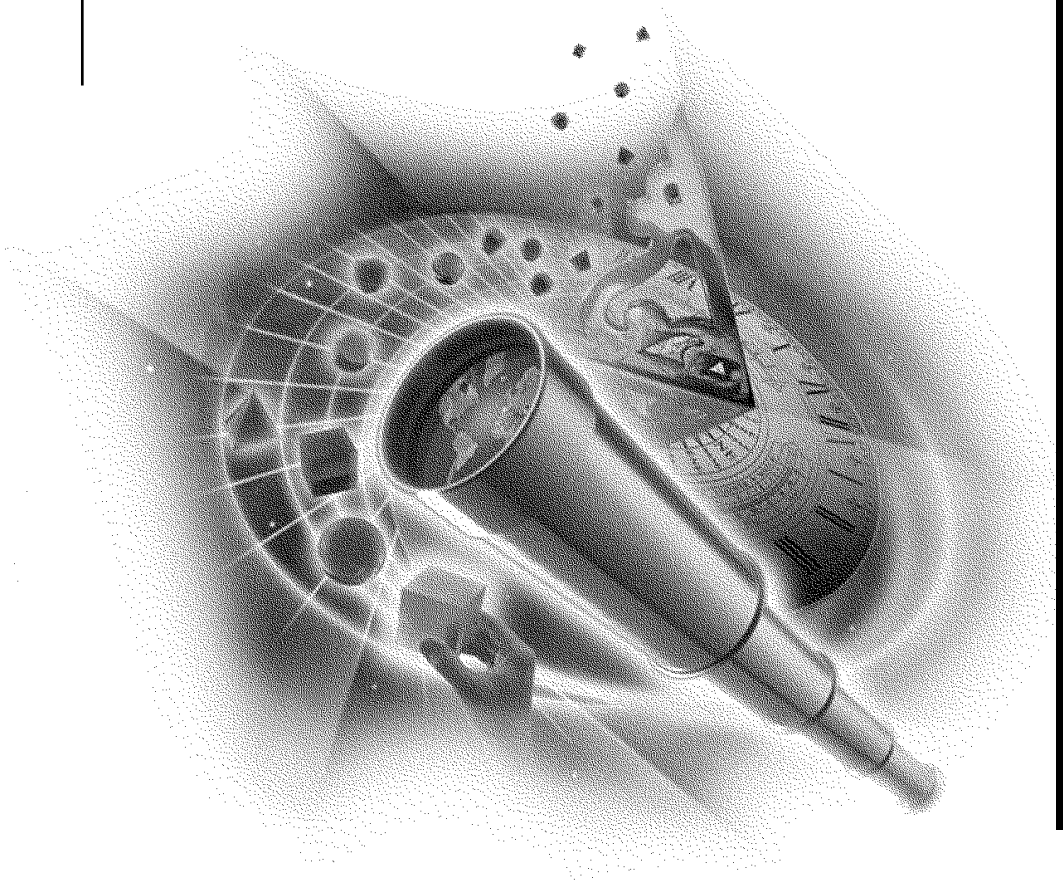
VERSION 2.0

NICI 2.0 Client Security Policy

for Windows 95/98



Novell International Cryptographic Infrastructure (NICI)

Novell®

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

This product may require export authorization from the U.S. Department of Commerce prior to exporting from the U.S. or Canada.

U.S. Patent Nos. 4,555,775; 5,157,663; 5,349,642; 5,455,932; 5,553,139; 5,553,143; 5,594,863; 5,608,903; 5,633,931; 5,652,854; 5,671,414; 5,677,851; 5,692,129; 5,758,069; 5,758,344; 5,761,499; 5,781,724; 5,781,733; 5,784,560; 5,787,439; 5,818,936; 5,828,882; 5,832,275; 5,832,483; 5,832,487; 5,859,978; 5,870,739; 5,873,079; 5,878,415; 5,884,304; 5,893,118; 5,903,650; 5,905,860; 5,913,025; 5,915,253; 5,925,108; 5,933,503; 5,933,826; 5,946,467; 5,956,718; 5,974,474. U.S. and Foreign Patents Pending.

Novell, Inc.
1800 South Novell Place
Provo, Utah 84606
U.S.A.

www.novell.com

Novell Trademarks

For a list of Novell trademarks, see the final appendix of this book.

Third-Party Trademarks

All third-party trademarks are the property of their respective owners.

Contents

Foreword

This document describes the functionality of Novell® International Cryptographic Infrastructure (NICI) in compliance with requirements of the Federal Information Processing Standard (FIPS) 140-1 standard for cryptography.

The cryptographic standards implemented within NICI will help ensure that cryptographic products developed according to FIPS standards will be interoperable industry-wide.

1 Introduction

Novell® International Cryptographic Infrastructure (NICI) consists of a set of modular components that have been implemented on a number of different platforms. Server-oriented versions have been implemented on Novell NetWare®, Microsoft* Windows* NT* 4.0, and Sun* Solaris* 2.6, with other hardware platforms and operating systems in process. Client-oriented versions have been implemented on Windows 95/98 and Windows NT to date.

The present version of the NICI code supports:

- DES (FIPS 46-3 and 81)
- Triple-DES
- SHA-1 (FIPS 180-1)
- RSA (X9.31)

Non-FIPS approved algorithms that also are supported include:

- Diffie-Helman (PKCS#3)
- RSA* encryption/decryption (PKCS#1)
- MD2
- MD4
- MD5
- HMAC-MD5
- HMAC-SHA-1
- RC2
- RC4
- RC5

- CAST128
- PKCS#12 Password Based Encryption (PBE)
- UNIX* Crypt
- LMdigest (CIFS)
- TLS-KeyExchange-RSASign
- NetWarePassword

1.1 Overview of FIPS 140-1 Categories

The Novell NICI 2.0 Client Security Policy for Windows 95/98 conforms to FIPS 140-1 Level 1, as shown in the following table, with category levels tested for the NICI 2.0 Windows 95/98 Client.

Table 1 FIPS 140-1 Test Category Levels

FIPS140-1 Test Category	Level
Cryptographic Modules	1
Module Interfaces	1
Roles and Services	1
Finite State Machine Model	1
Physical Security	1
Software Security	1
Operating System Security	1
Key Management	1
Cryptographic Algorithms	1
EMI/EMC	1
Self Tests	1

2 Cryptographic Channels

FIPS 140-1 defines a cryptographic boundary, and as well as channels though which information is allowed to enter and leave the cryptographic boundary. Defining such channels is normally straightforward for developers of hardware modules, but developers of software modules are faced with the task of choosing an appropriate set of channel definitions.

2.1 Data Input/Output Channel

FIPS 140-1 requires the definition of Data Input/Output (I/O) and Command/Status channel interfaces. NICI defines these interfaces through the Controlled Cryptographic Services API. The API provides the means to input and output data and to determine the status of the module. The Data Input/Output and the Status interface are active only during the User and Crypto Officer States.

2.2 Command/Status Channel

The FIPS 140-1 Control interface is used to initiate the NICI Module. It is activated by the operating system when an application program asks the operating system to attach NICI and causes it to commence operation. It may also be activated when the operating system commands NICI to shut down. Otherwise, it is active only during the User and Crypto Officer States, if and when commands are issued via the API in the form of procedure calls. A selected API call initiates a specific action, which constitutes “control.”

It should be noted that, under this definition of what constitutes a “channel,” such I/O ports that might be considered channels in other contexts are not FIPS 140-1 channels.

In particular, this applies to the internal I/O channel or bus, to any networking or other cards or boards with external interfaces, and to any internal cryptographic processors or accelerators that do not have their own independent I/O (external) ports. This would also apply to the case of a removable Smart Card or removable PCMCIA bus card, which would be considered inside of the cryptographic module boundary when it is in use.

3 Security Requirements

3.1 Cryptographic Modules

NICI consists of a set of software modules designed to run on a wide variety of modern operating systems and hardware platforms. This particular Security Policy document pertains to the NICI configuration, running on a Windows* 95/98 platform, which is a VXD primer. In FIPS 140-1 terms, NICI consists of a set of hardware, software, and firmware that make up a “multi-chip standalone module.”

The cryptographic boundary is effectively the outer cabinet that contains the computer, including the CPU processor(s), any and all storage media (hard disks, diskettes, etc.), any embedded cryptographic accelerators or smart cards, and any network ports or other forms of interfaces. Since NICI must be able to store at least one permanent key, the Key Storage Keys, in order to be able to securely wrap and unwrap other keys, that key is stored in an obfuscated form, along with a backup version.

3.2 Module Interfaces

NICI meets the FIPS 140-1 Level 1 requirements for Roles and Services, including provision of one or more User roles and a Crypto-Operator role.

In the case of a User, all functions are exercised through a common Application Programming Interface (API). The packaging of these systems may vary, depending on the operating systems’ platform characteristics. Access to the NICI functionality is provided by API calls from both C-language and Java* programs.

In the case of a Crypto Operator, some functions are exercised through the same API, but other functions—such as installing the system, installing the license materials, and zeroizing the permanent Key Storage Keys—are carried out by separate programs that only the Crypto Operator can exercise.

3.3 Roles and Services

Novell NICI 2.0 is FIPS 140-1 Level 1 compliant for Roles and Services. The available services are documented in Appendix A, "Appendix A -- CCS API Definitions," on page 25.

3.3.1 User Role

A single User role is supported in NICI. A "User" is an application program, running as a single process (but perhaps multi-threaded), that has been linked with the Novell NICI interface library. After authentication to the User state, the User program is able to perform crypto operations via the API set defined in the Controlled Cryptography Services Software Development Specification (CCS) document.

3.3.2 Crypto Officer Role

A single Crypto Officer role, the NICI Administrator, is supported in NICI. The purpose of the NICI Administrator is to set up, configure, and reconfigure the NICI software. In addition, the Crypto Officer can migrate or clone a given NICI server (or client) from one platform to another, even across operating systems and hardware platforms, and after the process has completed, zeroize the obfuscated Key Storage Keys of the original NICI instance if required.

3.4 Cryptographic Key Management

NICI provides extensive cryptographic key management services and facilities, and is unique in addressing these requirements from a cross-platform, general-purpose networking perspective. Compatible key management is provided for all cryptographic modules, including client and server implementations, on all supported platforms and for all algorithms, including secret key (symmetric) and public key (asymmetric) algorithms. Secret keys and private key are protected from unauthorized disclosure, modification, and substitution. Public keys are protected against unauthorized modification and substitution.

The Cryptography Manager (XMGR) function within NICI is exclusively responsible for implementing all key management functions, enforcing key use policies, and providing algorithm management services to the XLIB and other XMGR layer.

3.4.1 FIPS Approved Key Generation

The G function in the pseudo-random generator described in FIPS 186-2 is constructed using the SHA-1 hash function with b=512. See (http://csrc.nist.gov/fips/fips186-2.pdf). The “mod q” operation was eliminated by choosing $q > 2^{512}$.

The distributed seed material and installation time entropy is thoroughly mixed together using the FIPS-approved PRNG algorithm to create a cryptographic master seed, from which several unique working key generation seeds for each class of cryptographic keys are generated by NICI. Once the different working key generation seeds have been generated, individual keys and random numbers are themselves derived cryptographically using the same FIPS-approved key generation algorithm.

3.4.2 Key Distribution

3.4.2.1 NICI Wrapped Keys

Wrapping of keys is the mechanism that NICI provides for applications to obtain the value of secret or private keys for storage outside of NICI or for distribution among different instances of NICI. Various keys are provided by NICI for wrapping other keys. The same key (or corresponding private key of the same key pair) must subsequently be used to unwrap a wrapped key in order for it to be reloaded into NICI.

The key-management keys discussed below are all generated using algorithms from among the installed XENG modules, and with attributes conforming to the key usage policies that are in effect for key management. Those that are described below as being persistent are stored securely by NICI as an integral part of its infrastructure to persist across system shutdowns and restarts.

No means is provided for unauthorized applications to obtain any of the secret or private key-management keys (persistent or transient, wrapped or unwrapped) for storage or distribution outside of NICI. For purposes of distributing NICI's internal key-management keys as a part of system initialization, interfaces, known as XINIT modules, for wrapping and unwrapping them are provided for use only by other portions of the operating system that are trusted to participate in the initialization of NICI and its environment.

Key-wrapping keys may also be generated at the request of applications, which are then responsible for their secure storage (for example, by wrapping with any of the keys described in this section).

3.4.2.2 NICI Session Keys Session Keys

A unique session key is shared between a NICI client workstation and each NICI server instance. Session keys are intended only for wrapping of keys for distribution between clients and servers or between two servers. Each session key generated by the server is a transient symmetric key.

3.4.2.3 Key Wrapping Attributes

When a key is wrapped for storage or transmission outside of NICI, sensitive attributes such as secret or private key values are encrypted, and an integrity check value is used to protect the integrity of all attributes. Nonsensitive attributes can be stored in the clear outside NICI. (NICI public-key key-wrapping keys are stored or transmitted outside of NICI in X.509-compliant certificates for which NICI itself is the certification authority. These keys may not be used as server or end-user keys.)

An integrity check value, which is calculated cryptographically based on a symmetric wrapping key, ensures both the integrity of the key (that the key's attributes have not been accidentally or intentionally modified) and the authenticity of the key (that it originated in NICI and was not crafted outside of NICI).

However, keys that are wrapped using a public key cannot have their authenticity guaranteed without some additional mechanism that makes use of either a secret or private key whose value is not exposed outside of NICI. For example, a digital signature would serve this purpose. Such signatures are not required as part of the wrapping mechanism because that would excessively limit the flexibility and use of the key distribution mechanism in NICI, as well as the possible performance impact.

Therefore, at the discretion of the application requesting the wrapping, the integrity check value on a wrapped key's attributes may optionally be calculated using one of NICI's internal secret key-management or private CA keys described above, independent of the wrapping key that the application uses to protect sensitive key attributes. If the private CA key is used, a digital signature is then used as the integrity check value in the wrapped key. If a secret key is used, the integrity check value is a form of message authentication code (MAC) and the secret key here is called a sealing key. In either case, any instance of NICI that possesses the same key-management key or corresponding CA public key can then depend on the integrity of the associated attributes when reloading the wrapped key. Otherwise, these attributes must be considered only advisory in nature.

To maintain the integrity of NICI's own protection mechanisms, keys whose authenticity is not assured by one of the mechanisms described here cannot be used to wrap other keys or to generate or verify NICI public-key certificates.

3.4.3 Key Entry and Output

NICI does not possess a manual key entry method; all keys are entered electronically. Aside from the Crypto Operator's role in distributing configuration data (used under the control of the Crypto Operator at installation time), all keys are entered under the User's control via the API interface.

There should seldom, if ever, be a requirement for a User to directly enter into or output from NICI a raw, plaintext private or secret key.

There are two exceptions to this general rule. The first is for compatibility with other systems, where the human user has a personal cryptographic key and no way to securely store it except for a password-based encryption mechanism.

The second is not really a key injection or extraction per se, but rather a protocol-dependent key distribution mechanism which NICI itself does not yet support directly, via a Cryptographic Library (XLIB). The integrity and the confidentiality of such are provided by the protocol.

3.4.3.1 Password-Based Encryption (PBE) Wrapped Keys

Password-Based Encryption (PBE) is frequently required when interfacing with other, non-NICI systems such as browsers, S/MIME e-mail clients, and certain authentication methods. Since many of these applications are software-based, and since most of them run on non trusted platforms such as Windows 95/98, the only economically feasible way of protecting those keys is to use a Password-Based Encryption mechanism.

NICI implements the PKCS #12 recommendation for password-based encryption and decryption. With this scheme, the key to be protected is encrypted in a randomly generated intermediate key of suitable strength (depending on export requirements and algorithm availability). The intermediate key is created by hashing an arbitrarily long password or passphrase entered by the user, and then truncating the key as required to meet the key management policy constraints. PKCS #12 builds into this scheme a deliberate slow-down mechanism that requires hashing and rehashing the password many, many times before decrypting the intermediate key. This is to provide some level of protection against an off-line password guessing attack. The time taken is small by human standards (a second or less) but the amount of computer time required to do an exhaustive search would be very large.

3.4.3.2 Key Injection and Extraction

The NICI CCS API defines key injection and extraction functions, but their use is not recommended.

3.4.3.3 Protocol Support

At the present time, protocol support for unwrapping keys that have been wrapped in a User's private key has been provided for SSL, IPSEC, and IKE.

3.4.4 Key Storage

When keys have been unwrapped within NICI (that is, within the confines of the NICI cryptographic module boundaries), they are kept in the clear (in plaintext form) in order to minimize the latency and overhead when using them.

3.4.4.1 Key Storage Keys

As mentioned previously, the server's Key Storage Keys are written to the operating system, which is protected against unauthorized access. However, because of the importance of this key, it is also thoroughly obfuscated, in a manner intended to require very considerable reverse engineering to break.

Whenever a Key Storage Key is used to wrap another key for storage, the Key ID of that Key Storage Key is included in the wrapped key. In this manner, any previously generated, wrapped, and stored keys will be accessible, even if a new Key Storage Key is generated later. The KeyID contained in the wrapped key format also includes a unique ID to that particular machine and process, in order to help ensure that the correct Key Storage Key is being used to unwrap a particular key. At a minimum, this protects against the possibility that the wrapped key has been moved, migrated, or merged onto a new server, perhaps along with the data it protects, but somehow the correct Key Storage Key has been left behind. The integrity check in wrapped keys will catch this.

If some form of compromise of the Key Storage Key file should occur, all previously generated and wrapped keys on that server would potentially be compromised as well. This is unavoidable in a software-based key management system. However, because of the entropy added at NICI installation time, the attacker would not gain access to the new keys, except by reattacking the Key Storage Key file.

3.4.5 Key Destruction

When the particular NICI context associated with the usage of a set of keys is closed, all keys associated with that context within NICI are zeroized in memory. When NICI itself is closed within a given process, assuming it is closed gracefully and not by a system crash or power outage, all keys in all contexts are zeroized.

The destruction of the current and all previous Key Storage Keys in the Key Storage Keys file should be an extremely rare event, since it would effectively make it impossible to recover any previously wrapped keys. The only time this would be likely to occur would be if a particular machine were to be decommissioned and taken out of service, presumably after all of the information had been migrated to another machine.

Since the ability to zeroize all keys might make possible a very serious Denial of Service attack, NICI does not provide a specific tool or function to cause this to occur. Instead, in this event it is the Crypto Operator's responsibility to perform a complete low-level hardware formatting and reinitialization of the hard disk, thoroughly scrubbing the disk to make certain there is no readable residue. Various commercially available file scrubbing utilities can be used to perform this task.

3.5 Self-Test

NICI conforms to the FIPS 140-1 Level 1 requirements for self-test.

The required startup self-tests are performed every time the NICI is started by the operating system, prior to transitioning to the User state. If the self-tests do not run correctly, NICI will not start, and an error indication will be returned via the API if NICI is called.

3.5.1 Startup Self-Tests

NICI satisfies the requirements for FIPS 140-1 Level 1 for Power-Up Self-Tests

3.5.1.1 Cryptographic Algorithms Test

The SHA1 test runs the known answer tests described in Appendices A and B of FIPS Publication 180-1, Secure Hash Standard. See (http://www.itl.nist.gov/fipspubs/fip180-1.htm) or (http://csrc.nist.gov/fips/fips180-1.pdf).

DES and triple DES run the known answer tests described in NIST Special Publication 800-20, Table A.4, rounds 0 through 18. Both DES and Triple DES are operating in CBC mode with only an eight-byte IV. In the case of triple DES, the key in Table A.4 is repeated three times so as to test Encrypt-Decrypt-Encrypt with the same key in each stage. Similar procedures are in effect for RSA (X9.31).

3.5.1.2 Critical Functions Test

The nature and design of NICI precludes successful completion of the cryptographic algorithm tests and the Software/Firmware tests without all critical functions operating properly. Successful completion of these tests is sufficient to indicate that all critical functions are operating properly.

3.5.2 Conditional Self Tests

The following tests are performed as specified for each test.

3.5.2.1 Pair-Wise Consistency Tests for Public/Private Key Pairs

When a public/private key pair is generated, the key pair is tested for pair-wise consistency. The public key is used to encrypt a plaintext value and checked to ensure that an identity mapping did not occur, and then the private key is used to decrypt that value and the value compared to the original. If the values are not identical, the test fails. If the keys are to be used only for the calculation of a signature, then the consistency is tested by the calculation and verification of a signature. These tests are applied to RSA keys.

3.5.2.2 Software/Firmware Load Tests

At present, the NICI module is a self-contained unit (a device driver, VXD), and no other modules are loaded by NICI. Therefore, these tests are not applicable to NICI.

3.5.2.3 Continuous Random Number Test

The continuous random number generator tests specified in FIPS PUB 140-1. Security Requirements for Cryptographic Modules, Section 4.11.2 (see (http://www.itl.nist.gov/fipspubs/fip140-1.htm) or (http://csrc.nist.gov/fips/fip140-1.pdf)) will be applied to the operating specific random entropy generator routines prior to their being used to generate a cryptographic key, seed, or cryptographic random number. They will be applied independently, both before and after any cryptographic processing to add entropy or whitening. This will test both the entropy generator and the results of the key generation function.

4 Operating Modes

4.1 FIPS-Approved Algorithms

It is the application programmer's responsibility to only use FIPS approved algorithms if FIPS 140-1 Mode is required. See (http://www.itl.nist.gov/fipspubs/fip140-1.htm). The CMVP supports a Web site that lists the current approved FIPS algorithms (http://csrc.nist.gov/cryptval).

A Appendix A -- CCS API Definitions

For complete descriptions, please refer to the *Controlled Cryptography Services Software Development Specifications* document available from Novell.

API	Description
CCS_Init	Initializes the CCS library
CCS_Shutdown	Closes the CCS library
CCS_GetInfo	Return information about the CCS interface
CCS_GetPolicyInfo	Determines the policy constraints on key attributes for a given key type and usage
CCS_GetKMStrength	Returns the key management strength level
CCS_GetRandom	Returns a random number
CCS_GetAlgorithmInfo	Obtain information about a specific algorithm.
CCS_GetAlgorithmList	Obtain information about the algorithms available in the system.
CCS_GetMoreAlgorithmInfo	Obtain variable-length information about an algorithm.
CCS_CreateContext	Create a cryptography context.
CCS_DestroyContext	Destroy a cryptography context.

API	Description
CCS_DestroyObject	Destroy a CCS object.
CCS_FindObjectsInit	Initialize a search for objects that match a template.
CCS_FindObjects	Continue a search for objects that match a template.
CCS_GetAttributeValue	Obtain the value of one or more object attributes.
CCS_SetAttributeValue	Modify the values of one or more object attributes.
CCS_DataEncryptInit	Initialize a data encryption operation.
CCS_Encrypt	Encrypt single-part data.
CCS_EncryptUpdate	Continue a multi-part encryption operation.
CCS_EncryptFinal	Finish a multi-part encryption operation.
CCS_EncryptRestart	Reinitialize an encryption operation.
CCS_DataDecryptInit	Initialize a data decryption operation.
CCS_Decrypt	Decrypt encrypted data in a single part.
CCS_DecryptUpdate	Continue a multi-part decryption operation.
CCS_DecryptFinal	Finish a multi-part decryption operation.
CCS_DecryptRestart	Reinitialize a decryption operation.
CCS_Obfuscate	Obfuscates an input string.
CCS_DeObfuscate	De-obfuscates an input string.
CCS_pbeEncrypt	Encrypt data in a single part using a password and password-based algorithm as described in PKCS#5 or PKCS#12.

API	Description
CCS_pbeDecrypt	Decrypt data in a single part using a password and password-based algorithm as described in PKCS#5 or PKCS#12.
CCS_pbeSign	Generate signature for input data in a single part using a password and password-based algorithm as described in PKCS#12.
CCS_pbeVerify	Verify input data and its signature in a single part using a password and password-based algorithm as described in PKCS#12.
CCS_pbeShroudPrivateKey	Encrypt a PKCS#8 private key using a password and password-based algorithm as described in PKCS#5 or PKCS#12.
CCS_pbeUnshroudPrivateKey	Decrypt and load an encrypted PKCS#8 private key using the password and the password-based algorithm as described in PKCS#5 or PKCS#12.
CCS_LoadPFXPrivateKeyWithPassword	Loads zero or more private keys encrypted in a password from a PKCS#12 PFX structure. See PKCS#12 document for details. Only PKCS#8 private keys are supported.
CCS_LoadPFXCertificateWithPassword	Loads zero or more X.509 certificates and public keys in those certificates from a PKCS#12 PFX structure. The certificates either can be encrypted in a safe bag or can be in plain form. See PKCS#12 and RFC 2459 documents for details.
CCS_DigestInit	Initialize a message-digesting operation.
CCS_Digest	Digest data in a single part.

API	Description
CCS_DigestUpdate	Continue a multi-part message-digesting operation.
CCS_DigestFinal	Finish a multi-part message-digesting operation.
CCS_DigestRestart	Reinitialize a message-digesting operation.
CCS_SignInit	Initialize a signature operation.
CCS_Sign	Sign data in a single part.
CCS_SignUpdate	Continue a multi-part signature operation.
CCS_SignFinal	Finish a multi-part signature operation.
CCS_SignRestart	Reinitialize a signature operation.
CCS_SignRecoverInit	Initialize a signature operation with data recovery.
CCS_SignRecover	Sign data in a single part, with data recovery.
CCS_SignRecoverRestart	Reinitialize a signature operation with data recovery.
CCS_VerifyInit	Initialize a verification operation.
CCS_Verify	Verify data in a single part.
CCS_VerifyUpdate	Continue a multi-part verification operation.
CCS_VerifyFinal	Finish a multi-part verification operation.
CCS_VerifyRestart	Reinitialize a verification operation.
CCS_VerifyRecoverInit	Initialize a signature verification operation with data recovery.
CCS_VerifyRecover	Verify a signature on data in a single part, with data recovery.

API	Description
CCS_VerifyRecoverRestart	Reinitialize a verification operation with data recovery.
IKE_Sign	Sign using an IKE Authentication Phase 1 authentication algorithm. The algorithms and mechanisms are described in RFC 2409: The Internet Key Exchange.
IKE_Verify	Verify using an IKE Authentication Phase 1 authentication algorithm. The algorithms and mechanisms are described in RFC 2409: The Internet Key Exchange.
CCS_GenerateKey	Generate a secret key.
CCS_GenerateKeyPair	Generate a public-key/private-key pair.
CCS_WrapKey	Wrap (i.e. encrypt) a key for storage or distribution external to CCS.
CCS_UnwrapKey	Unwrap (i.e. decrypt) a key.
CCS_InjectKey	This is the raw (i.e., plaintext) key injection function that is used for legacy applications with raw key access, and required to use NICI with their existing raw keys.
CCS_ExtractKey	Extract attributes of a key, including its value (NICI_A_KEY_VALUE) attribute.
CCS_LoadCertificate	Load a public-key certificate, verify its signature and load the resulting public key.
CCS_LoadSelfSignedCertificate	Load a self-signed public-key certificate, verify its signature and load the resulting public key.
CCS_LoadUnverifiedCertificate	Load a public-key certificate and the resulting public key without verifying the certificate signature.

API	Description
CCS_GenerateCertificate	Create and sign a public-key certificate.
CCS_GenerateCertificateFromRequest	Create and sign a public-key certificate whose public key is provided by a PKCS #10 Certification Request.
CCS_GetLocalCertificate	Return a public-key certificate or local portion of the certification path for one of the NICI-predefined public keys.
CCS_GetCertificate	Return a public-key certificate or complete certification path for one of the NICI-predefined public keys.
CCS_GenerateKeyExchangeParameters	This is the parameter generation stage of a key agreement algorithm.
CCS_KeyExchangePhase1	This is the phase 1 of a key exchange algorithm.
CCS_KeyExchangePhase2	This is the phase 2 of a key exchange algorithm.