SecureToken ST3

Developer's Guide

Version 1.2

http://www.securemetric.com

Disclaimer

The information contained in this Software Development Kit is for general guidance on using SecureMetric products. While SecureMetric has made every attempt to ensure the information contained in this Software Development Kit is believed to be accurate and reliable, however SecureMetric is not responsible for any errors or omissions, nor for any infringement of patents or other rights of third parties resulting from its use. Any third party's trademarks or trade names are the property of their respective owners.

All the Products, Software Samples, Tools, Utilities and Documents contained in this Software Development Kit are the Intellectual Properties of SecureMetric. SecureMetric reserves the right to make changes, updates, amendment, at anytime and without notice.

Revision History:

Date	Version	Description
January 2007	1.0	1st Edition
July 2008	1.1	1st Revision
Jan 2009	1.2	2^{nd} Revision

Software Developer's Agreement

All Products of SecureMetric Technology Sdn. Bhd. (SecureMetric) including, but not limited to, evaluation copies, diskettes, CD-ROMs, hardware and documentation, and all future orders, are subjects to the terms of this Agreement. If developers 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 developers the cost of the Product, less freight and reasonable handling charges.

1. **Allowable Use** - Developers 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. Developers 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. Developers may not reverse engineer the Software or any part of the product or attempt to discover the Software's source code. Developers 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 SecureMetric provided enhancement or upgrade to the Product.

3. **Warranty** - SecureMetric 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 developers.

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

Warranty claims must be made in writing to SecureMetric 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 SecureMetric. Any Products that developers return to SecureMetric, or a SecureMetric 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 SecureMetric's Liability** - SecureMetric's entire liability to developers or any other party for any cause whatsoever, whether in contract or in tort, including negligence, shall not exceed the price developers 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 SecureMetric be liable for any damages caused by developers failure to meet developer's obligations, nor for any loss of data, profit or savings, or any other consequential and incidental damages, even if SecureMetric has been advised of the possibility of damages, or for any claim by developers based on any third-party claim.

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

CE Attestation of Conformity

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

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

Test report No.:	SHEMO081100414IT01
Test standards:	EN 55022: 2006
	EN 55024: 1998/A1 :2001/A2 :2003
	EN 61000-3-2: 2006
	EN 61000-3-3: 1995/A1 :2001/A2 :2005

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

FCC certificate of approval

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

Test report No.:	SHEMO081100414IT02
Test standards:	CFR 47 Part 2: 2007
	CFR 47 Part 15: 2007
	ANSI C63.4: 2003

USB

This equipment is USB based.

WEEE

Dispose in separate collection.

Contents

1 Overview ... 1

1.1 SecureToken ST3 Application Programming Interface ... 1
1.2 Developing with MS CryptoAPI ... 1
1.2.1 Information Concealment ... 2
1.2.2 Identity Authentication ... 3
1.2.3 Integrity Check ... 4
1.2.4 CSP and Encryption Process ... 4
1.2.5 CSP Context ... 5
1.2.6 CryptoAPI Architecture ... 6
1.3 Developing SecureToken ST3 Applications with PKCS#11 ... 7

2 CSP Module ... 9

2.1 Description ... 9
2.1.1 Profile ... 9
2.1.2 Features ... 9
2.2 Supported Algorithms ... 10
2.3 Function Implementation ... 11
2.4 Parameters of the Functions ... 12
2.4.1 CPAcquireContext ... 12
2.4.2 CPGetProvParam ... 12
2.4.3 CPReleaseContext ... 13
2.4.4 CPSetProvParam ... 13
2.4.5 CPDeriveKey ... 13
2.4.6 CPDestroyKey ... 14
2.4.7 CPDuplicateKey ... 14
2.4.8 CPExportKey ... 14
2.4.9 CPGenKey ... 14
2.4.10 CPGenRandom ... 14
2.4.11 CPGetKeyParam ... 14
2.4.12 CPGetUserKey ... 15
2.4.13 CPImportKey ... 15
2.4.14 CPSetKeyParam ... 15
2.4.15 CPDecrypt ... 16
2.4.16 CPEncrypt ... 16
2.4.17 CPCreateHash ... 16
2.4.18 CPDestroyHash ... 16
2.4.19 CPDuplicateHash ... 16
2.4.20 CPGetHashParam ... 16
2.4.21 CPHashData ... 17
2.4.22 CPHashSessionKey ... 17
2.4.23 CPSetHashParam ... 17
2.4.24 CPSignHash ... 17
2.4.25 CPVerifySignature ... 17
2.5 Function Calling Notes ... 18
2.5.1 Introduction ... 18
2.5.2 Development Samples ... 18

3 PKCS#11 Module ... 19

3.1 Description ... 19
3.2 Supported PKCS#11 Objects ... 20
3.3 Supported Algorithms ... 21
3.4 Supported PKCS#11 Interface Functions ... 22

1 Overview

This chapter describes the development of SecureToken ST3 applications, including the APIs supported by SecureToken ST3 and the development method for each of these APIs. This chapter covers the following topics:

- SecureToken ST3 APIs
- Developing SecureToken ST3 Applications with MS CryptoAPI
- Developing SecureToken ST3 Applications with PKCS#11

1.1 SecureToken ST3 Application Programming Interface

The development of SecureToken ST3 applications mainly comprises two aspects: developing PKI applications and developing smartcard applications. For developing SecureToken ST3 PKI applications, two main types of APIs are provided. They are PKCS#11 which is compliant with the RSA PKCS#11 API standards and CSP for Microsoft CryptoAPI 2.0 which is compliant with Microsoft CryptoAPI standard. Because these two standards are accepted by most software providers and hardware manufacturers, SecureToken ST3 can be integrated with applications designed based on these two types of API without the need of additional development. Another kind of API is provided for smartcard PC/SC interfaces.

SecureToken ST3 PKI API itself is built based on the PC/SC interface. Software developers can choose one or multiple types of API according to their project requirements.

1.2 Developing with MS CryptoAPI

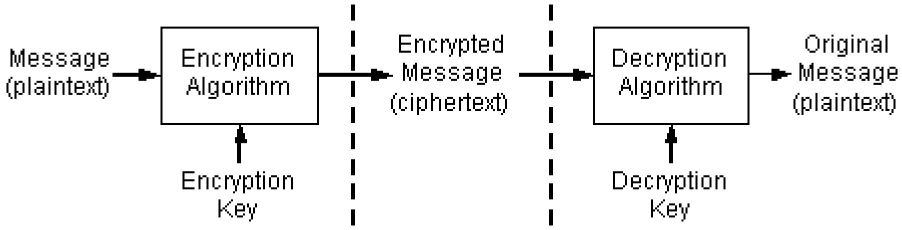
Microsoft CryptoAPI is provided by the Win32 platform for developers designing data encryption and security applications. CryptoAPI comprises a basic ASN.1 encryption / decryption interface, hashing interface, data encryption/decryption interface, digital certificate managing interface and many other important cryptographic features. The data encryption and decryption support symmetrical and public key algorithms. All of the Microsoft applications such as Internet Explorer, Outlook and many other third party applications are developed based on CryptoAPI.

There are three key requirements for secure data transmission over insecure networks: information concealment, identity authentication and integrity check. CryptoAPI does not only satisfy these requirements, but it also provides standard ASN.1 encryption/decryption, data encryption/decryption, digital certificate and certificate storage management, Certificate Trust List (CTL) and Certificate Revocation List (CRL) features.

1.2.1 Information Concealment

The purpose of information concealment is to make sure that content can only be retrieved by authorized people. 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. Herein the word "information" means a set of data. The term "plain text" means the data that has not been encrypted. The term "cipher text" means the encrypted data.

Cipher text could be transferred through insecure media or networks without compromising security. After that, the cipher text could be retrieved back to plain text. This process can be demonstrated as follows:



The concept of data encryption and decryption is fairly simple. To encrypt data, an encryption key is required. The key is like the key used to open a door in function. When performing decryption, a decryption key is required as well. The encryption key and the decryption key could be the same or different.

The encryption key must be stored safely and securely. When provided to other users, the transfer process of the key must be secure. The access to the decryption key must be under strict control, as it can be used to decrypt the encrypted data with the same encryption key.

1.2.2 Identity Authentication

The precondition of secure communication is that both sides of the communication definitely know each other’s identity. The purpose of identity authentication is to verify the true identity of a person or an entity involved in the communication. The document to identify the identity is called a credential. Just like withdrawing money from a bank, people need to prove their identity by using an identity card. Passport is another example. The landing-waiter needs to check the passport to verify the passenger’s true identity. The verification is based on the belief that the passport publishing department had verified the person’s identity. In the above examples, the identity credential exists in the form of a physical document.

Identity authentication sometimes is used to identify that the received data is from the correct source. For example, when A sends some data to B, B needs to verify the data received is really sent by A (other than other people in the name of A). To achieve this authentication request, CryptoAPI provides the digital signature and verification functions to perform the authentication.

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 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 are used to ensure that the goods are unchanged after it left the factory.

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 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:

- Application 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.

- Application 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.

- Application 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 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 special handler containing a special key container. The selection of key container can be specified or be the logon user's default container. CryptAcquireContext can be used to create new key containers.

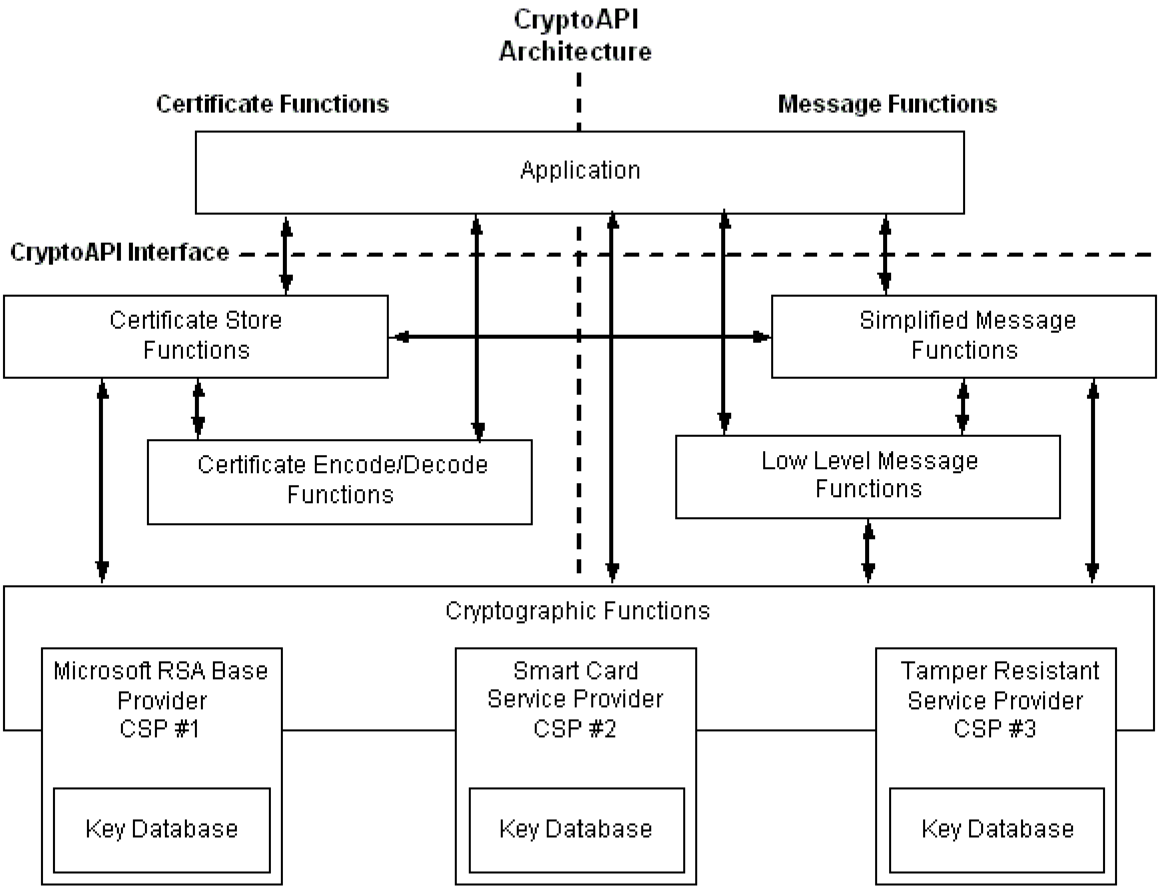
The CSP module itself has a name and a type. For example, Windows operating system's default installed CSP is: Microsoft Base Cryptographic Provider. Its type is PROV_RSA_FULL. Each CSP's name must be different, but their types can be the same.

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. Application can use the handler to access the CSP and the key container in the CSP.

1.2.6 CryptoAPI Architecture

CSP architecture is mainly comprised of five kinds of elements:

- **Cryptographic Functions**
  - Functions are used to link and create CSP handler. These functions allow applications to choose special CSP by specifying its name and type.
  - Key generation functions are used to create and store an encryption key. Their functions include changing encryption mode, initializing encryption vector, etc.
  - Key exchange functions are used to exchange and transmit keys.
- **Certificate Encoding/Decoding Functions**
  - These functions are used to encrypt and decrypt data, including calculating the data's hashing value.
- **Certificate Storage Functions**
  - These functions are used to manage digital certificate sets.
- **Simplified Message Functions**
  - These functions are used to encrypt and decrypt messages and data, sign the message and data, and verify the signature validity of messages and data.
- **Low Level Message Functions**
  - These functions implement the Simplified Message Functions. It provides more detailed control on each message operation.



The prefix of each set of functions has the following format:

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

1.3 Developing SecureToken ST3 Applications with PKCS#11

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 set up 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 PKCS#11 interface of SecureToken ST3 is compliant with the PKCS#11 standard version 2.20.

Before programming with the SecureToken ST3 PKCS#11 interfaces, developers should be familiar with the PKCS#11 standards. The standard's related documents can be downloaded from the RSA Company's website at
http://www.rsa.com/rsalabs/node.asp?id=2133

2 CSP Module

This chapter introduces the CryptoAPI development supported by SecureToken ST3. In particular, the CSP interface name of SecureToken ST3, supported functions and algorithm implementation are described. This chapter covers the following topics:

- Description
- Supported Algorithms
- Function Implementation
- Parameters of the Functions
- Function Calling Notes

2.1 Description

SecureToken ST3 provides standard CSP modules for seamless integration with CryptoAPI applications. SecureToken ST3 CSP module is compliant with Microsoft Crypto Service Provider programming standard. It can be compatible with current and future CryptoAPI applications.

2.1.1 Profile

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: SecureToken ST3 CSP V1.0

Developers may notice that the hardware type of SecureToken ST3 is denoted in the name.

2.1.2 Features

The CSP of SecureToken ST3:

- ✓ Provides secure RSA key-pair storage container;
- ✓ Provides several grouping encryption and hashing algorithms;
- ✓ Supports hardware implemented RSA operation (Maximum: 2048 bits);
- ✓ Supports hardware implemented random number generation;
- ✓ Supports multi-thread access and multi-device management;
- ✓ Supports multi-certificate applications;
- ✓ Is compliant with PKCS#11 data format;
- ✓ Truly supports dual-credential, i.e. there may be two key pairs (AT_KEYEXCHANGE and AT_SIGNATURE) and corresponding certificates in a container;
- ✓ Supports Windows 98 SE or higher operating systems (Except for Windows Vista);
- ✓ Is seamlessly compliant with existing Windows applications, such as Office encryption and decryption, Windows native logon, Windows network domain logon, Microsoft VPN Client logon, Internet Explorer Webpage and SSL Website logon, and Outlook secured emails;
- ✓ Uses fingerprint instead of PIN for authenticating, providing higher security.

2.2 Supported Algorithms

The supported algorithms for the CSP module of SecureToken ST3 are provided in the following list:

Algorithm	Default Length (bit)	Minimum Length (bit)	Maximum Length (bit)	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_SSF33	128	128	128	
CALG_SCB2	128	128	128	
CALG_SHA1	160	160	160	Hashing
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 for SecureToken ST3. 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	Support
Connection Functions		
CPAcquireContext	Create a context for the 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.	Implemeted
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, the function, OffloadModExpo, is defined in the CSP standard. It is not supported by the CSP module of SecureToken ST3 currently.

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 scenario.

——*pszContainer*

It could 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 the analysis on CSP, when the value of dwParam is PP_ENUMALGS or PP_ENUMALGS_EX, enumerating begins in case dwFlags is CRYPT_FIRST; or (the value is 0 or CRYPT_NEXT) enumerate the next. When the value of dwParam is PP_ENUMCONTAINERS, enumerating begins in case dwFlags is CRYPT_FIRST (1) or CRYPT_FIRST|CRYPT_NEXT (3); or enumerates the next when the value is 0 or CRYPT_NEXT. dwFlags does not support CRYPT_MACHINE_KEYSET. When dwParam is other value, do not check the value of dwFlags.

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. If developers have not registered a fingerprint in SecureToken ST3, pbData is an initial password (a string that ends with '\0'). Otherwise, if developers have registered a fingerprint and pbData is not NULL, an error will be returned.

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 cases: (CRYPT_CREATE_SALT | CRYPT_NO_SALT), CRYPT_PREGEN and CRYPT_USER_PROTECTED. Not supported for other cases.

2.4.6 CPDestroyKey

No further details.

2.4.7 CPDuplicateKey

Not supported.

2.4.8 CPExportKey

——*dwBlobType*

It supports only PUBLICKEYBLOB SIMPLEBLOB and PLAINTEXTKEYBLOB, does not support PRIVATEKEYBLOB, OPAQUEKEYBLOB, and 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

No further details.

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

No further details.

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 Function Calling Notes

2.5.1 Introduction

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 of SecureToken ST3 would require authentication to the user. But if developers expect to avoid this dialog box, developers should set the flag CRYPT_SILENT. However, doing so will cause that all operations with access to the private key and protected data fail, because SecureToken ST3 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 interface of SecureToken ST3 and compile and debug them in SDK package under Samples\CryptAPI. Some samples may require Platform SDK from Microsoft.

3 PKCS#11 Module

This chapter introduces the PKCS#11 development supported by SecureToken ST3. In particular, the PKCS#11 interface name of SecureToken ST3, supported functions and algorithm implementation are described. This chapter covers the following topics:

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

3.1 Description

SecureToken ST3 PKCS#11 interface is provided as a Win32 dynamic linking library (DLL). Developers can statically (using .lib file) or dynamically perform the access. The PKCS#11 standard related documents are listed below:

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 (SecureToken ST3 extension algorithms and return values included)
cryptoki_win32.h	\Include\pkcs11 (Required type definition for the first header files under Windows included)
cryptoki_linux.h	\Include\pkcs11 (Required type definition for the first header files under Linux included)
auxiliary.h	\Include\pkcs11 (The definition of SecureToken ST3 related extension functions included)
St3csp11_s.lib	\Lib

St3csp11_s.dll is the core file for SecureToken ST3. It is placed in system directory. It implements all interface functions defined in RSA PKCS#11 standard. If developers need to use these interfaces and all interfaces and definition developers wish to access are PKCS#11 specific, the file cryptoki_win32.h (windows platform) or cryptoki_linux.h (Linux platform) must be included in developer's project. If developers use SecureToken ST3-specific extension functions and algorithms, developers need only to include cryptoki_ext.h. This header file includes all other header files inside. If developers use static connection to gain the access to the SecureToken ST3 PKCS#11 libraries, developers need also to include the project needs to include st3csp11_s.dll in developer's project.

3.2 Supported PKCS#11 Objects

SecureToken ST3 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 SecureToken ST3's security storage zone until being deleted by the application. The other group is session objects. This group of objects is only used in the temporary communication session. Once the session is finished, the object will be deleted as well. The object's attributed which decides the object lifetime is CKA_TOKEN and is a Boolean value. All the objects have this attribute. Developers need to decide different object's lifetimes according to the SecureToken ST3 memory space's size limit. Only the important objects should be stored within the SecureToken ST3 internal memory space.

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 privileges. 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 object's attribute which decides the accessing type is CKA_PRIVATE. It is a Boolean value and all the objects have this attribute. Application can decide one object is public or private by its actual usage. One thing needed to be taken into account is that the sizes of private storage zone and public storage zone are limited. These two storage zones are independent. Application must assign the size well. Once the sizes are decided in the SecureToken ST3 initialization, they can not be changed anymore.

3.3 Supported Algorithms

The following table lists all the encryption algorithms supported by SecureToken ST3 PKCS#11 module:

Cryptographic Algorithm	Encryption /Decryption	Signature Check	Hashing	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_KEY_GEN				√	
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 table lists the key length supported by SecureToken ST3 PKCS#11 module:

Cryptographic Algorithm	Key Length
CKM_RSA_KEY_PAIR_GEN	512, 1024,2048bits
CKM_RC2_KEY_GEN	1-128bytes
CKM_RC4_KEY_GEN	1-256bytes
CKM_DES_KEY_GEN	8bytes
CKM_DES3_KEY_GEN	24bytes
CKM_SSF33_KEY_GEN	16 bytes
CKM_SCB2_KEY_GEN	16 bytes
CKM_GENERIC_SECRET_KEY_GEN	1-256bytes

3.4 Supported PKCS#11 Interface Functions

PKCS#11 is the universal standard for Cryptoki hardware. Different hardware manufacturers may implement it with minor variance.

SecureToken ST3 PKCS#11 interface module also has a little difference:

- C_WaitForSlotEvents function is not fully implemented. It does not support block-calling mode. If applications need this calling mode, it needs to implement the mode by itself.
- Some of the interfaces defined in PKCS#11 standard are not implemented. Once an application calls this kind of interface, the value CKR_FUNCTION_NOT_SUPPORT will be returned.

Note: SecureToken ST3 is just the "token" mentioned in PKCS#11 standard.

PKCS#11 standard calls smartcard reader "slot". As the SecureToken ST3's usage does not require a smartcard reader, the slot is only a virtual device. But for applications, there will be no difference.

The following table lists all the interfaces defined in PKCS#11 2.20 standards:

Name	Description	Support
Basic Functions		
C_Initialize	This function initializes the library. It must be called before calling other functions with the only exception being the	Implemented

	C_GetFunctionList function.	
C_Finalize	This function should be called when finished accessing.	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 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 on the token.	Implemented
C_Logout	Log out the token.	Implemented
Object Management Functions		
C_CreateObject	Create new Cryptoki object.	Implemented
C_CopyObject	Create the copy of object.	Not Implemented
C_DestroyObject	Destroy the object.	Implemented
C_GetObjectSize	Get the size of object.	
C_GetAttributeValue	Get the attributes of the object.	Not 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 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
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	Not Implemented

	operation.	
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	Unwrap a private key or encryption key.	Implemented
Random Number Generation Functions		
C_SeedRandom	Input seed for random generator.	Implemented
C_GenerateRandom	Generate random number.	Implemented
Parallel Management Functions		
C_GetFunctionStatus	It has been deprecated.	Not Implemented
C_CancelFunction	It has been deprecated.	Not Implemented