LunaCA Security Policies

DOCUMENT CLASS:	**Overview**
CODE NAME:	**n/a**
SECURITY LEVEL:	**Unrestricted**
ORIGINATOR:	**Wayne A. Reed**
DEPARTMENT:	**Engineering**
DATE ORIGINATED:	**21 October 1998**
DOCUMENT NUMBER:	**802508**
VERSION:	**1.40**
PROJECT NO:	**5550-100**
PRINTED BY:	**Wayne Reed**
PRINTED ON:	**September 20, 2000 at 8:18 AM**

Select embedded fields and press F9 to update.

APPROVALS		
_____ Roger Hebb	Manager, Firmware Development	_____ Date
_____ Bruno Couillard	Chief Technical Officer	_____ Date
_____ Wayne Reed	V.P., Engineering	_____ Date

Revision History

Revision	Date	Description
1.00	21 August 1998	Created document from *Luna2 Security Policies*.
1.10	24 August 1998	Revised TPV key single function bit; added a statement on physical security.
1.20	21 September 1998	Modified copyright and security classification.
1.30	20 October 1998	Added algorithm matrix and FPV matrix.
1.40	21 October 1998	Added Luna command session and login state information.

Table of Contents

1. Introduction

1.1. Purpose

This document describes the security policies implemented by the LunaCA PC Card (*LunaCA token*) and how the design of its firmware enforces these policies.

1.2. Scope

This document addresses the LunaCA token's security policies.

1.3. Intended Audience

The intended audience for this document is: the LunaCA Engineering and Product Management Team, external agencies for validation or endorsement of the LunaCA token; selected industry partners; and potential users of LunaCA that want to understand the security policies of the product for FIPS operations.

The reader of this document should be familiar with the PKCS#11 standard [PKCS#11] defined by RSA Laboratories.

2. LunaCA Overview

The LunaCA token securely stores data and keying material inside its cryptographic boundary. The LunaCA token also performs cryptographic operations on data provided by external applications using the keying material stored in the token. These abilities are defined as key management, object management, and cryptographic capability.

Before a LunaCA token can be used to perform any cryptographic or key/object management functions, the token must receive a valid operator identity (also known as a user number), as well as valid authentication code (a PIN). These two inputs are processed by the token during a "LOGIN" command. When this command has completed successfully, the token allows the user to perform operations based on the policy settings defined for that token.

The token has the ability to distinguish two categories of users: super-users and normal users. The super-user category is referred to as the Security Officer (SO) and the normal user category is referred to as the user. A token can have only one SO. The SO is allowed to perform all of the cryptographic, key and object management functions provided by the token, as well as a set of functions called the SO functions. These SO functions are available only to the SO, and they allow the SO to manage the token policy.

For a LunaCA token, there is no limit on the number of users that can be created by the SO. All users are subjected to the same policy settings as those established by the SO. However, each user has its own authentication code (or PIN) initially assigned under control of the SO, which is used internally to protect the data the user owns.

The LunaCA token meets and is validated against FIPS 140-1 level 2 physical security requirements. For example, one aspect of physical security is through tamper evidence provided by the case: an attacker cannot get into the LunaCA token and access plaintext keys in an operational state. Contact Chrysalis-ITS for more details of the physical security used to protect the LunaCA token.

3. Security Policy Tools

The LunaCA token provides two levels of security policy enforcement. A vector that is loaded on the token during manufacturing dictates the first level of security. This vector, called the Fixed Policy Vector (FPV), establishes an envelope of security that cannot be modified after manufacturing.

The second level of security is provided by policy vector that can be modified by the token's SO. This vector is called the Token Policy Vector (TPV), and consists of a series of policy settings that can be established and modified by the token's legitimate SO.

3.1. Fixed Policy Vector (FPV)

The FPV contains the settings necessary to enforce policy rules that apply across a wide range of token users, regardless of their organizational policies. For example, one bit in the FPV defines whether the token can be exported. In an exportable version, the token provides a reduced set of algorithms and imposes limitations on maximum key lengths as required by export regulations.

The FPV cannot be modified by the SO or any of the users. The FPV is put on the token when it is manufactured and remains in place until the token is destroyed or the firmware is erased. The integrity of the FPV is maintained through the same mechanism used to protect the executable code from being modified. This mechanism is a 32-bit CRC computation.

The format of the FPV is a 32-bit vector that is divided into four fields of eight bits. These fields and their contents are defined in the following sections.

3.1.1. Number of SO Login Fails Allowed

This field defines the number of consecutive failed login attempts that can be made by the SO before the token erases the flash memory to prevent illegal access to its contents.

This security measure prevents an impostor from cracking the SO's password on the token.

3.1.2. Secret Key Policy

The following table defines the flags that identify the security policies that are followed for secret key objects.

Name	Description
FPV_SEC_KEY_POLICY_SENSITIVE	This bit determines whether a secret key object must always be made sensitive or if it can be determined by the high-level application using the token. When this bit is set, all secret keys stored on the token are sensitive. The keys are encrypted when in the flash memory and they can be extracted only outside of the token in encrypted form using the GESC_WRAP_KEY function. This bit must be set for FIPS-compliant tokens.
FPV_SEC_KEY_POLICY_NO_CREATE	This bit determines whether a secret key object can be created by an external application using the token, instead of being generated by the token. When this bit is set, an external application cannot create a secret key on the token; it is not possible to enter a secret key in plain text form on the token. This bit must be set for FIPS-compliant tokens.

3.1.3. Private Key Policy

The following table defines the flags that identify the security policies that are followed for private key objects.

Name	Description
FPV_PRI_KEY_POLICY_SENSITIVE	This bit determines whether a private key object must always be made sensitive or if it can be determined by the high-level application using the token through PKCS#11. When this bit is set, all private keys stored on the token must be flagged as sensitive whether or not the high-level application requested this flag when the keys were created. When this bit is set, all private keys are encrypted while stored in flash memory. Note: After a private key is sensitive, it cannot be extracted from the token even in encrypted format. This bit must be set for FIPS-compliant tokens.
FPV_PRI_KEY_POLICY_NO_CREATE	This bit determines whether a private key object can be created by an external application using the GESC_CREATE_OBJ call, instead of being generated by the token. When this bit is set, an external application cannot create a private key on the token; it is not possible to enter a private key in plain text form on the token. This bit must be set for FIPS-compliant tokens.

3.1.4. Token Security Policy

The following table defines the flags that identify the security policies that dictate the behavior of the token in general.

Name	Description
FPV_SECURITY_POLICY_DOMESTIC	This bit determines whether the token can be exported. When this bit is set, the token is configured for the domestic market and cannot be exported. This bit is verified internally every time a cryptographic function implying an encryption or a decryption is performed. If the bit is set, no restrictions exist on key sizes. If the bit is not set, a limitation of 56 bits is applied to any symmetric keys used for encryption or decryption, and a 512-bit limitation on asymmetric keys used for wrapping and unwrapping operations. Signature and verification operations are not restricted in terms of key lengths.
FPV_SECURITY_POLICY_SERVER	This bit indicates that the token is intended for use in a server environment. When this bit is set, server operations are enabled. For a LunaCA token, this bit is not set.
FPV_SECURITY_POLICY_USE_CAV	This bit is used by the firmware to determine whether the token was loaded with Cryptographic Algorithm Vector (CAV). When this bit is set, CAV is present. This setting is intended for tokens that were manufactured before CAV was created and that are being updated with new firmware.
FPV_SECURITY_POLICY_WRAPPING_TOKEN	This bit determines whether RSA private keys can be wrapped. When this bit is set, an RSA private key can be wrapped.
FPV_SECURITY_POLICY_USE_M_OF_N	This bit defines whether the token can perform M of N activation. When this bit is set, the token can be configured to perform M of N activation. M of N is a principal feature of a LunaCA token.
FPV_SECURITY_POLICY_USE_RAW_RSA	This bit determines whether RAW RSA operations can be performed on the token. When this bit is set, RAW RSA operations are allowed. RAW RSA provides access to RSA to perform encrypt and decrypt operations on data without any padding.
FPV_SECURITY_POLICY_SPECIAL_CLONING	This bit determines whether the token can participate in key cloning. When this bit is set, key cloning can be performed. Key cloning is a principal feature of a LunaCA token.
FPV_ENABLE_CCM	This bit determines whether a custom command module can be loaded onto the token. When this bit is set, a custom command module can be loaded onto the token. This bit must be cleared (i.e., zero) for FIPS-compliant tokens.
FPV_PIN_MUST_USE_SP	This bit determines if the serial communication port must be used to enter an authentication code. When this bit is set, an authentication code can only be entered through the serial communication port. When this bit is

Name	Description
	cleared, authentication codes are entered via the host computer. Use of the serial communication port is NOT a feature of a LunaCA token; this bit is clear on a LunaCA token.
FPV_MOFN_MUST_USE_SP	This bit determines if the serial communication port must be used to enter the M of N secret. When this bit is set, the M of N secret can only be entered through the serial communication port. When this bit is cleared, the M of N secret is entered via the host computer. Use of the serial communication port is NOT a feature of a LunaCA token; this bit is clear on a LunaCA token.
FPV_KCV_MUST_USE_SP	This bit determines if the serial communication port must be used to enter the key cloning domain identifier. When this bit is set, the key cloning domain identifier can only be entered through the serial communication port. When this bit is cleared, the key cloning domain identifier is entered via the host computer. Use of the serial communication port is NOT a feature of a LunaCA token; this bit is clear on a LunaCA token.

3.2. Cryptographic Algorithm Vector (CAV)

CAV contains a series of flags that identify which cryptographic algorithms are active on a token. One bit is defined for each algorithm that the token firmware can perform.

Name	Description
CA_CAV_DES_ENCRYPT	When set, the token can perform a DES encryption operation.
CA_CAV_DES_DECRYPT	When set, the token can perform a DES decryption operation.
CA_CAV_3DES_ENCRYPT	When set, the token can perform a triple-DES encryption operation.
CA_CAV_3DES_DECRYPT	When set, the token can perform a DES decryption operation.
CA_CAV_RC2_ENCRYPT	When set, the token can perform an RC2 encryption operation.
CA_CAV_RC2_DECRYPT	When set, the token can perform an RC2 decryption operation.
CA_CAV_RC4_ENCRYPT	When set, the token can perform an RC4 encryption operation.
CA_CAV_RC4_DECRYPT	When set, the token can perform an RC4 decryption operation.
CA_CAV_RC5_ENCRYPT	When set, the token can perform an RC5 encryption operation.
CA_CAV_RC5_DECRYPT	When set, the token can perform an RC5 decryption operation.
CA_CAV_CAST_ENCRYPT	When set, the token can perform a CAST encryption operation.
CA_CAV_CAST_DECRYPT	When set, the token can perform a CAST decryption operation.
CA_CAV_CAST3_ENCRYPT	When set, the token can perform a CAST3 encryption operation.
CA_CAV_CAST3_DECRYPT	When set, the token can perform a CAST3 decryption operation.
CA_CAV_CAST5_ENCRYPT	When set, the token can perform a CAST5 encryption operation.
CA_CAV_CAST5_DECRYPT	When set, the token can perform a CAST5 decryption operation.
CA_CAV_MD2	When set, the token can perform an MD2 operation.
CA_CAV_MD5	When set, the token can perform an MD5 operation.
CA_CAV_SHA_1	When set, the token can perform an SHA-1 operation.
CA_CAV_RSA	When set, the token can perform an RSA operation.
CA_CAV_DSA	When set, the token can perform a DSA operation.
CA_CAV_DH	When set, the token can perform a Diffie Hellman operation.

There are two fields in CAV that each consist of four bits. These fields represent the major and minor version of CAV.

Name	Description
CA_CAV_VERSION_MAJOR	These four bits represent the major version number of the CAV vector.
CA_CAV_VERSION_MINOR	These four bits represent the minor version of the CAV vector.

3.3. Token Policy Vector (TPV)

The TPV contains the settings necessary to enforce policy rules locally in an organization. For example, one bit in the TPV defines whether the token can perform a signature operation using a signing key generated by an outside process or if it must use an internally-generated key for this function. The TPV can be modified by the token's SO. The TPV contents are used by the internal code to validate the operations performed by the token's USER.

The format of the TPV is a 32-bit vector that is divided into four fields of eight bits. These fields and their contents are defined in the following sections.

3.3.1. Number of User Login Fails Allowed

This field defines the number of consecutive failed login attempts that can be made by a USER before the USER gets locked out or the USER's data is erase. This security feature prevents illegal access to the USER's data and keys: it prevents an impostor from cracking the USER's password on the token. Whether the user is locked out or the data is erased depends upon the "USER zeroize" bit. If the USER zeroize bit is disabled, too many failed login attempts results in the USER getting locked out. In this case, a USER must make a request to the SO to regain access to the token. The SO also provides a new password for the USER.

3.3.2. Minimum/Maximum PIN Length

These two fields define the minimum and maximum length restrictions for a USER's PIN.

3.3.3. Local Policies

The following table defines the flags that identify the security policies that dictate the behavior of the users on the token.

Name	Description
TPV_USER_ZEROIZE	This bit determines whether the token can be zeroized by a normal user or if only the SO can zeroize the token. This bit indicates whether the token is centrally controlled. When this bit is set, it indicates that a valid token user can zeroize the token. This bit enables using the token in an environment where the SO is not commonly used. When this bit is set, the SO cannot change a user password, and a user is zeroized after too many unsuccessful login attempts.
TPV_USER_FW_UPDATE	This bit determines whether the firmware can be updated by a normal user or if only the token's SO can update the firmware. When this bit is set, a normal user can perform the firmware update.
TPV_M_OF_N_ACTIVATION	This bit determines whether M of N activation is required for a user to gain access to the token. When this bit and the FPV_SECURITY_POLICY_USE_M_OF_N bit in the FPV is set, the token is not activated until the required number of parts to a split secret have been entered.
TPV_KEY_ATTRIB_LOCK	This bit determines whether the flag attributes of a key can be modified once the key is a valid object on the token. When this bit is set, it indicates that the flag attributes of a key cannot be modified after they have been established. For example, if this bit is set and a DES key is created for encryption and decryption, these attributes cannot be changed to wrap and unwrap once the key exists on the token.
TPV_KEY_SINGLE_FUNCTION	This bit determines whether a key can be used to perform multiple types of operations (i.e., use a key for encrypting, signing, and wrapping). When this bit is set, it indicates that keys can be used only to perform single functions. For symmetric keys, a single function is considered to be a pair of related functions such as encryption/decryption, wrapping/unwrapping, or sign/verify. With the validated release of LunaCA, multiple use of a key is allowed regardless of the value of

Name	Description
	TPV_KEY_SINGLE_FUNCTION.
TPV_SIGNING_KEY_LOCAL	When performing a signing operation, the private key used may have been generated locally or provided by an external source. In most environments, it is preferable to have the signing/verifying key pair generated by the token and never extracted from it. However, in certain cases the signing keys are generated externally and loaded on the token for subsequent signature operations. When this bit is set, it indicates that externally generated keys cannot be used for signing operations performed by the token.
TPV_MOFN_MUST_USE_SP	This bit determines whether the M of N secret must be entered through the serial communication port. When this bit and the FPV_MOFN_MAY_USE_SP bit in the FPV is set, the token must use the serial communication port to enter the M of N secret. If the FPV_MOFN_MAY_USE_SP bit is not set, the TPV_MOFN_MUST_USE_SP bit is ignored and the M of N secret must be entered through the host computer. Ordinarily, this bit is not set for LunaCA tokens.
TPV_KCV_MUST_USE_SP	This bit determines whether the key cloning domain identifier must be entered through the serial communication port. When this bit and the FPV_KCV_MAY_USE_SP bit in the FPV is set, the token must use the serial communication port to enter the key cloning domain identifier. If the FPV_KCV_MAY_USE_SP bit is not set, the TPV_KCV_MUST_USE_SP bit is ignored and the key cloning domain identifier must be entered through the host computer. Ordinarily, this bit is not set for LunaCA tokens.

4. Identification and Authentication (I&A)

The LunaCA token enforces an identity-based user authentication policy. For normal users, the user number and a valid PIN must be provided to the token before access to private data and token services can be granted. For the SO, only a PIN is required.

Note: Normal users also have a text-based name associated with them. The name corresponding to a particular user number can be queried from the token.

The PINs for the SO and users can be changed at any time by their respective owners. The SO can also re-instate users with lost PINs. Re-instating users does not affect the cryptographic material and data owned by the user and protected under the old PIN.

The LunaCA token implements policy that limits the number of login attempts. This feature prevents an exhaustive search approach for finding the PIN of the SO or user. The implementation of this feature differs from that of an SO PIN search and that of a user PIN search.

For a user PIN search:

- If “*n”* consecutive user logon attempts fail, the token flags the event in the user’s authorization vector (UAV). This locks out the user until the SO logs back on to the token and unlocks the user. (The value of "n" is defined by the SO in the TPV.)

For an SO PIN search:

- If “*n*” consecutive SO logon attempts fail, the token is zeroized and its operational state goes to ZEROIZED. (The value of "n" is defined in the FPV, which is defined when a LunaCA token is manufactured and cannot be modified without invalidating the CRC value of the software load.)

5. Discretionary Access Control (DAC)

Every data object on the token can be public or private. Private data objects are labeled with a number that corresponds to the owner and can be accessed only by the legitimate owner. A user cannot create a key or certificate object as a public object. Only data objects can be public or private.

The token does not allow for any granularity of ownership other than that of individual or public (i.e., a data object cannot be owned by two users and restricted from other users). Also, the ownership of an object implies read/write/modify/execute access to the object, which means full access rights to the object.

6. M of N Activation

LunaCA supports a token activation feature called *M of N*. The concept of the M of N activation capability provides protection of a secret by "splitting" it into "N" pieces, where any "M" of these pieces must be reassembled to reconstruct the original secret. The LunaCA feature is based on Shamir's threshold scheme. This scheme allows a secret value to be shared by "n" external recipients without risking any compromise to the secret.

M of N activation must be performed when there are no permanent sensitive objects stored on the token. Otherwise, there is a risk of corrupting objects stored in flash prior to generating the M of N set or after "deactivating" the feature. Management of M of N activation is an operational issue for the SO.

Note that for M of N activation to be effective, the values of M and N should be two or greater. Consider, for example, a 1 of 1 share to be of little value in securing activation of a token.

7. Object Reuse

The token enforces an object reuse policy in that every object is allocated a portion of memory (flash or SRAM). The policy also ensures that no other objects are placed in the same memory location unless all previous memory content are initialized and purged. When cryptographic functions are performed, a cryptographic context is created to hold data required by the function (e.g., a DES key schedule for a DES function or an MD5 chaining vector). The cryptographic context only exists in SRAM memory and is not assigned to any functions except those defined by its owner. The memory assigned to a cryptographic context is always purged of its content before being handed over to a function.

8. References

[PKCS#11] *PKCS #11: Cryptographic Token Interface Standard*, RSA Laboratories, Draft 1, version 2.01, September 12,1997.

APPENDIX A. Cryptographic Algorithms Support

Encrypt/Decrypt:

- DES-ECB
- DES-CBC
- 3-DES-ECB
- 3-DES-CBC
- RC2-ECB
- RC2-CBC
- RC4
- RC5-ECB
- RC5-CBC
- CAST-ECB
- CAST-CBC
- CAST3-ECB
- CAST3-CBC
- CAST5-ECB
- CAST5-CBC
- RSA X-509

Digest:

- MD2
- MD5
- SHA-1

Sign/Verify:

- RSA -1024
- RSA -2048
- DSA
- DES-MAC
- 3-DES-MAC
- RC2-MAC
- RC5-MAC
- CAST-MAC
- CAST3-MAC
- CAST5-MAC
- SSL3-MD5-MAC
- SSL3-SHA1-MAC
- HMAC-SHA1
- HMAC-MD5

Generate Key:

- DES
- double length DES
- triple length DES
- RC2
- RC4
- RC5
- CAST
- CAST3
- CAST5
- PBE-MD2-DES
- PBE-MD5-DES
- PBE-MD5-CAST
- PBE-MD5-CAST3
- PBE-SHA-1-CAST5
- GENERIC-SECRET
- SSL PRE-MASTER

Generate Key Pair:

- RSA-1024
- RSA-2048
- DSA-1024
- DH-1024

Wrap Symmetric Key Using Symmetric Algorithm:

- DES-ECB
- 3-DES-ECB
- RC2-ECB
- CAST-ECB
- CAST3-ECB
- CAST5-ECB

Wrap Symmetric Key Using Asymmetric Algorithm:

- RSA-1024
- RSA-2048

Wrap Asymmetric Key Using Symmetric Algorithm:

- 3-DES-CBC[1]

Unwrap Symmetric Key With Symmetric Algorithm:

- DES-ECB
- 3-DES-ECB
- RC2-ECB
- CAST-ECB
- CAST3-ECB
- CAST5-ECB

Unwrap Symmetric Key With Asymmetric Algorithm:

- RSA-1024
- RSA-2048

Unwrap Asymmetric Key With Symmetric Algorithm:

- DES-CBC
- 3-DES-CBC
- CAST-CBC
- CAST3-CBC
- CAST5-CBC

Derive Key Value:

- DH-1024
- concatenate Base & Key
- concatenate Base & Data
- concatenate Data & Base
- XOR Base and Data
- Extract Key from Key
- MD2 Derivation
- MD5 Derivation
- SHA-1 Derivation
- SSL3-Master
- SSL3-Key & MAC

APPENDIX B. Fixed Policy Vector Settings

	Standard LunaCA Domestic	Standard LunaCA Export
Token Policy Vector Settings		
TPV_USER_ZEROIZE	1	1
TPV_USER_FW_UPDATE	0	0
TPV_M_OF_N_ACTIVATION	0	0
TPV_KEY_ATTRIB_LOCK	1	1
TPV_KEY_SINGLE_FUNCTION	0	0
TPV_SIGNING_KEY_LOCAL	1	1
TPV_MAX_PIN_LEN	48	48
TPV_MIN_PIN_LEN	4	4
TPV_LOGIN_FAILS_ALLOWED	10	10
Fixed Policy Vector Settings		
FPV_SECURITY_POLICY_DOMESTIC	1	0
FPV_SECURITY_POLICY_SERVER	1	1
FPV_SECURITY_POLICY_USE_CAV	0	0
FPV_SECURITY_POLICY_WRAPPING_TOKEN	0	0
FPV_SECURITY_POLICY_USE_M_OF_N	1	1
FPV_SECURITY_POLICY_USE_RAW_RSA	1	1
FPV_SECURITY_POLICY_SPECIAL_CLONING	1	1
FPV_ENABLE_CCM	0	0
FPV_SEC_KEY_POLICY_SENSITIVE	1	1
FPV_SEC_KEY_POLICY_NO_CREATE	1	1
FPV_PRI_KEY_POLICY_SENSITIVE	1	1
FPV_PRI_KEY_POLICY_NO_CREATE	1	1
FPV_SO_LOGIN_FAILS_ALLOWED	3	3
FPV_PIN_MUST_USE_SP	0	0
FPV_MOFN_MUST_USE_SP	0	0
FPV_KCV_MUST_USE_SP	0	0

	Standard LunaCA Domestic	Standard LunaCA Export
Cryptographic Algorithm Vector Settings		
CA_CAV_VERSION_MAJOR	15	15
CA_CAV_VERSION_MINOR	15	15
CA_CAV_DES_ENCRYPT	1	1
CA_CAV_DES_DECRYPT	1	1
CA_CAV_3DES_ENCRYPT	1	1
CA_CAV_3DES_DECRYPT	1	1
CA_CAV_RC2_ENCRYPT	1	1
CA_CAV_RC2_DECRYPT	1	1
CA_CAV_RC4_ENCRYPT	1	1
CA_CAV_RC4_DECRYPT	1	1
CA_CAV_RC5_ENCRYPT	1	1
CA_CAV_RC5_DECRYPT	1	1
CA_CAV_CAST_ENCRYPT	1	1
CA_CAV_CAST_DECRYPT	1	1
CA_CAV_CAST3_ENCRYPT	1	1
CA_CAV_CAST3_DECRYPT	1	1
CA_CAV_CAST5_ENCRYPT	1	1
CA_CAV_CAST5_DECRYPT	1	1
CA_CAV_MD2	1	1
CA_CAV_MD5	1	1
CA_CAV_SHA_1	1	1
CA_CAV_RSA	1	1
CA_CAV_DSA	1	1
CA_CAV_DH	1	1

APPENDIX C. Session And Login States Required For Luna Token Commands

Command To Token	No Session Open	Session Open, No Login	SO Logged On	User Logged On
Token Main Module Commands				
LUNA_ZEROIZE	√			
LUNA_INIT_TOKEN			√	
LUNA_GET	√			
LUNA_GET_USV			√	
LUNA_SET_TPV			√	
LUNA_FW_UPDATE			√	
Session Manager Commands				
LUNA_OPEN_ACCESS	√			
LUNA_CLEAN_ACCESS	√			
LUNA_CLOSE_ACCESS	√			
LUNA_OPEN_SESSION	√			
LUNA_CLOSE_SESSION		√		
LUNA_CLOSE_ALL_SESSIONS	√			
LUNA_GET_SESSION_INFO		√		
LUNA_EXTRACT_CONTEXTS		√		
LUNA_INSERT_CONTEXTS		√		
User Module Commands				
LUNA_GET_USER_LIST		√		
LUNA_GET_USER_NAME		√		
LUNA_LOGIN		√		
LUNA_LOGOUT				√
LUNA_SET_PIN				√
LUNA_INIT_PIN			√	
LUNA_CREATE_USER			√	
LUNA_DELETE_USER			√	
Object Management Module				
LUNA_CREATE_OBJECT		√		
LUNA_COPY_OBJECT		√		
LUNA_DESTROY_OBJECT		√		
LUNA_GET_OBJECT_SIZE		√		
LUNA_GET_ATTRIBUTE_VALUE		√		
LUNA_GET_ATTRIBUTE_LENGTH		√		
LUNA_MODIFY_OBJECT		√		
LUNA_FIND_OBJECTS		√		
Random Number Generator Module				
LUNA_GET_RANDOM		√		
LUNA_SEED_RANDOM		√		
Key Management Module				
LUNA_GENERATE_KEY				√
LUNA_GENERATE_KEY_W_VALUE				√
LUNA_GENERATE_KEY_PAIR				√
LUNA_WRAP_KEY				√
LUNA_UNWRAP_KEY				√
LUNA_UNWRAP_KEY_W_VALUE				√
LUNA_DERIVE_KEY				√
LUNA_DERIVE_KEY_W_VALUE				√
LUNA_MFG_LOAD				√
Cryptographic Algorithm Module				

Command To Token	No Session Open	Session Open, No Login	SO Logged On	User Logged On
LUNA_ENCRYPT_INIT				√
LUNA_ENCRYPT_INIT_W_VALUE				√
LUNA_ENCRYPT				√
LUNA_ENCRYPT_FIFO				√
LUNA_ENCRYPT_END				√
LUNA_DECRYPT_INIT				√
LUNA_DECRYPT_INIT_W_VALUE				√
LUNA_DECRYPT				√
LUNA_DECRYPT_FIFO				√
LUNA_DECRYPT_END				√
LUNA_DECRYPT_RAW_RSA				√
LUNA_DIGEST_INIT		√		
LUNA_DIGEST		√		
LUNA_DIGEST_FIFO		√		
LUNA_DIGEST_KEY				√
LUNA_DIGEST_KEY_VALUE				√
LUNA_DIGEST_END		√		
LUNA_SIGN_INIT				√
LUNA_SIGN_INIT_W_VALUE				√
LUNA_SIGN				√
LUNA_SIGN_FIFO				√
LUNA_SIGN_END				√
LUNA_SIGN_UPDATE_KEY				√
LUNA_SIGN_FINAL_DERIVE_KEY				√
LUNA_VERIFY_INIT				√
LUNA_VERIFY_INIT_W_VALUE				√
LUNA_VERIFY				√
LUNA_VERIFY_FIFO				√
LUNA_VERIFY_END				√
LUNA_GET_MECH_LIST	√			
LUNA_GET_MECH_INFO	√			
LUNA_SELF_TEST	√			
LUNA_SET_UP_MASKING_KEY	√			
LUNA_CLONE_AS_SOURCE				√
LUNA_CLONE_AS_TARGET_INIT				√
LUNA_CLONE_AS_TARGET				√
LUNA_GEN_TKN_KEYS			√	
LUNA_LOAD_CERT			√	
LUNA_GEN_KCV				√
LUNA_LOAD_CUSTOMER_VERIFICATION_KEY			√	
LUNA_M_OF_N_GENERATE			√	
LUNA_M_OF_N_ACTIVATE				√
LUNA_M_OF_N_MODIFY			√	
LUNA_ISAKMP_INIT				√
LUNA_ISAKMP_PROCESS_PACKET				√
LUNA_ISAKMP_END				√
Custom Command Module				
LUNA_LOAD_CUSTOM_MODULE			√	
LUNA_LOAD_ENCRYPTED_CUSTOM_MODULE			√	
LUNA_UNLOAD_CUSTOM_MODULE			√	
LUNA_EXECUTE_CUSTOM_COMMAND				√
LUNA_GET_CUSTOM_MODULE_LIST	√			
LUNA_GET_CUSTOM_MODULE_INFO	√			