Novell International Cryptographic Infrastructure (NICI)

NICI 2.7.1 Cryptographic Library FIPS 140-2 Level 2

Security Policy

Version 1.5
March 23, 2007

Novell Inc.

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

www.novell.com
Novell Trademarks

Table of Contents

1 Introduction

The Novell International Cryptographic Infrastructure (NICI) consists of a set of components that have been implemented on a number of different platforms.

This document describes the Security Policy for NICI version 2.7.1 to meet the FIPS 140-2 Level 2 requirements as it has been implemented for the following platforms:

- SuSE Linux Enterprise Server v8 with Service Pack 3 on IBM eServer 325
- Trusted Solaris v8 on Sun Sunblade 100
- Windows 2000 Professional, Server with SP3 and Q326886 on Dell Optiplex GX400

2 Security Policy

The Novell NICI 2.7.1 Cryptography Library Security Policy, conforms to FIPS 140-2 Level 2 as shown in the Table 1.

FIPS140-2 TEST CATEGORY	LEVEL
Cryptographic Module Specification	2
Cryptographic Module Ports and Interfaces	2
Roles, Services, and Authentication	2
Finite State Model	2
Physical Security	N/A
Operational Environment	2
Cryptographic Key Management	2
EMI/EMC	2
Self Tests	2
Design Assurance	2
Mitigation of Other Attacks	N/A

Table 1. **FIPS 140-2 Test Category Levels**

2.1 Cryptographic Modules

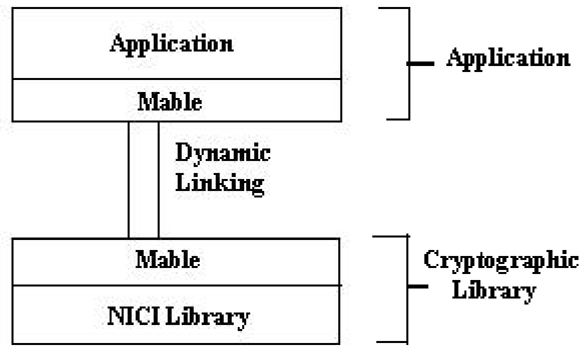
NICI consists of a set of software libraries designed to run on a wide variety of modern operating systems and hardware platforms. In this configuration, NICI is a shared library (.so or dll). In FIPS 140-2 terms, NICI consists of a set of hardware, software, and firmware that make up a "multi-chip standalone module."

The module consists of the following components:

A C2 TCSEC equivalent system consisting of a hardware platform and operating system software. The test system was an EAL4 evaluated Operating System configuration running on the platforms specified above. Configuration details are listed in section 3 (Implementation Guidance) of this document.

NICI 2.7.1 for Trusted Solaris, SuSE Linux Enterprise Server, and Windows 2000 Server consists of a matched upper library, which is linked to the application, and a lower library that is installed on the system.

Figure 1. Software block diagram



The physical cryptographic boundary is defined by the server hardware. The logical cryptographic boundary encompasses the dynamic linked library (ccsw32.dll) in Windows systems and the shared object (libccs2.so.2.7.1) on Unix systems.

Since NICI must be able to store at least one permanent key, the Key Storage Key, in order to be able to securely wrap and unwrap other keys, that key is stored in a DES encrypted form per user, encrypted under an embedded key encryption key, protected by the common criteria evaluated operating system's mechanisms. Stored NICI keys can be zeroized by reformatting the computer's hard drives.

MABLE is the Module Authentication and Binding Library Extensions (patent issued) technology used to authenticate NICI to an application and to provide ongoing binding between an application and NICI as if the application is statically linked to NICI. Upper MABLE is statically linked to an application and contains the challenge generation, certificate verification, and ongoing binding mechanism functions. Lower MABLE is statically linked to NICI and contains the response-to-challenge generation, signature creation, and ongoing binding mechanism functions.

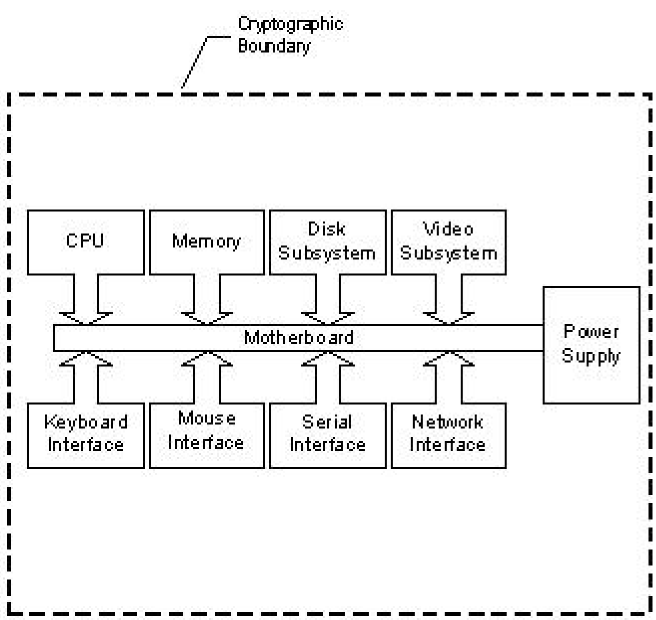


Figure 2. Hardware Block Diagram

All NICI software including executable and data files is protected by the Operating System's access control mechanisms covered by its DAC (Discretionary Access Controls) policy and enforced by TSF (TOE Security Function) installed in accordance with its CC evaluation. The shared object module is protected by file system access controls from unauthorized tampering. The Operating System's file, memory, and process access controls enforced by TSF protect per user NICI configuration files and run-time memory image from tampering and access by other users. Similarly, the NICI configuration file is protected by the operating system's access control mechanisms.

NICI 2.7.1 as evaluated requires the Operating System installed in its CC Evaluated evaluated configuration. See section 3, "Installation Guidance" for further information.

2.2 Module Interfaces

FIPS 140-2 defines a cryptographic boundary, and as well as interfaces through which information is allowed to enter and leave the cryptographic boundary. Defining such interfaces is normally straightforward for developers of hardware modules, but developers of software modules are faced with the task of choosing an appropriate set of interface definitions.

2.2.1 Data Input/Output Interface

FIPS 140-2 requires the definition of Data Input/Output (I/O) and Command/Status interfaces. NICI defines these interfaces through the Controlled Cryptographic Services (CCS) 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 interfaces are active only during the User and Crypto-Officer States.

2.2.2 Command/Status Interface

The FIPS 140-2 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 set.

NICI has the following logical interfaces: data in, data out, control-in, and status out. These interfaces are supported by the API set.

2.3 Roles and Services

Novell NICI 2.7.1 is FIPS 140-2 Level 2 compliant for Roles and Services. NICI also supports concurrent operators; the CC evaluated operating system's security mechanisms maintain the separation of the roles and services.

Operation	User Role	Crypto Officer Role
Install NICI		X
Upgrade NICI		X
Configure NICI		X
Zeroize Keys		X
Encrypt/Decrypt	X	X
Generate Keys and Random Data	X	X
Sign/Verify	X	X

2.3.1 User Role

A "User" is an application program, running as a single or multiple process (perhaps multi-threaded), which has been linked with the Novell NICI interface library. This version of NICI supports multiple processes with different user identities with separation between such multiple instances relying on the access mechanisms provided by the operating system. Each instance of NICI has an identity and a separate memory space with access to a unique set of key materials. Authenticating to the operating system authenticates the application to NICI; as an application cannot use NICI unless it authenticates to the operating system first. After authentication to the User state, the User program is able to perform Cryptographic operations via the API set defined in the Controlled Cryptography Services Software Development Specification (CCS) document.

NICI relies on the authentication mechanism provided by the operating system on which it is being run. Each of the operating systems on which NICI has been validated should be installed in their CC evaluated configuration (see the Crypto-Officer Guidance). This means the authentication mechanism requires a minimum password length of eight alphanumeric characters, providing at a minimum a 1 in 10,000,000 chance of successfully attacking the password. The mechanism strength applies to both the user and crypto-officer roles.

NICI maintains a set of persistent unique keys per user with independent seeding and key generation capability per process. The operating system maintains the separation of such set of keys in a multi-user (multiple operator) setting. Each operator may potentially have more than one process. Each operator is associated with a unique User ID. All processes with the same User ID have access to a unique set of keys.

2.3.2 Crypto-Officer Role

A single Crypto-Officer role is supported in NICI, the NICI Administrator, as the “root” defined on the Operating System. Authenticating to the Operating System assigns the Crypto-Officer, or the NICI Administrator, role to the “root” user. The purpose of the NICI Administrator is to setup, configure, reconfigure and uninstall the NICI software. In addition, the Crypto-Officer can zeroize the Key Storage Keys of the original NICI instance if required, by either rebooting the box or restarting the application to reload the library. The NICI Administrator is also the security administrator as defined by the Operating System.

2.4 Finite State Machine Model

NICI has an embedded finite state machine that is compliant with the FIPS 140-2 specification. The finite state machine is described fully in a separate document that is submitted during the FIPS 140-2 level 2 validation process.

2.5 Physical Security

This is not applicable to NICI as it is a software module.

2.6 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, 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. NICI implements all key management functions, enforces key use policies, and provides algorithm management services to applications.

NICI key use policies are comprised of key usage flags (encrypt, wrap, sign, etc.), key types (TDES, RSA, AES, etc.), and algorithms (RSA, TDES, DSA, etc.).

A key management key must have wrap and key management encrypt key usage flags set in order to wrap keys. Key type must also match the algorithm used with a particular key. For instance, a TDES key can not be used with the AES algorithm. These combined constitute NICI key use policies.

2.6.1 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*).

2.6.2 Key Distribution

NICI key distribution capabilities comply with FIPS 171 options 1 (key exchange role), 4 (MAC), 5 (key and IV generation), 6 (key generation techniques), and 14 (send IV).

2.6.2.1 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 key wrapping. 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 generated with attributes conforming to the key management usage policies. Those that are described below as being persistent per user are stored securely by NICI as an integral part of its infrastructure to persist across system shutdowns and restarts. The persistent NICI key management keys are TDES or RSA keys.

All TDES and HMAC-SHA1 keys are generated and used in FIPS mode. NICI uses HMAC-SHA1 to provide integrity of persistent keys in FIPS mode. NICI uses RSA digital signatures to sign certificates and to encrypt keys in FIPS mode. NICI does not use RSA for data encryption in FIPS mode. No means is provided for unauthorized applications to obtain any of the secret or private key-management keys for storage or distribution outside of NICI.

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).

NICI keys are listed in the following table. SENSITIVE is an attribute of a key set at key generation time, and EXTRACT is a key usage flag. They are enforced by the NICI key use policies. Other NICI key attributes and key usage flags are described in the *CCS Software Development Specification* document.

Key Name	Key Type / Algorithm	Key Usages	Description	Storage and Zeroization
STORAGE	Triple-DES	WRAP, UNWRAP, SENSITIVE	Symmetric key wrapping key, generated and maintained by NICI. FIPS approved.	Stored on disk in the xmgrcfg file in the /var/opt/novell/nici directory. It is stored in an obfuscated format, but for FIPS purposes it is stored in plain text.
SESSION	DES / Triple-DES	WRAP, UNWRAP, SENSITIVE	Symmetric key-wrapping key per connection between a client and server. Generated by NICI, present while the connection is active. FIPS approved.	Stored in the memory. Zeroized when the application closes the context associated with the key.
CA	RSA	SIGN, VERIFY, SENSITIVE	NICI's machine-unique CA key-pair, 1024 bits. Generated and maintained by NICI. Not FIPS approved.	It stored in the xmgrcfg file in the /var/opt/novell/nici/ directory. It is stored in an obfuscated format, but for FIPS purposes it is stored in plain text.
PARTITION	DES / Triple-DES	WRAP, UNWRAP, SENSITIVE	Security Domain Keys, key wrapping purposes only. Generated and maintained by NICI. FIPS approved.	Stored on disk in the nicisdi.key file in the /var/opt/novell/nici directory. This is encrypted with the Storage key. Zeroized when the storage key is deleted or lost .
FOREIGN	Any	EXTRACT and any, but not SENSITIVE	Use of these keys are not FIPS 140-2 approved. Generator unknown, maybe NICI.	Could be stored in memory or disk depending on the application.
XKEY	Random Value	Used for seeding the FIPS approved PRNGs.	Generated by NICI using an entropy collection method. Only use as a seed value for random number generation.	Stored in memory.
Other	DES, Triple-DES, DSA, FIPS-RSA (signature), FIPS-RSA (key distribution), HMAC-SHA1 (signature)	Any, but not EXTRACT	These keys are generated by NICI using a FIPS approved algorithm. FIPS approved.	Stored in memory.
Other	RSA (encryption)	Any, but not	NICI-generated, but not FIPS	Stored in memory

		EXTRACT	approved.	
Authentication Data (password)	Alpha-numeric password	Used by operating system for user authentication	Externally generated passwords for operating system authentication.	Stored on disk in obfuscated format and controlled by operating system. Stored in plaintext for FIPS purposes.

It is the application's responsibility to use the FIPS approved APIs, algorithms, and keys to maintain the FIPS 140-2 mode of operation. Use of any one of the non-FIPS algorithms or non-FIPS approved APIs would invalidate the FIPS mode of operation.

2.6.2.2 NICI Session Keys

Starting with version 2.7.0, NICI operates in a server mode. It supports NICI operating in a client mode from any previous versions. A unique session wrapping key is shared between a NICI client instance and a NICI server instance. NICI Session wrapping keys are intended only for wrapping of keys for distribution between clients and servers, or between two servers. Each session wrapping key is a transient symmetric key. Session wrapping keys can not be extracted in encrypted or unencrypted form for any kind of persistent storage outside of NICI; they are in-memory transient keys useable for key wrapping and unwrapping during the lifetime of a session as defined by the application (typically a network connection to a server/client). The terms server mode and client mode refer to the mode in which a user application operates and does not have any affect on the module's operation.

2.6.2.3 Key Wrapping Attributes

Key wrapping is a secure way of transferring keys in and out of NICI. It has confidentiality, a mechanism for the key value and an integrity mechanism for all attributes.

A wrapped key includes all attributes of a key, such as key usage flags, key ID, and key value. The key value attribute is encrypted in the wrapping key to provide confidentiality. A SHA1 hash is computed over the entire wrapped key (attributes and the encrypted key value), and is encrypted by the wrapping key to generate a MAC. This provides integrity of the wrapped key. The user specifies the encryption algorithm (wrapping algorithm) and the wrapping key.

NICI public-key key-wrapping keys may be 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.

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.

At the discretion of an application requesting the wrapping, the integrity check, such as HMAC-SHA1 or a DSA signature, on the wrapped key's attributes may optionally be calculated using a key management key, independent of the wrapping key that the application chooses to protect sensitive key attributes. 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 internal NICI keys.

2.6.3 Key Entry and Output

NICI does not possess a manual key entry method; all keys are entered electronically. Aside from the Crypto-Officer'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.

Typical key entry to NICI is done via key unwrapping, i.e., by decrypting the key value, and verifying the integrity of the attributes associated with the key. NICI maintains a set of key wrapping keys for this purpose. See Key Storage Keys section for details.

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, except for compatibility with legacy systems.

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. The integrity and the confidentiality of such keys are provided by the protocol.

Raw access to all keys are controlled by three independent controls in NICI: 1) Key's usage must have EXTRACT bit set, 2) Key's usage must not have the SENSITIVE attribute set, and 3) The user must call the *CCS_ExtractKey* API. An application does not have access to plaintext secret keys unless it calls the *CCS_ExtractKey* API.

2.6.3.1 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 manyof 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. The password-based encryption API set is not FIPS 140-2 approved.

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 import requirements and algorithm availability). Hashing 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. As noted above, use of the password-based encryption API set is not FIPS 140-2 approved.

2.6.3.2 Key Injection and Extraction

The NICI CCS API documentation defines key injection and extraction functions,, but their use would invalidate the FIPS 140-2 mode of operation.

2.6.3.3 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/TLS and IPSEC. Use of these APIs is not approved for the FIPS 140-2 mode.

2.6.4 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. Access to the memory managed by the CC evaluated operating system is deemed adequate for FIPS 140-2 level 2 security.

2.6.4.1 Key Storage Keys

As mentioned previously, per-user Key Storage Keys are written to the operating system supported storage, which is protected against unauthorized access by the operating system's mechanisms.

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 Keys is generated later. The Key ID 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 system, 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.

Should some form of compromise of the Key Storage Keys file occur, that requires violation of the CC operating system's security mechanisms, 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 instantiation time, the attacker would not gain access to the new keys, except by re-attacking the Key Storage Key file, which effectively requires circumvention of the CC evaluation of the operating system.

2.6.5 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 and destroyed 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.

2.7 Cryptographic Algorithms

NICI 2.7.1 supports the following FIPS approved algorithms:

Algorithm	Key Size(s)	Algorithm Certificate #
AES (FIPS 197)	128 bits, 256 bits	432
TDES (FIPS 46-3 and 81)	112 and 156-bits	461
SHA (SHA-1, SHA 192, SHA 256, SHA 384) (FIPS 180-1)	128, 256, 384,512 bit hashes	502
RSA (PKCS#1)	1024, 2048, 4096 bits	163
DSA (FIPS 186-2)	1024 bits	179
HMAC SHA-1 (FIPS 198)	Keyed Hash Algorithm	204

Non-FIPS approved algorithms that also are supported include:
1. Diffie-Hellman (PKCS#3)
2. ECDSA
3. EC DH (160-384bits)
4. DES
5. MD2 (RFC 1319)
6. MD4 (RFC 1320)
7. MD5 (RFC 1321)
8. HMAC-MD5 (RFC 2104)
9. RC2 (RFC 2268)
10. RC4
11. RC5 (RFC 2040)
12. CAST128 (RFC 2144)
13. Password Based Encryption, six algorithms (PKCS#12)
14. UNIX* Crypt
15. LMdigest (CIFS)
16. TLS-KeyExchange-RSASign
17. X9.62 RNG
18. NetWarePassword.(Novell)

The Diffie-Hellman algorithm, though not FIPS approved is allowed in a FIPS mode of operation.

The following EC key sizes are supported but not tested for FIPS.

ECDSA (FIPS 186-2)	Special Curves specified in FIPS 186-2 for the key sizes and Curves defined in ANSI X9.62 – 163, 176, 191, 192, 208, 224, 233, 239, 256, 272, 283, 304, 359, 368, 384, 409, 521, and 571 bits

When only FIPS approved algorithms are used, NICI can be said to be functioning in FIPS mode. If any non-FIPS approved algorithm (numbers 2 and 4-18) is used, NICI is running in non-FIPS mode. It is the application programmer’s responsibility to enforce FIPS and Non-FIPS modes of operation.

2.8 EMI/EMC

This is not applicable to NICI as it is a software module.

2.9 Self-Tests

NICI conforms to the FIPS 140-2 Level 2 requirements for self-test.

The required start-up 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.

2.9.1 Startup Self-Tests

NICI satisfies the requirements for FIPS 140-2 Level 2 for Power-up Self-Tests.

2.9.1.1 Cryptographic Algorithms Test

Known answer tests are performed for RSA, DSA, ECDSA, TDES, AES, HMAC-SHA-1 and PRNG implementation upon startup. Pair-wise consistency tests are performed for RSA, DSA and ECDSA upon startup.

2.9.1.2 Software/Firmware Test

NICI performs a software integrity self check at startup.

On Windows and Unix like platforms, NICI complies with FIPS 140-2 by storing a DAC for the NICI shared library when the module is installed. This DAC is stored, in the configuration file on unix like systems and in the registry on windows,each of which is under the control of the Crypto-Officer and protected operating system's file system permissions. The DAC for the shared library is calculated at initialization and compared with the stored version. NICI fails initialization if the DAC does not match. NICI is using HMAC-SHA1 to compute the DAC.

In all these cases, if any check fails NICI will give out an error message and will become unusable.

2.9.1.3 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.

2.9.2 Conditional Self Tests

The following tests are performed as specified for each test:

2.9.2.1 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 tests 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, DSA and ECDSA keys.

2.9.2.2 Continuous Random Number Test

The continuous random number generator tests specified in FIPS *PUB 140-2. Security Requirements for Cryptographic Modules*, Section 4.11.2 (see http://www.itl.nist.gov/fipspubs/fip140-2.htm or http://csrc.nist.gov/fips/fips140-2.pdf), are 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 are 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, etc.

3 Installation Guidance

3.1 Crypto-Officer Guidance

The following steps should be followed by the Crypto-Officer to ensure that NICI is installed in a FIPS 140-2 Level 2 compliant

mode:
– The system supported installation mechanism for installing NICI should be obtained from Novell's web site. (http://www.novell.com/)
– The operating system should be installed and set up as per the guidelines below.
– NICI should always be installed using the system supported installation mechanism. Manually copying the configuration files and libraries into their respective locations does not ensure that the permissions on the files are set properly and should not be done.

3.1.1 FIPS 140-2 Level 2 Installation Requirements

For NICI version 2.7.1 for to be compliant with the FIPS 140-2 Level 2 specification the following requirements must be met.

(1) NICI must be installed on one of the CC evaluated computing platforms listed in section 1 according to the manufacturers Trusted Facilities Manual.

(2) NICI must be installed using the standard NICI 2.7.1 Installation Program to insure that file permissions are correctly set.

3.1.2 Evaluated Configuration

NICI 2.7.1 was evaluated in the following configuration

(1) A CC evaluated computing platform consisting of a generic hardware with the CC evaluated Operating System installed as specified by its evaluation.

   – The instructions for Solaris can be found in the document at (http://www.commoncriteriaportal.org/public/files/epfiles/TSolaris8_Issue3.1.pdf)

   – For SuSE Linux Enteprise Server 8, the document is (http://www.bsi.bund.de/zertifiz/zert/reporte/0234a.pdf)

   – For Microsoft Windows 2000 Server the instructions are (http://www.microsoft.com/technet/security/prodtech/windows2000/w2kccscg/default.mspx)

(2) NICI was installed using the standard installation program.

3.2 User Guidance

The following steps should be followed by the User:
– To ensure that NICI operates in an approved mode of operation, the user should not use non-FIPS approved algorithms.

Novell recommends the following instructions be followed to ensure smooth operation:

– The Crypto-Officer must ensure that the operating system is properly installed with the latest security patches.
– Once used NICI's keys should not be deleted
– Configuration files, license key and archive files should not be updated out-of-band. This makes NICI regenerate user keys, thus making NICI unusable.

APPENDIX A – CCS API Definitions

For complete descriptions, please refer to the *Controlled Cryptography Services 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.
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_SetNewIV	Sets a new IV
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#12.
CCS_pbeDecrypt	Decrypt data in a single part using a password and password-based algorithm as described in 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#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.
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.
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.
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.