FEITIAN

iR301-U Smart Card Reader

V1.0

Feitian Technologies Co., Ltd.

Website: www.FTsafe.com

Revision History:

Date	Revision	Description
Jul. 2012	V1.0	Release of the first version

Software Developer's Agreement

All Products of Feitian Technologies Co., Ltd. (Feitian) including, but not limited to, evaluation copies, diskettes, CD-ROMs, hardware and documentation, and all future orders, are subject to the terms of this Agreement. If you do not agree with the terms herein, please return the evaluation package to us, postage and insurance prepaid, within seven days of their receipt, and we will reimburse you the cost of the Product, less freight and reasonable handling charges.

1. Allowable Use – You may merge and link the Software with other programs for the sole purpose of protecting those programs in accordance with the usage described in the Developer's Guide. You may make archival copies of the Software.

2. Prohibited Use – The Software or hardware or any other part of the Product may not be copied, reengineered, disassembled, decompiled, revised, enhanced or otherwise modified, except as specifically allowed in item 1. You may not reverse engineer the Software or any part of the product or attempt to discover the Software's source code. You may not use the magnetic or optical media included with the Product for the purposes of transferring or storing data that was not either an original part of the Product, or a Feitian provided enhancement or upgrade to the Product.

3. Warranty – Feitian warrants that the hardware and Software storage media are substantially free from significant defects of workmanship or materials for a time period of twelve (12) months from the date of delivery of the Product to you.

4. Breach of Warranty – In the event of breach of this warranty, Feitian's sole obligation is to replace or repair, at the discretion of Feitian, any Product free of charge. Any replaced Product becomes the property of Feitian.

Warranty claims must be made in writing to Feitian during the warranty period and within fourteen (14) days after the observation of the defect. All warranty claims must be accompanied by evidence of the defect that is deemed satisfactory by Feitian. Any Products that you return to Feitian, or a Feitian authorized distributor, must be sent with freight and insurance prepaid.

EXCEPT AS STATED ABOVE, THERE IS NO OTHER WARRANTY OR REPRESENTATION OF THE PRODUCT, EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

5. Limitation of Feitian's Liability – Feitian's entire liability to you or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price you paid for the unit of the Product that caused the damages or are the subject of, or indirectly related to the cause of action. In no event shall Feitian be liable for any damages caused by your failure to meet your obligations, nor for any loss of data, profit or savings, or any other consequential and incidental damages, even if Feitian has been advised of the possibility of damages, or for any claim by you based on any third-party claim.

6. Termination – This Agreement shall terminate if you fail to comply with the terms herein. Items 2, 3, 4 and 5 shall survive any termination of this Agreement.

Contents

Chapter 1. Overview

This chapter describes how to develop ePass2002Auto applications, including the development interfaces supported by the product (ePass2002Auto) and how to develop applications based on these interfaces. This chapter covers the following topics:

- Supported application development interfaces
- Development of applications using MS CryptoAPI interfaces
- Development of applications using PKCS#11 interfaces

1.1 Application Development Interfaces

The product is provided with two sorts of application development interfaces, PKCS#11 and CSP for Microsoft CryptoAPI 2.0, which can be used to develop your PKI (Public Key Infrastructure) applications. These interface standards are supported broadly in the industry. The product works with applications compatible with these interface standards without further development.

1.2 Development of Applications Using MS CryptoAPI Interfaces

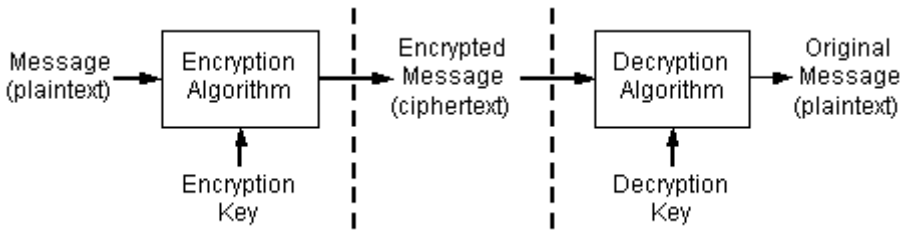
Microsoft CryptoAPI is provided under the Win32 platform for developers to design data encryption and security applications. The CryptoAPI set of functions involve basic ASN.1 encoding/decoding, hash, data encryption/decryption, digital certificate management etc. The data encryption/decryption can be achieved by symmetrical or public key algorithms. All of 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 satisfaction of 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.

1.2.1 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 be the plain text. This process is demonstrated as follows:



The concepts of data encryption and decryption are fairly 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 of 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.

1.2.2 Identity Authentication

The prerequisite of secure communication is that both parties on the two sides of the communication definitely 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 the credential is the passport used to determine the true identify 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 such a kind of credential. It is a valid 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. 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. CA uses trusted service program to handle applicant's requests and certificate issues.

1.2.3 Integrity Check

All the information transferred by unsafe media faces the risk of being tampered. The seal is used as a tool for an integrity check in the real world. For example, the unrecoverable package and intact seal could testify that the item inside is kept unchanged 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 changed. To build the integrity check mechanism, both

the information and the verification information for it (which is 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.

1.2.4 CSP and Encryption Process

CryptoAPI functions use "Cryptographic Service Providers" (CSPs) to perform the data encryption/decryption and encryption key storage management. All of the CSPs are independent modules. Theoretically, CSPs should be independent of specific applications, say; each of the applications could use any CSP. But 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 controlled strictly so 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. 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 is applied such as finger print scanning.

The simplest CSP is comprised of a Win32 Dynamic Linking Library (DLL) and a signature file. Only by providing the correct signature file, the CSP can 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.

1.2.5 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 successfully call, the function returns the CSP operating handler. Application can use the handler to access the CSP and the key container in the CSP.

1.2.6 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. Their 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**

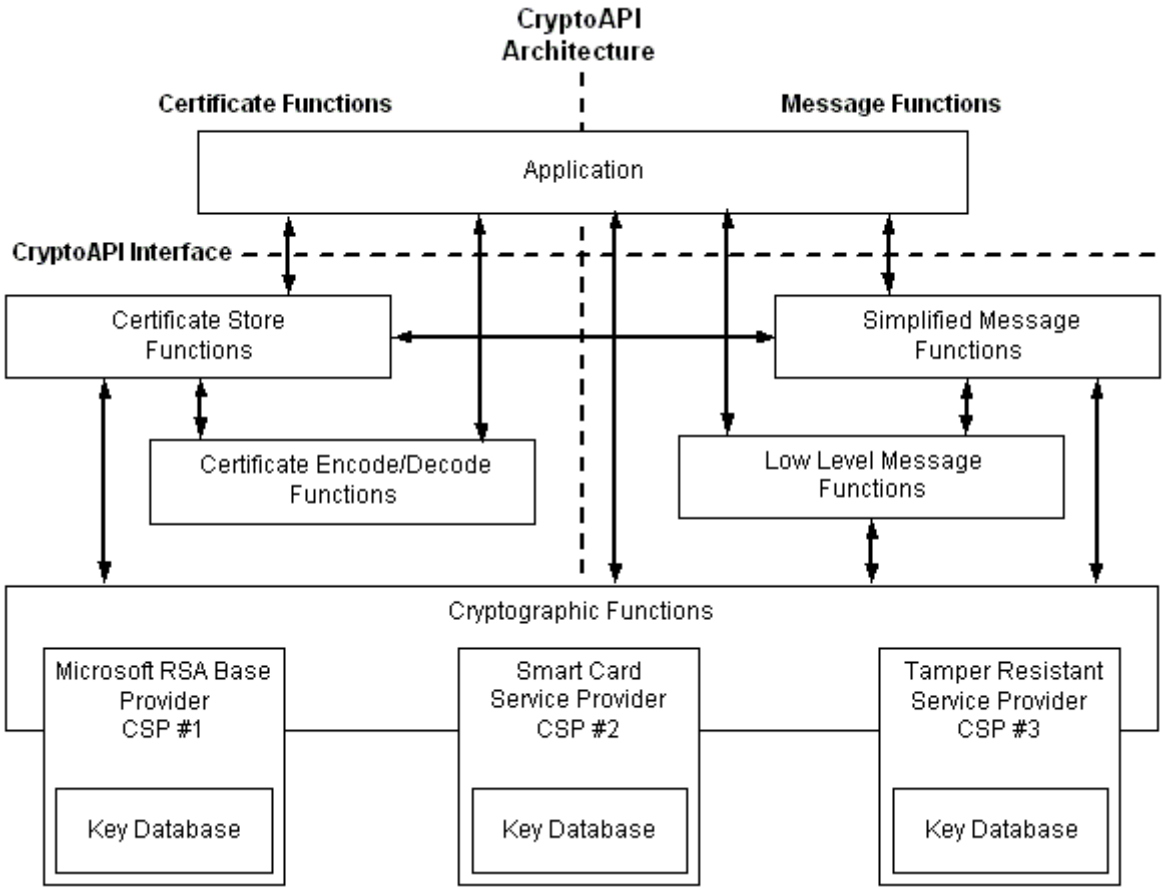
Used to manage digital certificate sets.

- **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.



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

1.3 Development of Applications Using PKCS#11 Interfaces

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

PKCS#11 standard is one of the PKCS standard set. PKCS#11 standard (also known as "Cryptoki") is used to resolve the compatibility problems of interaction between different manufacturers and public key applications. It defines a uniform programming interface model – Cryptoki tokens. The ePass2002Auto PKCS#11 interfaces are compliant with the PKCS#11 standard version 2.20.

Before programming with the ePass2002Auto PKCS#11 interfaces, developers should be familiar with the PKCS#11 standard. The standard's related documents can be downloaded from the RSA website at http://www.rsa.com/rsalabs/node.asp?id=2133

Chapter 2. CSP Module

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

- Description of CSP Module
- Supported Algorithms
- Function Implementation
- Parameters of the Functions
- Function Calling

2.1 Description of CSP Module

The product 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: EnterSafe ePass2002Auto CSP v1.0

  The hardware type is indicated in the name.

2.1.2 Features

The CSP module of the product 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 2000, Windows XP, Windows Server 2003, Windows Vista, Windows 7 and Windows 2008 (32/64 bits)
- Seamlessly compatible with existing Windows platform applications, such as Office encryption and decryption, Internet Explorer webpage and SSL website login, secured emails of Outlook (Express) etc.

2.2 Supported Algorithms

The supported algorithms for the CSP module are listed below:

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	512	2048	Signature verification
CALG_RSA_KEYX or AT_KEYEXCHANGE	1024	512	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 there is the interface in CSP module, but it is not implemented. "Not Supported" indicates that there is no that interface in 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 Parameters of the Functions

2.4.1 CPAcquireContext

——dwFlags

It supports the following values: CRYPT_VERIFYCONTEXT, CRYPT_NEWKEYSET, CRYPT_DELETEKEYSET and CRYPT_SILENT; No CRYPT_MACHINE_KEYSE processing cases;

——pszContainer

It 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.

2.4.2 CPGetProvParam

——dwParam

It 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.

2.4.3 CPReleaseContext

——dwFlags

Its value must be zero.

2.4.4 CPSetProvParam

——dwParam

It supports the following values: PP_KEYEXCHANGE_PIN and PP_SIGNATURE_PIN. Logout if pbData is NULL.

It does not support other values.

——*dwFlags*

Not checked.

2.4.5 CPDeriveKey

——*Algid*

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

——*dwFlags*

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

2.4.6 CPDestroyKey

Further description not required.

2.4.7 CPDuplicateKey

Not supported.

2.4.8 CPExportKey

——*dwBlobType*

It 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.

2.4.9 CPGenKey

——*Algid*

It 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.

2.4.10 CPGenRandom

Further description not required.

2.4.11 CPGetKeyParam

This function 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*

It must be zero.

2.4.12 CPGetUserKey

——*dwParam*

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

2.4.13 CPImportKey

——*pbData*

This keyBlob supports SIMPLEBLOB, PUBLICKEYBLOB and PRIVATEKEYBLOB.

——*dwFlags*

Ignored.

2.4.14 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*

It must be zero.

2.4.15 CPDecrypt

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

——*dwFlags*

It must be zero.

2.4.16 CPEncrypt

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

——*dwFlags*

It must be zero.

2.4.17 CPCreateHash

——*Algid*

It supports the following algorithms: CALG_MD2, CALG_MD5, CALG_SHA1 and CALG_SSL3_SHAMD5.

——*dwFlags*

It must be zero.

2.4.18 CPDestroyHash

Further description not required.

2.4.19 CPDuplicateHash

Not supported.

2.4.20 CPGetHashParam

——*dwParam*

It supports the following values: HP_ALGID, HP_HASHSIZE and HP_HASHVAL.

——*dwFlags*

It must be zero.

2.4.21 CPHashData

——*dwFlags*

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

2.4.22 CPHashSessionKey

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

2.4.23 CPSetHashParam

——*dwParam*

It supports only the value of HP_HASHVAL.

——*dwFlags*

It must be zero.

2.4.24 CPSignHash

——*sDescription*

Ignored.

——*dwFlags*

It supports only the value of CRYPT_NOHASHOID. Other values are ignored.

2.4.25 CPVerifySignature

——*sDescription*

Ignored.

——*dwFlags*

It does not support any value.

2.5 Description of Function Calling

2.5.1 General

The function firstly called is CPAcquireContext among all CSP functions. 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 of 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.

2.5.2 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.

Chapter 3. PKCS#11 Module

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

- Description of PKCS#11 Module
- Supported PKCS#11 Objects
- Supported Algorithms
- Supported PKCS#11 Interface Functions

3.1 Description of PKCS#11 Module

The PKCS#11 interfaces are provided in a Win32 dynamic linking library (DLL), which can be accessed through a static (using .lib file) or dynamic link. The following is the files relating 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_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)
es_eps2k2a.lib	\Lib (PKCS#11 interface library)

es_eps2k2a.dll is the core library file for ePass2002Auto. 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.

3.2 Supported PKCS#11 Objects

The ePass2002Auto 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 the 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 accessing privileges. All the objects can be divided into two types according to their different accessing 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. Application can decide one object is public or private by its actual usage. Importantly, the private memory area and the public memory area are all limited to certain capacity, and independent of each other. These two storage zones are independent. Applications must balance their size appropriately. Once the size is determined during the initialization of the product, it could not be changed later.

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	Package
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_DES_OFB64	√				
CKM_DES_OFB8	√				
CKM_DES_CFB64	√				
CKM_DES_CFB8	√				
CKM_DES3_KEY_GEN				√	
CKM_DES3_ECB	√				√
CKM_DES3_CBC	√				√
CKM_SSF33_ECB	√				√
CKM_SSF33_CBC	√				√
CKM_SCB2_KEY_GEN				√	
CKM_SCB2_ECB	√				√

CKM_SCB2_CBC	√				√
CKM_MD2			√		
CKM_MD5			√		
CKM_SHA_1			√		
CKM_SHA224			√		
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	512, 1024, 2048 bits
CKM_RC2_KEY_GEN	1 - 128 bytes
CKM_RC4_KEY_GEN	1 - 256 bytes
CKM_DES_KEY_GEN	8 bytes
CKM_DES3_KEY_GEN	24 bytes
CKM_SCB2_KEY_GEN	16 bytes
CKM_GENERIC_SECRET_KEY_GEN	1 - 256 bytes

3.4 Supported PKCS#11 Interface Functions

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

Some of the interfaces defined in the PKCS#11 standard 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 standard.

In the PKCS#11 standard, a smartcard reader is called a "slot". Since the product does not need such a reader when it is used, the "slot" is only a virtual device. But there is no difference for applications on this point.

The following is a list of all interfaces defined in the PKCS#11 2.20 standard:

Name	Description	Supported or Not
Basic Functions		
C_Initialize	This function initializes the library. It must be called before calling other functions with the only exception being the C_GetFunctionList function.	Implemented
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
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
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	Implemented

	operation.	
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
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
C_UnwrapKey	Un-wrap a private key or encryption key.	Implemented

Random Number Generation Functions		
C_SeedRandom	Add a seed to the random generator.	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

Appendix: Terms and Abbreviations

Entry	Description
ePass2002Auto	A portable cryptographic device introduced by Feitian, with high performance, high security, flexibility, low cost features, and automatic installation of the middleware.
Token	A collective name of cryptographic devices, including the smart card, and other devices that stores passwords or certificates.
USB Token	A cryptographic device with a USB interface.
CryptoAPI Interface (CAPI)	An interface used for cryptography operations, provided by Microsoft. It provides a package of cryptographic algorithms, independent of the device, or implemented by software. With this interface, it is easy to develop PKI applications for data encryption/decryption, authentication and signature on Windows platforms.
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 manner of resource sharing.