUTOALLOCATE is specified.

1.2.5 Smart Card Monitoring Functions

These functions allow applications to track the current status of the smart card and reader. Most of them use structure array SCARD_READERSTATE to identify the status of the hardware.

Function	Description
SCardLocateCards	Look up a smart card matching a specified ATR string.
SCardGetStatusChange	Block execution until the current availability of the cards in a specific set of readers changes.
SCardCancel	Terminate all outstanding actions within a specific resource manager context.

1.2.6 Smart Card and Reader Accessing Functions

These functions can be used to connect to and access a specified smart card device, by performing I/O operations on the smart card using a data block containing control information which starts with structure SCARD_IO_REQUEST.

Function	Description
ScardConnect	Connect to a smart card.
ScardReconnect	Re-establish a connection to a smart card.
ScardDisconnect	Terminate a connection to a smart card.
ScardBegingTransaction	Start exclusive access to a smart card device and suspend access to the smart card device by other applications.
ScardStatus	Provide the current status of a reader.
ScardTransmit	Transmit data with a smart card via T=0 or T=1 protocol.

1.2.7 Direct Card Accessing Functions

The smart card subsystem under the Win32 platform allows applications to access the smart card devices which do not fully comply with the ISO7816 standards. Thus, Win32 smart card functions allow applications to send control commands and data to a reader directly. To use these functions, you must define an identifier for each of the properties you want to control. The Win32 smart card subset also defines some existing property marks.

Function	Description
ScardControl	Provide direct access control over a reader.
ScardGetAttrib	Obtain the properties of a reader.
ScardSetAttrib	Set the properties of a reader.

For platforms such as Windows 2000 and above, the components of the smart card subsystem are configured automatically when the operating system is installed.

For more information on the Win32 smart card set of functions, please refer to MSDN documents.

1.3 Application Development Using MS CryptoAPI Interfaces

Microsoft CryptoAPI is provided under the Win32 platform for developers to design data encryption and security applications. The CryptoAPI set of functions involve basic ASN.1 encoding/decoding, hash, data encryption/decryption, digital certificate management, etc. The data encryption/decryption can be achieved by symmetrical or public key algorithms. All Microsoft Win32 applications, such as Internet Explorer and Outlook, and many other third-party applications are based on the CryptoAPI interfaces for data encipherment.

There are three key requirements for secure data transmission over an insecure network: information concealment, identity authentication and integrity check. In addition to satisfying these requirements, the CryptoAPI interfaces provide standard ASN.1 encoding/decoding, data encryption/decryption, digital certificate and certificate storage management, Certificate Trust List (CTL), and Certificate Revocation List (CRL) functions.

1.3.1 Information Concealment

The aim of information concealment is to make sure that the content of transferred information can be retrieved by authorized people only. Normally, information concealment is achieved by applying some cryptographic methods. Data encryption algorithms can ensure secure information concealment and transmission with algorithms converting plain-text data to a set of hash data. It is almost impossible to deduce plain text from cipher text forcibly without the encryption key for "good" encryption algorithms. The original data could be ASCII text files, database files or any other kind of files which need to be transmitted securely. The term "information" means a set of data. The term "plain text" means unencrypted data. The term "cipher text" means encrypted data.

The cipher text could be transferred through insecure media or networks without compromising its security. After that, it is restored to plain text. This process is demonstrated in Figure 2:

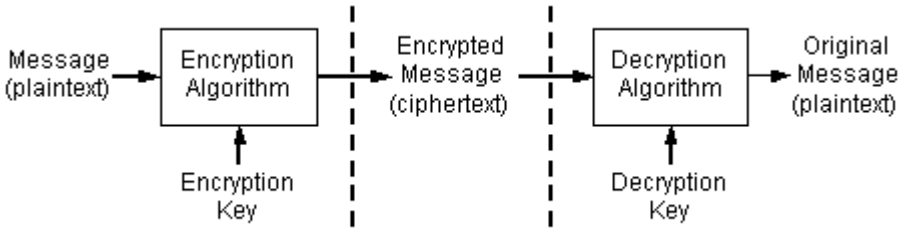

Figure 2 - Cipher text process

The concepts of data encryption and decryption are simple. To encrypt data, an encryption key is required. When performing a decryption process, a decryption key is required as well. The encryption key and the decryption key could be identical or completely different.

The encryption key must be stored safely and securely. When provided to other users, the transfer process for the key should be secure and reliable. Access control to the decryption key is also necessary, as it can be used to decrypt all data encrypted with the paired encryption key.

1.3.2 Identity Authentication

The prerequisite of secure communication is that both parties on the two sides of the communication know the identity of its opposite. Identity authentication is used to verify the true identity of a person or entity involved in an information exchange. The document used to identify the person or entity is called a credential. An example of a credential is the passport used to determine the true identity of the holder by custom officials. The credential is a physical document here.

Identity authentication could also be used to determine if the received data is exactly the sent data or not. For example, part B may want to verify if the received data is from part A indeed, instead of a pretender. In this context, the digital signature and verification functions of CryptoAPI can be used.

Because there is no physical link between the data transferred over the network and the user, the credential used to authenticate the data should also be transferable on the network. The credentials must be issued by trusted authorities.

Digital certificates, also referred to as the certificate, are a kind of such a credential. This credential used to authenticate on the network.

The digital certificate is a credential issued by a trusted organization or entity called a Certificate Authority (CA). It contains an appropriate public key, the certificate subject, and user information. The CA issues a certificate only when it has verified the accuracy of user information and a public key’s validity.

The information exchanged between the certificate applicant and the CA can use physical media, such as floppy disks, for transmission. Typically, this kind of information exchange is achieved through the network. A CA uses a trusted service program to handle an applicant’s requests and certificate issues.

1.3.3 Integrity Check

All the information transferred by unsafe media faces the risk of being tampered. In the real world, a seal can be used as a tool for an integrity check. For example, an unbroken seal on a package testifies that the item inside has been untouched after its departure from the manufacturer.

For the same reason, the information receiver not only needs to verify that the information is from the correct sender, but also needs to check the information has not been tampered with (changed). To build the integrity check mechanism, both the information and the verification information for it (usually called a hash value) must be sent together. The information and its verification information could be sent together with the digital certificate to prove information integrity.

1.3.4 CSP and Encryption Process

CryptoAPI functions use "Cryptographic Service Providers" (CSPs) to perform the data encryption/decryption and encryption key storage management. All the CSPs are independent modules. Theoretically, CSPs should be independent of specific applications, say; each of the applications could use any CSP. However, sometimes some applications can only interact with some specific CSPs. The relationship between CSPs and applications is similar to the Windows GDI model. CSPs work like graphic hardware drivers.

The storage security of the encryption key is laid on the CSP's implementation. It is not laid on the operating system. This makes it so that the application can be run under different security environments without modification.

The communication between application program and encryption module must be strictly controlled so that the application's security and migration can be guaranteed. Here are three applicable rules:

- Applications must not access the contents of the encryption key directly because all the encryption keys are generated within the CSP and applications use a transparent handler to handle it. This avoids any circumstances where the encryption key is leaked by the application or the related dynamic linking library and the encryption key is derived from a bad random factor.
- Applications must not specify the detailed implementation of the encryption operation. CSP API allows the application to choose the algorithm for performing encryption operations and signature operations. The actual implementation should be performed within the CSP.
- Applications must not process the data in the verification voucher or other identity authentication data. A user's identity authentication should be achieved by the CSP. This ensures the application needs to be modified in the future when more identity authentication approaches are applied such as fingerprint scanning.

The simplest CSP is comprised of a Win32 Dynamic Linking Library (DLL) and a signature file. Only by providing the correct signature file can the CSP be recognized and used by CryptoAPI. CryptoAPI will check the signatures of CSPs periodically to prevent them being tampered with.

Some CSP modules perform sensitive encryption operations at separate memory spaces by calling local RPC or hardware driver programs. Placing encryption keys and performing sensitive encryption operations in separate memory space or hardware can ensure the keys are not tampered with by the applications.

It is not recommended to have an application rely on only one specific CSP. For example, Microsoft Base Cryptographic Provider provides a 40-bit communication key and 512-bit public key. Applications should avoid only using these sizes as the length of communication and public key, because once an application uses another CSP, the key length might change. Good applications should interact with different CSPs.

1.3.5 CSP Context

The first CryptoAPI function called by an application must be CryptAcquireContext. This function returns a CSP operation handle specifying a certain key container. The key

container can be selected specially. Or the default container for current user can be used. The function can also be used to create a new key container.

The CSP module itself has a name and a type. For example, Windows operating system's default installed CSP is Microsoft Base Cryptographic Provider. Its type is PROV_RSA_FULL. Each CSP's name must be different, but their types can be same.

When an application calls the CryptoAcquireContext function to get a CSP operation handler, it can specify the name and type of CSP. When the CSP name and type are specified, only the matching CSP will be called. After a successful call, the function returns the CSP operating handler. The application can then use the handler to access the CSP and the key container in the CSP.

1.3.6 CryptoAPI Architecture

Cryptographic Functions

Used to link and create CSP handle. This set of functions allows applications to choose a specific CSP module by specifying its name and type.

Key Generation Functions

Used to create and store an encryption key. Features include change of encryption mode, initialization of encryption vector etc.

Key Exchange Functions

Used to exchange and transmit keys.

Certificate Encoding/Decoding Functions

Used to encrypt and decrypt data, including support for data hash operations.

Certificate Storage Functions

Used to manage digital certificate sets.

Simplified Message Functions

Used to encrypt and decrypt messages and data, sign them, and verify the validity of their signature.

Low Level Message Functions

Actual implementation of the simplified message processing functions, with more specific controls over message operations.

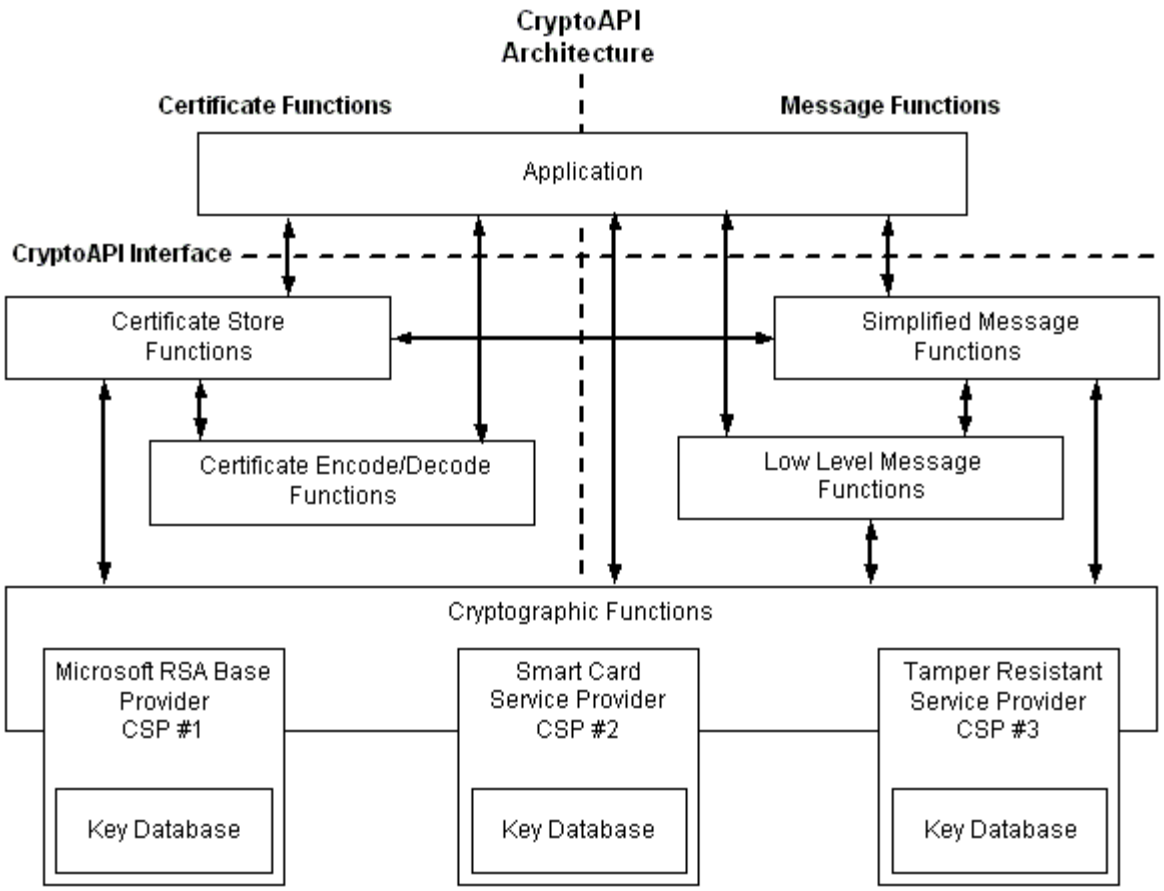

Figure 3 - Crypto API architecture

The prefix of a set of functions has a specific form, as follows:

Function Category	Prefix Convention
Cryptographic Functions	Crypt
Certificate Encoding/Decoding Functions	Crypt
Certificate Storage Functions	Store
Simplified Message Functions	Message
Low Level Message Functions	Msg

1.4 Application Development Using PKCS#11 Interfaces

Because of the blooming growth of Internet, the security requirement for applications has become increasingly important. The growth of security products also results in requirements for interacting with applications. RSA Security worked out the Public Key Cryptographic Standard (PKCS) to meet these requirements.

PKCS#11 standard is one of the PKCS standard set. The PKCS#11 standard (also known as "Cryptoki") is used to resolve the compatibility problems from interactions between different manufacturers and public key applications. It defines a uniform programming interface model: Cryptoki tokens. The PK Card PKCS#11 interfaces are compliant with the PKCS#11 standard version 2.20.

Before programming with the PK Card PKCS#11 interfaces, developers should be familiar with PKCS#11 standards. The standard's related documents can be downloaded from the RSA website at http://www.rsa.com/rsalabs/node.asp?id=2133

2 CSP Module

This section introduces the CryptoAPI development interfaces supported by HYP2003. In particular, the CSP interface names, supported functions, and algorithm implementation are described. This section covers the following topics:

2.1 CSP Module Description

HYP2003 provides a standard CSP module for seamless integration with CryptoAPI applications. The CSP module complies with Microsoft Crypto Service Provider programming standard. It can be compatible with current and future CryptoAPI applications.

2.1.1 Basic Information

- Type: PROV_RSA_FULL

This general type of CSP provides support for digital signature and data encryption and decryption. All public key operations are processed using RSA algorithms.

- Name: HyperPKI HYP2003 CSP V1.0

This CSP is also registered as a Smart Card type in Windows system, so that HYP2003 can support smart card related operations, such as Windows Smart Card Logon.

2.1.2 Features

The CSP module of HYP2003 is designed to have the followings features:

- Secure RSA key-pair storage container
- Different block encryption and hash algorithms
- Support for RSA operations done by the hardware (up to 2048 bits)

- Support for random number generation by the hardware
- Support for multi-thread access and multi-device management
- Support for multi-certificate applications
- Compliant with PKCS#11 data format
- Support for dual credentials by allowing two key-pairs (AT_KEYEXCHANGE and AT_SIGNATURE) and corresponding certificates in a single container
- Support for Windows XP/Vista/7/8/8.1/10 and Windows Server
- Seamlessly compatible with existing Windows platform applications, such as Microsoft Office encryption and decryption, Internet Explorer webpage and SSL website logon, secure emails on Microsoft Outlook (Express), etc.

2.2 Supported Algorithms

The following is a list of all cryptographic algorithms supported by the product's CSP module:

Algorithm	Default Length (in bits)	Min. Length (in bits)	Max. Length (in bits)	Purpose
CALG_RC2	40	8	1024	Encryption and decryption
CALG_RC4	40	8	2048	
CALG_DES	56	56	56	
CALG_3DES	192	192	192	
CALG_SHA1	160	160	160	Hash operation
CALG_MD2	128	128	128	
CALG_MD5	128	128	128	
CALG_SSL3_SHAMD5	288	288	288	
CALG_RSA_SIGN or AT_SIGNATURE	1024	1024	2048	Signature verification
CALG_RSA_KEYX or AT_KEYEXCHANGE	1024	1024	2048	Encryption, decryption, and signature verification

2.3 Function Implementation

The following table summarizes the support and implementation of CSP interface functions. "Not Implemented" indicates that the interface exists in the CSP module, but it is not implemented. "Not Supported" indicates that the interface does not exist at all in the CSP module.

It is reasonable that some functions listed in the table are not supported because the CSP type is PROV_RSA_FULL. The "Not Implemented" functions return FALSE and the ErrorCode is set to E_NOTIMPL. CryptoAPI applications are not required to call these interface functions directly.

Name	Description	Availability
Connection Functions		
CPAcquireContext	Create a context for an application.	Implemented
CPGetProvParam	Return CSP related information.	Implemented
CPReleaseContext	Release the context created by CPAcquireContext.	Implemented
CPSetProvParam	Set CSP parameter operations.	Implemented
Key Generation and Exchange Functions		
CPDeriveKey	Generate a session key from a data hash. The key is unique.	Implemented
CPDestroyKey	Release a key handle. The handle will be invalid then, and the key cannot be accessed.	Implemented
CPDuplicateKey	Create a copy of a key.	Not Supported
CPExportKey	Export a key from a CSP container.	Implemented
CPGenKey	Generate a key or key pair.	Implemented
CPGenRandom	Write a random number to a buffer.	Implemented

CPGetKeyParam	Get the attributes of an encryption key.	Implemented
CPGetUserKey	Get the persisted key pairs from a CSP container.	Implemented
CPImportKey	Import a key from a blob to a CSP container.	Implemented
CPSetKeyParam	Set key attributes.	Implemented
Data Encryption Functions		
CPDecrypt	Decrypt the encrypted data.	Implemented
CPEncrypt	Encrypt the plain text.	Implemented
Hashing and Digitally Signing Functions		
CPCreateHash	Initialize and hash input data.	Implemented
CPDestroyHash	Delete the handle of a hashed object.	Implemented
CPDuplicateHash	Create a copy of a hashed object.	Not Supported
CPGetHashParam	Get the calculation result of a hashed object.	Implemented
CPHashData	Hash input data.	Implemented
CPHashSessionKey	Hash a session key and do not expose its value to the application.	Not Implemented
CPSetHashParam	Customize the attributes of a hashed object.	Implemented
CPSignHash	Sign a hashed object.	Implemented
CPVerifySignature	Verify a digital signature.	Implemented

In addition, although the function OffloadModExpo is defined in the CSP standard, it is not supported by the CSP module for the moment.

2.4 Function Parameters

i. CPAcquireContext

—dwFlags

Supports the following values: CRYPT_VERIFYCONTEXT, CRYPT_NEWKEYSET, CRYPT_DELETEKEYSET and CRYPT_SILENT; No CRYPT_MACHINE_KEYSE processing cases.

——pszContainer

May be NULL or "", or a string with a reader name (the length of the string should not exceed MAX_PATH) depending on the value of *dwFlags*.

ii. CPGetProvParam

——dwParam

Supports the following values: 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; and does not support the following values: PP_KEYSET_SEC_DESCR, PP_USE_HARDWARE_RNG etc.

—dwFlags

According to CSP analysis, when the value of *dwParam* is PP_ENUMALGS or PP_ENUMALGS_EX, enumeration begins if *dwFlags* is CRYPT_FIRST; or if the value is 0 or CRYPT_NEXT, the next is enumerated. When the value of *dwParam* is PP_ENUMCONTAINERS, enumeration begins if *dwFlags* is CRYPT_FIRST (1) or CRYPT_FIRST|CRYPT_NEXT (3); or the next is enumerated if its value is 0 or CRYPT_NEXT. *dwFlags* does not support CRYPT_MACHINE_KEYSET. When *dwParam* is set to other values, the value of *dwFlags* is not checked.

iii. CPReleaseContext

——dwFlags

Value must be zero.

iv. CPSetProvParam

——dwParam

Supports the following values: PP_KEYEXCHANGE_PIN and PP_SIGNATURE_PIN. Logout if *pbData* is NULL.

Does not support other values.

——dwFlags

Not checked.

v. CPDeriveKey

——Algid

Supports the following algorithms only: CALG_RC2, CALG_RC4, CALG_DES, and CALG_3DES.

——dwFlags

Returns an error for the following values: (CRYPT_CREATE_SALT | CRYPT_NO_SALT), CRYPT_PREGEN and CRYPT_USER_PROTECTED. Not supported for other values.

vi. CPDestroyKey

Further description not required.

vii. CPDuplicateKey

Not supported.

viii. CPExportKey

——dwBlobType

Supports only PUBLICKEYBLOB and SIMPLEBLOB, and does not support PRIVATEKEYBLOB, OPAQUEKEYBLOB, and PLAINTEXTKEYBLOB etc.

——dwFlags

If *dwBlobType* is PUBLICKEYBLOB or SIMPLEBLOB, dwFlags must be zero. The value of this parameter is ignored for other cases.

ix. CPGenKey

——Algid

Supports the following values: CALG_RSA_KEYX, CALG_RSA_SIGN, AT_KEYEXCHANGE, AT_SIGNATURE, CALG_DES, CALG_RC2, CALG_RC4 and CALG_3DES. CALG_3DES_112 is supported for the next version.

——dwFlags

Not supported CSP returns an error message: CRYPT_CREATE_SALT, CRYPT_NO_SALT, or CRYPT_PREGEN. The length of the key to be generated is the first two bytes of this parameter (the key with default length will be generated for 0). The last two bytes are ignored.

x. CPGenRandom

Further description not required.

xi. CPGetKeyParam

Supports only CALG_RSA_KEYX, CALG_RSA_SIGN, AT_KEYEXCHANGE, AT_SIGNATURE, CALG_DES, CALG_RC2, CALG_RC4 and CALG_3DES key types.

——dwParam

For the key types like CALG_RSA_KEYX, CALG_RSA_SIGN, AT_KEYEXCHANGE and AT_SIGNATURE, its value could be KP_PERMISSIONS, KP_CERTIFICATE, KP_BLOCKLEN, KP_KEYLEN or KP_ALGID; for the key type like CALG_RC2, its value could be KP_BLOCKLEN, KP_EFFECTIVE_KEYLEN, KP_KEYLEN, KP_ALGID or KP_SALT; for the key type like CALG_RC4, its value could be KP_BLOCKLEN (return value 0), KP_KEYLEN, KP_ALGID or KP_SALT; for the key types like CALG_3DES and CALG_DES, its value could be KP_BLOCKLEN, KP_KEYLEN or KP_ALGID.

——dwFlags

Must be zero.

xii. CPGetUserKey

—dwParam

Supports the following values: AT_KEYEXCHANGE, AT_SIGNATURE, and (AT_KEYEXCHANGE | AT_SIGNATURE).

xiii. CPImportKey

——pbData

This keyBlob supports SIMPLEBLOB, PUBLICKEYBLOB and PRIVATEKEYBLOB.

——dwFlags

Ignored.

xiv. CPSetKeyParam

——dwParam

For the key types like CALG_RC2, CALG_DES and CALG_3DES, its value is KP_IV; for the key type like CALG_RC2, its value is KP_EFFECTIVE_KEYLEN; for the key types like CALG_RC2 and CALG_RC4, its value is KP_SALT or KP_SALT_EX; for the key types like CALG_RSA_KEYX, CALG_RSA_SIGN, AT_KEYEXCHANGE and AT_SIGNATURE, its value is KP_CERTIFICATE.

——dwFlags

Must be zero.

xv. CPDecrypt

Supports the following key types: CALG_RSA_KEYX, AT_KEYEXCHANGE, CALG_RC2, CALG_DES, CALG_3DES and CALG_RC4.

——*dwFlags*

Must be zero.

xvi. CPEncrypt

Supports the following key types: CALG_RSA_KEYX, AT_KEYEXCHANGE, CALG_RC2, CALG_DES, CALG_3DES and CALG_RC4.

——*dwFlags*

Must be zero.

xvii. CPCreateHash

——*Algid*

Supports the following algorithms: CALG_MD2, CALG_MD5, CALG_SHA1 and CALG_SSL3_SHAMD5.

——*dwFlags*

Must be zero.

xviii. CPDestroyHash

Further description not required.

xix. CPDuplicateHash

Not supported.

xx. CPGetHashParam

——*dwParam*

Supports the following values: HP_ALGID, HP_HASHSIZE and HP_HASHVAL.

——*dwFlags*

Must be zero.

xxi. CPHashData

——*dwFlags*

Must be zero. It does not support the value of CRYPT_USERDATA.

xxii. CPHashSessionKey

Not implemented. It returns FALSE and sets ErrorCode to E_NOTIMPL.

xxiii. CPSetHashParam

——*dwParam*

Supports only the value of HP_HASHVAL.

——*dwFlags*

Must be zero.

xxiv. CPSignHash

——*sDescription*

Ignored.

——*dwFlags*

Supports only the value of CRYPT_NOHASHOID. Other values are ignored.

xxv. CPVerifySignature

——*sDescription*

Ignored.

——*dwFlags*

Does not support any value.

2.5 Function Calling Description

xxvi. General

The function first called among all CSP functions is CPAcquireContext. Upper applications call this function to determine which key container they operate on. Each key container can only store one RSA key pair of the same type and many session keys at one time. The RSA key pair is an object that could be persisted, while the session keys exist only at runtime. If an application requests the access to the private key in the container, the CSP module would require authentication to the user. But if this dialog box is not expected, set a flag, CRYPT_SILENT. However, doing so will cause that all operations with access to the private key and protected data fail, because the product does not support the use of CPSetProvParam for setting user identification.

xxvii. Development Samples

Developers could find some sample programs developed with the CryptoAPI interfaces and compile and debug them in SDK package under `Samples\CryptAPI`. Some samples may require Platform SDK from Microsoft.

3 PKCS#11 Module

This section introduces the PKCS#11 interface development. In particular, the PKCS#11 interface names, supported functions and algorithm implementation are described. This section covers the following topics:

3.1 PKCS#11 Module Description

The PKCS#11 interfaces are provided in a Win32 dynamic linking library (DLL), which can be accessed through a static link (using .lib file) or dynamic link. The following are the files related to the PKCS#11 interfaces:

File	SDK Path
pkcs11.h	\Include\pkcs11 (provided by RSA)
pkcs11f.h	\Include\pkcs11 (provided by RSA)
pkcs11t.h	\Include\pkcs11 (provided by RSA)
cryptoki.h	\Include\pkcs11 (provided by RSA)
cryptoki_ext.h	\Include\pkcs11 (extension algorithms and return values)
cryptoki_win32.h	\Include\pkcs11 (type definition of the first 3 header files required for Windows platforms)
cryptoki_linux.h	\Include\pkcs11 (type definition of the first 3 header files required for Linux platforms)
auxiliary.h	\Include\pkcs11 (definition of extension functions)

HyperPKICsp11_2003.lib	\Lib (PKCS#11 interface library)

`HyperPKICsp11_2003.dll` is the core library file for the PK card. It is located under the system directory. The library implements all interface functions defined in RSA PKCS#11 standard. If developers need to use these interfaces and all interfaces and definitions developers wish to access are compliant with the PKCS#11 standard, the file `cryptoki_win32.h` (for Windows platforms) or `cryptoki_linux.h` (for Linux platforms) must be included in the project. If the extension functions and algorithms are to be used, simply get the file `cryptoki_ext` involved. You can include the library in your project and call it implicitly if you do not want to call the library by LoadLibrary.

3.2 Supported PKCS#11 Objects

HYP2003 PKCS#11 module supports creating and using the following objects:

Class Object	Description
CKO_DATA	Object defined by application. Object's structure is decided by the application. The data rendering is also handled by the application.
CKO_SECRET_KEY	Key of symmetry encryption algorithm.
CKO_CERTIFICATE	X.509 digital certificate object.
CKO_PUBLIC_KEY	RSA public key object.
CKO_PRIVATE_KEY	RSA private key object.
CKO_MECHANISM	Algorithm object.

All objects can be divided into groups according to the length of their lifetime. One group is a permanently stored object. This group of objects will be stored in a secure memory area until being deleted by the application. Another group includes session objects. This group of objects is only used for temporary communication sessions. Once the session is finished, the object will be deleted. The property CKA_TOKEN decides the lifetime of the object, which has a Boolean value. All the objects have this property. Developers need to establish a storage policy for objects according to the memory size of the product. Only the significant objects can be stored within the internal memory of the product.

Besides lifetime difference, the PKCS#11 objects also have a difference in access privileges. All the objects can be divided into two types according to their different access privilege. One type is *public object*, with this type of object being accessed by any user. The other type is *private object*, which can only be accessed by users who have passed identity verification. The property CKA_PRIVATE decides the access type, which has a Boolean value. All objects have this property. The application can decide whether one object is public or private by its actual usage.

3.3 Supported Algorithms

The following is a list of all cryptographic algorithms supported by the PKCS#11 module of the product:

Algorithm	Encryption Decryption	Signature Check	Hash	Key-pair Generation	Wrap
CKM_RSA_PKCS_KEY_PAIR_GEN				√	
CKM_RSA_PKCS	√	√			√
CKM_MD2_RSA_PKCS		√			
CKM_MD5_RSA_PKCS		√			
CKM_SHA1_RSA_PKCS		√			
CKM_SHA224_RSA_PKCS		√			
CKM_SHA256_RSA_PKCS		√			
CKM_SHA384_RSA_PKCS		√			
CKM_SHA512_RSA_PKCS		√			
CKM_RSA_X9_31_KEY_PAIR_GEN				√	
CKM_RSA_X9_31		√			
CKM_SHA1_RSA_X9_31		√			
CKM_RSA_PKCS_OAEP	√				
CKM_RSA_PKCS_PSS		√			
CKM_SHA1_RSA_PKCS_PSS		√			
CKM_SHA256_RSA_PKCS_PSS		√			
CKM_SHA224_RSA_PKCS_PSS		√			
CKM_SHA384_RSA_PKCS_PSS		√			
CKM_SHA512_RSA_PKCS_PSS		√			
CKM_RSA_X_509	√	√			√
CKM_EC_KEY_PAIR_GEN				√	
CKM_ECDSA		√			

Algorithm	Encryption Decryption	Signature Check	Hash	Key-pair Generation	Wrap
CKM_ECDSA_SHA1		√			
CKM_DH_PKCS_KEY_PAIR_GEN				√	
CKM_RC2_KEY_GEN				√	
CKM_RC2_ECB	√				
CKM_RC2_CBC	√				
CKM_RC2_CBC_PAD	√				
CKM_RC4_KEY_GEN				√	
CKM_RC4	√				
CKM_DES_KEY_GEN				√	
CKM_DES_ECB	√				
CKM_DES_CBC	√				
CKM_DES_CBC_PAD	√				
CKM_DES3_KEY_GEN				√	
CKM_DES3_ECB	√				
CKM_DES3_CBC	√				
CKM_DES3_CBC_PAD	√				
CKM_AES_KEY_GEN				√	
CKM_AES_ECB	√				
CKM_AES_CBC	√				
CKM_AES_CBC_PAD	√				
CKM_MD2			√		
CKM_MD5			√		
CKM_SHA_1			√		
CKM_SHA224			√		

Algorithm	Encryption Decryption	Signature Check	Hash	Key-pair Generation	Wrap
CKM_SHA256			√		
CKM_SHA384			√		
CKM_SHA512			√		

The following list provides the key length supported by the PKCS#11 module of the product:

Algorithm	Key Length
CKM_RSA_KEY_PAIR_GEN	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_GENERIC_SECRET_KEY_GEN	1-256bytes
CKM_AES_KEY_GEN	128,192,256bits

3.4 Supported PKCS#11 Interface Functions

PKCS#11 is a universal standard for the Cryptoki hardware. The implementation of PKCS#11 from different hardware manufacturers may vary.

Some of the interfaces defined in the PKCS#11 standards are not implemented by the product. Once they are called, a value CKR_FUNCTION_NOT_SUPPORT will be returned.

NOTE: The product is the "token" mentioned in the PKCS#11 standards.

The following is a list of all interfaces defined in the PKCS#11 2.11 standards:

Name	Description	Availability
Basic Functions		
C_Initialize	This function initializes the library. It must be called before calling other	Implemented

Name	Description	Availability
	functions with the only exception being the C_GetFunctionList function.	
C_Finalize	This function should be called at the end of access.	Implemented
C_GetInfo	Get the information of Cryptoki library.	Implemented
C_GetFunctionList	Get the function pointer list of the library.	Implemented
Slot and Token Management Functions		
C_GetSlotList	Get slot list.	Implemented
C_GetSlotInfo	Get slot information.	Implemented
C_GetTokenInfo	Get token information in the slot.	Implemented
C_WaitForSlotEvent	Wait for slot event, such as token is inserted or removed.	Implemented
C_GetMechanismList	Get the library's supported algorithm list.	Implemented
C_GetMechanismInfo	Get the detail information of the algorithm.	Implemented
C_InitToken	Initialize a token.	Implemented
C_InitPIN	Initialize USER PIN.	Implemented
C_SetPIN	Set current user PIN.	Implemented
Session Management Functions		
C_OpenSession	Open a session between application and token.	Implemented
C_CloseSession	Close session.	Implemented
C_CloseAllSessions	Close all the opened session.	Implemented
C_GetSessionInfo	Get session information.	Implemented

Name	Description	Availability
C_GetOperationState	Get current operation state.	Not Implemented
C_SetOperationState	Use state returned by C_GetOperationState to resume the library's operating state.	Not Implemented
C_Login	Log into the token.	Implemented
C_Logout	Log out from the token.	Implemented
Object Management Functions		
C_CreateObject	Create new Cryptoki object.	Implemented
C_CopyObject	Create the copy of the object.	Not Implemented
C_DestroyObject	Destroy the object.	Implemented
C_GetObjectSize	Get the size of the object.	Not Implemented
C_GetAttributeValue	Get the attributes of the object.	Implemented
C_SetAttributeValue	Set the attributes of the object.	Implemented
C_FindObjectsInit	Initialize an object finding operation.	Implemented
C_FindObjects	Perform an object finding operation.	Implemented
C_FindObjectsFinal	Finish an object finding operation.	Implemented
Encryption Functions		
C_EncryptInit	Initialize an encryption operation.	Implemented
C_Encrypt	Encrypt the data.	Implemented
C_EncryptUpdate	Continue encrypting data.	Implemented

Name	Description	Availability
C_EncryptFinal	End a data encryption operation.	Implemented
Decryption Functions		
C_DecryptInit	Initialize a decryption operation.	Implemented
C_Decrypt	Decrypt the data.	Implemented
C_DecryptUpdate	Continue decrypting data.	Implemented
C_DecryptFinal	End a data decryption operation.	Implemented
Digest Functions		
C_DigestInit	Initialize a digest operation.	Implemented
C_Digest	Input data for digesting.	Implemented
C_DigestUpdate	Continue digesting data.	Implemented
C_DigestKey	Continue digesting key.	Not Implemented
C_DigestFinal	End a data digest operation.	Implemented
Signature Functions		
C_SignInit	Initialize a signature operation.	Implemented
C_Sign	Signature operation.	Implemented
C_SignUpdate	Update signature operation.	Implemented
C_SignFinal	Finalize signature operation.	Implemented
C_SignRecoverInit	Initialize a data recoverable signature operation.	Implemented
C_SignRecover	Recover signature operation.	Implemented

Name	Description	Availability
Signature Verification Functions		
C_VerifyInit	Initialize a signature verification operation.	Implemented
C_Verify	Verification operation.	Implemented
C_VerifyUpdate	Update verification operation.	Implemented
C_VerifyFinal	Finalize verification operation.	Implemented
C_VerifyRecoverInit	Initialize a data recoverable verification operation.	Implemented
C_VerifyRecover	Recover verification operation.	Implemented
Digest Encryption Functions		
C_DigestEncryptUpdate	Continue a digest and encryption operation.	Not Implemented
C_DecryptDigestUpdate	Continue a digest and decryption operation.	Not Implemented
C_SignEncryptUpdate	Continue a signature and encryption operation.	Not Implemented
C_DecryptVerifyUpdate	Continue a signature and decryption operation.	Not Implemented
Key Management Functions		
C_GenerateKey	Generate the key and create the new key object.	Implemented
C_GenerateKeyPair	Generate the key pair and create the new public key object.	Implemented
C_DeriveKey	Derive a private key or encryption key.	Not Implemented
C_WrapKey	Wrap a private key or encryption key.	Implemented

Name	Description	Availability
C_UnwrapKey	Un-wrap a private key or encryption key.	Implemented
Random Number Generation Functions		
C_SeedRandom	Add a seed to the random generator.	Not Implemented
C_GenerateRandom	Generate a random number.	Implemented
Parallel Management Functions		
C_GetFunctionStatus	Already been deprecated.	Not Implemented
C_CancelFunction	Already been deprecated.	Not Implemented

4 Smart Card Mini Driver Module

This section introduces the CryptoAPI interface development. In particular, the Smart Card Mini Driver interface names, supported functions, and algorithm implementation are described. This section covers the following topics:

- HYP2003 Smart Card Mini Driver Module Description
- Supported Algorithms
- Function Implementation
- Function Parameters
- Function Calling Description

4.1 HYP2003 Smart Card Mini Driver Module Description

The Smart Card Mini Driver interface is the underlayer of Microsoft Base Smart Card Crypto Provider and Microsoft Smart Card Key Storage Provider. It provides encryption algorithm and file storage function.

HYP2003 provides a standard Smart Card Mini Driver Module to implement Microsoft Smart Card Base Cryptographic Service Provider (CSP) and Cryptography API: Next Generation (CNG) Key Storage Provider (KSP). The Smart Card Mini Driver Module for HYP2003 fully complies with Microsoft Windows Smart Card Mini Driver coding standard. It's compatible with existing and future Crypto API applications. The procedure flow of the Smart Card Mini Driver is shown in Figure 4.

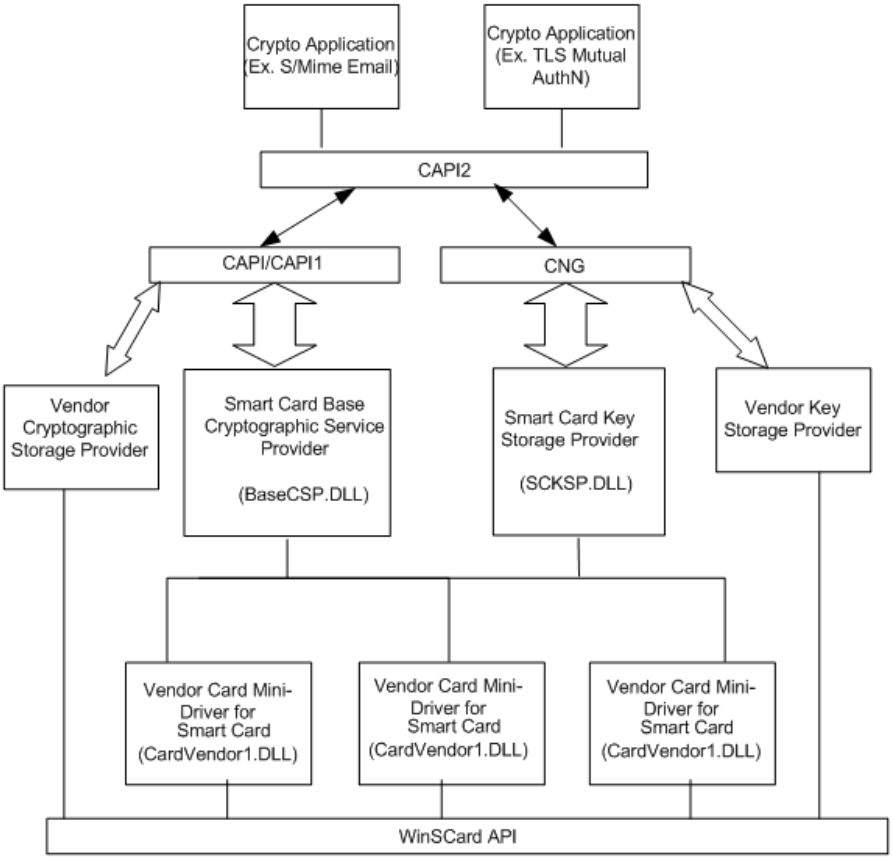

Figure 4 - Smart Card Mini Driver procedure flow

4.1.1 Basic Information

Supported Versions: V4, V5, V6, V7

V4 supports CSP basic function, one user PIN and one administrator PIN

V5 supports AT_ECDHE_* algorithm

V6 supports that one PIN binding one container

V7 supports secure key injection

Name: Microsoft Base Smart Card Crypto Provider / Microsoft Smart Card Key Storage Provider

Registration Mechanism:

	Windows XP and Windows 2003	Windows Vista/2008/7
CAPI-1	Registry file Must set "Crypto Provider" registry key.	Manifest Must set "Crypto Provider" registry key.
CNG	Not supported	Manifest Must set "Smart Card Key Storage Provider" key.

NOTE:

"Crypto Provider"="Microsoft Base Smart Card Crypto Provider"
"Smart Card Key Storage Provider"="Microsoft Smart Card Key Storage Provider"

4.1.2 Features

Smart Card Mini Driver for HYP2003 has following features:

- Provides secure container for RSA/ECC key pair
- Supports RSA2048 in hardware
- Supports ECC256 in hardware
- Supports creation and deletion of binary file
- Supports hardware random number generation
- Supports multi-thread access and multi-device management
- Supports multi-certificate application
- Compatible with PKCS#11 data format
- Supports dual certification – one container includes two key pairs (AT_KEYEXCHANGE and AT_SIGNATURE) and corresponding certificates
- Support Windows2000 and above (Windows2000/XP/server 2003 need to install the MS patch KB909520)
- Seamless compatibility with Windows applications such as Microsoft Office Encryption/Decryption, Internet Explorer web logon and SSL logon, Microsoft Outlook (Express) secure email, Smart Card Logon, MS VPN connection, and more.

4.2 Supported Algorithms

The following is a list of all cryptographic algorithms supported by the Smart Card Mini Driver module for the product:

Algorithm	Default Length (bit)	Min. Length (bit)	Max. Length (bit)	Purpose
AT_ECDSA_P256	256	256	256	Signature
AT_SIGNATURE	1024	1024	2048	Signature verification
AT_KEYEXCHANGE	1024	1024	2048	Encryption, decryption, and signature verification

4.3 Function Implementation

The following table summarizes the support and implementation of Smart Card Mini Driver interface functions. "Not Implemented" indicates that the interface exists in the CSP module, but it is not implemented. "Not Supported" indicates that the interface does not exist at all in the CSP module.

All the functions with "not implemented" return SCARD_E_UNSUPPORTED_FEATURE.

Name	Description	Availability
Connection Functions		
CardAcquireContext	Create a context for the application	Implemented
CardDeleteContext	Release the context created by CardDeleteContext	Implemented
CardGetProperty	Get the basic property of Smart Card Mini Driver	Implemented
CardSetProperty	Set the basic property of Smart Card Mini Driver	Implemented
PIN Management Function		
CardGetChallenge	Get random number	Implemented
CardAuthenticatePin	Verify user PIN	Implemented
CardAuthenticateChallenge	Verify Administrator PIN externally	Implemented
CardDeauthenticate	Invalidate the PIN permissions of Administrator or user	Implemented

Name	Description	Availability
CardUnblockPin	Unlock user PIN.	Implemented
CardChangeAuthenticator	Change PIN	Implemented
CardAuthenticateEx	Verify PIN	Implemented
CardChangeAuthenticatorEx	Change PIN	Implemented
CardDeauthenticateEx	Invalidate appointed PIN permissions	Implemented
CardGetChallengeEx	Get random number	Implemented
File System Management Function		
CardCreateDirectory	Create a directory	Implemented
CardDeleteDirectory	Delete a directory	Implemented
CardReadFile	Read file content	Implemented
CardCreateFile	Create a file	Implemented
CardGetFileInfo	Get file information	Implemented
CardWriteFile	Write file content	Implemented
CardDeleteFile	Delete a file	Implemented
CardEnumFiles	Enumerate file name	Implemented
CardQueryFreeSpace	Get key space	Implemented
Container Management Function		
CardCreateContainer	Create a container	Implemented
CardCreateContainerEx	Create a container which has bound the PIN	Implemented
CardDeleteContainer	Delete a container	Implemented
CardGetContainerInfo	Get container information	Implemented
CardGetContainerProperty	Get container property	Implemented
CardSetContainerProperty	Set container property	Not Implemented
CardQueryKeySizes	Get supported ECC/RSA length	Implemented

Name	Description	Availability
CardQueryCapabilities	Estimate whether Mini Driver support creating key pair	Implemented
Asymmetric Key Operation		
CardRSADecrypt	RSA decryption	Implemented
CardSignData	RSA/ECC signature	Implemented
ECDH Algorithm Function		
CardConstructDHAgreement	Create diffie hellman protocol	Implemented
CardDestroyDHAgreement	Release diffie hellman protocol	Implemented
CardDeriveKey	Generate session key based on diffie hellman protocol	Implemented
Secure Key Injection Function		
MDImportSessionKey		Not Implemented
MDEncryptData		Not Implemented
CardImportSessionKey		Not Implemented
CardGetSharedKeyHandle		Not Implemented
CardGetAlgorithmProperty		Not Implemented
CardGetKeyProperty		Not Implemented
CardSetKeyProperty		Not Implemented
CardDestroyKey		Not Implemented
CardProcessEncryptedData		Not Implemented

4.4 Function Parameters

For specification details of function parameters, refer to
http://www.microsoft.com/whdc/device/input/smartcard/sc-minidriver.mspx

4.5 Function Calling Description

4.5.1 General

Smart Card Mini Driver interface functions are called by Microsoft Base Smart Card Crypto Provider and Microsoft Smart Card Key Storage Provider. It is not necessary to call the Smart Card Mini Driver interface directly. The way to call Microsoft Base Smart Card Crypto Provider and Microsoft Smart Card Key Storage Provider is the same as other CSP such as the one developed by user itself.

If the user wants to import an asymmetric key pair through Base CSP externally, then the user needs to change the configuration of Base CSP. To do so:

1. Open the registry list
2. Find `HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Cryptography\Defaults\Provider\Microsoft Base Smart Card Crypto Provider`
3. Change the value of `AllowPrivateExchangeKeyImport` and `AllowPrivateSignatureKeyImport` to 1.

Appendix: Terms and Abbreviations

Entry	Description
HYP2003	A smart card-based token with FIPS certification for PKI applications, introduced by Hypersecu Information Systems. It is designed for PKI application systems.
CryptoAPI Interface (CAPI)	An interface used for cryptography operations, provided by Microsoft. It provides cryptographic algorithm encapsulation of equipment irrelevant or implemented by software. With this interface, it is simple to develop PKI applications for data encryption/decryption, authentication, and digital signing on Windows platforms.
Smart Card Mini Driver Interface	An interface used for cryptography operations, provided by Microsoft. It provides cryptographic algorithm encapsulation of equipment irrelevant or implemented by software for Microsoft Base Smart Card Crypto Provider and Microsoft Smart Card Key Storage Provider.
PKCS#11 Interface	A programming interface introduced by RSA. It abstracts the cryptographic device into a universal logic view, Cryptographic Token, for use by upper-level applications, providing device independency and a method of resource sharing.
FIPS	Federal Information Processing Standards, a set of computer security standards developed by the National Institute of Standards and Technology (NIST).