Security First Corp.
Securing Your Digital World®

SecureParser®
Cryptographic Module

Security Policy

Versions 4.5.0 & 4.5.1
Revision 1.05

31 January 2008

Revision History

Revision History			
Version	***Date***	***Author***	***Notes***
0.01	01/11/2007	Infogard, Security First Corp.	Documentation Workshop (Infogard template)
0.02	02/26/2007	Security First Corp.	Accumulated changes to date after Documentation Workshop.
0.03	03/20/2007	Security First Corp.	Added key sizes for DSA & RSA keys in Section 3 Modes of Operation
0.04	03/22/2007	Security First Corp.	Added power-on self-test RSA encrypt/decrypt
0.05	04/20/2007	Security First Corp.	Corrected Security Rule 25 to reflect PRNG is based on AES Encrypt, not AES Decrypt
0.06	05/22/2007	Security First Corp.	Clarified that the same HMAC key is used for Data & Share authentication/integrity
0.07	06/07/2007	Security First Corp.	Added Algorithm certificate numbers
0.08	06/12/2007	Security First Corp.	Added entropy assessment details
0.09	07/11/2007	Security First Corp.	Updates after Operational Testing. ECDSA added. Overview updated.
1.00	08/07/2007	Security First Corp.	Final edit before submission
1.01	11/5/2007	Security First Corp. (ISE input)	V4.5.1 revision for submission. Updated API section, removed references to single-threaded requirement.
1.02	12/16/2007	Security First Corp.	Title page updated.

		(ISE input)	
1.03	01/07/2008	Security First Corp. (ISE input)	Responses to CMVP Comments. Additional V4.5.1 changes for clarity regarding Multi-threading and Kernel mode.
1.04	01/18/2008	Security First Corp. (ISE input)	Responses to CMVP Comments round 2. All references to MS RSAENH.dll removed. Standard platform services are providing entropy. Security Rule 24:3 corrected.
1.05	01/31/2008	Security First Corp. (ISE input)	**Responses to CMVP Comments round 3.** Clarification: Key entry/output is always encrypted. PRNG_Seed_Value rationale of strength modified as per CMVP suggestion.

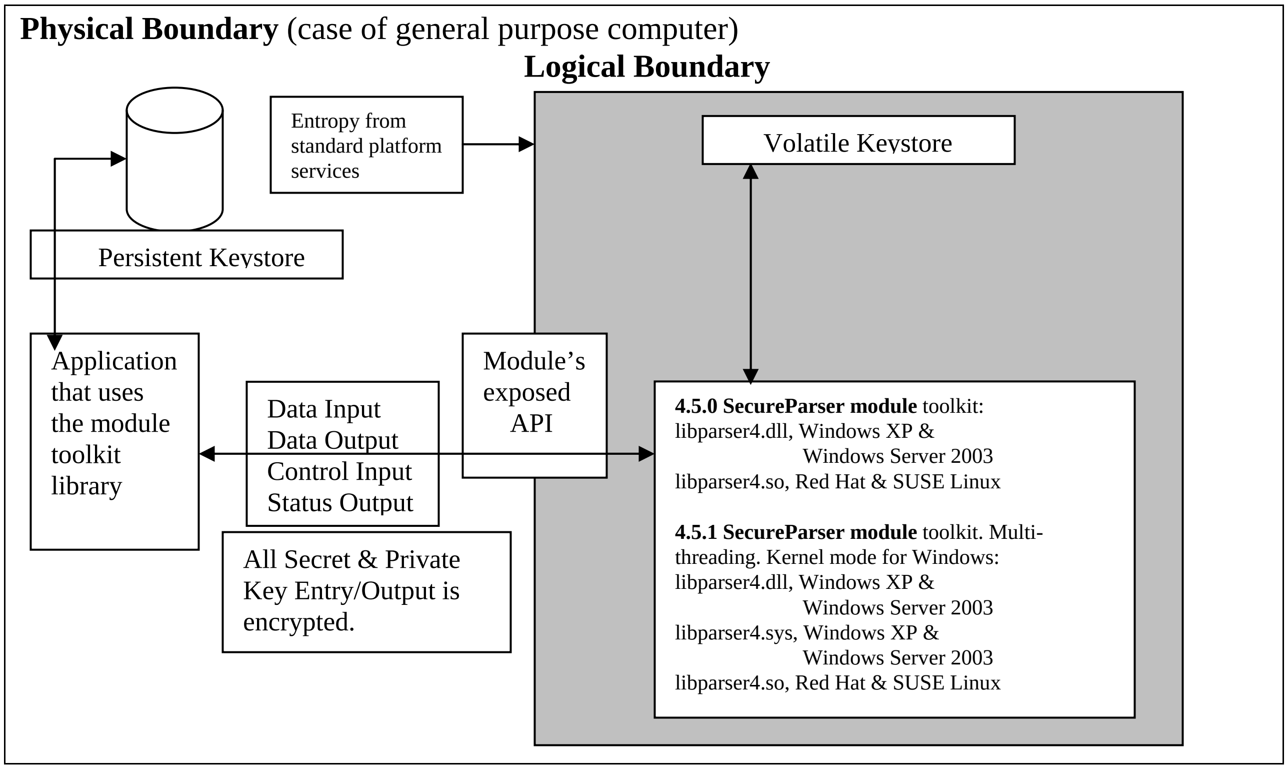
TABLE OF CONTENTS

1. Module Overview

The SecureParser® Encryption module is a software only module that is installed on a multi-chip standalone device such as a General Purpose Computer (GPC). The SecureParser module is a security and high data availability architecture delivered in the form of a toolkit that provides cryptographic data splitting (data encryption, random or deterministic distribution to multiple shares including additional fault tolerant bits, key splitting, authentication, integrity, share reassembly, key restoration and decryption) of arbitrary data. The SecureParser accepts any type of digital data and cryptographically splits it into shares so that no discernible data is transmitted across a network or lands on a single storage device. During the split process, additional redundant data may be optionally written to each share enabling the capability of restoring the original data when all shares are not available. Transmission of smaller shares improves performance over multiple network paths or I/O channels. The shares can be stored in geographically disbursed nodes providing for continuous access to online information. Each share contains an integrity check that prevents tampering with the stored data and is immediately recognized by the other shares. Any change to the data in a share would preclude that share from being used in the data rebuild process. The encryption keys are encrypted with a Workgroup key, split and stored with the data which effectively eliminates file level key management. Data availability and survivability will be greatly improved and allow for a return to operations (RTO) of zero loss time when faced with environmental, hostile or accidental catastrophes. The SecureParser module is designed to be integrated at any point where data is written, retrieved, sent or received.

Boundaries

Figure 1 – Image of the Cryptographic Module


Physical Boundary (case of general purpose computer)
Logical Boundary
Entropy from standard platform services
Volatile Keystore
Persistent Keystore
Application that uses the module toolkit library
Data Input
Data Output
Control Input
Status Output
Module's exposed API
All Secret & Private Key Entry/Output is encrypted.
4.5.0 SecureParser module toolkit:
libparser4.dll, Windows XP & Windows Server 2003
libparser4.so, Red Hat & SUSE Linux
4.5.1 SecureParser module toolkit. Multi-threading. Kernel mode for Windows:
libparser4.dll, Windows XP & Windows Server 2003
libparser4.sys, Windows XP & Windows Server 2003
libparser4.so, Red Hat & SUSE Linux

Logical Boundary v4.5.0

When operating on Microsoft Windows operating system platforms Windows XP and Windows Server 2003, the SecureParser module cryptographic logical boundary is defined as containing a single executable, the SecureParser libparser4.dll. Seed values for the SecureParser's random number generator will be imported from standard operating system services within the physical boundary of the general purpose computer.

When operating on Linux operating system platforms Red Hat Enterprise v4, and SUSE Enterprise v10, the SecureParser module cryptographic logical boundary is defined as containing a single executable, the SecureParser libparser4.so. Seed values for the SecureParser's random number generator will be imported from standard operating system services within the physical boundary of the general purpose computer.

Logical Boundary v4.5.1

Version 4.5.1 adds Multi-threading capabilities on all the tested operating systems, and the additional option to execute the module in Kernel mode on Windows XP and Windows Server 2003.

When operating on Microsoft Windows operating system platforms Windows XP and Windows Server 2003, the SecureParser module cryptographic logical boundary is defined as containing a single executable, either the SecureParser libparser4.dll or the SecureParser libparser4.sys (libparser4.sys operates in Kernel mode). Seed values for the SecureParser's random number generator will be imported from standard operating system services within the physical boundary of the general purpose computer.

When operating on Linux operating system platforms Red Hat Enterprise v4, and SUSE Enterprise v10, the SecureParser module cryptographic logical boundary is defined as containing a single executable, the SecureParser libparser4.so. Seed values for the SecureParser's random number generator will be imported from standard operating system services within the physical boundary of the general purpose computer.

Physical Boundary

The SecureParser cryptographic physical boundary is the case of the General Purpose Computer (GPC) on which the libparser4 executable is instantiated. Ports at the physical boundary of the GPC are those typical of a GPC for connecting external devices such as keyboards, monitors, mice, and printers. These devices are outside the physical boundary of the cryptographic module and are excluded from the validation.

Operating Systems & Platforms

The SecureParser module has been tested on and found to be conformant with the requirements of FIPS 140-2 overall level 1 on the following GPC operating systems: Windows XP; Windows Server 2003; Red Hat Linux Enterprise v4; SUSE Linux Enterprise v10.

Operational testing was performed on all the above operating systems on a Dell Optiplex 210L Model DCSM.

Additionally, the module runs without recompilation on other GPC's equipped with x86 compatible processors running Microsoft Windows 2000, and Microsoft Windows 2000 Server. The module also runs without recompilation on other GPC's equipped with x86 compatible processors running kernels compatible with RedHat Linux Enterprise v4 and SUSE Linux Enterprise v10.

2. Security Level

The cryptographic module meets the overall requirements applicable to Level 1 security of FIPS 140-2.

Table 1 - Module Security Level Specification

Security Requirements Section	Level
Cryptographic Module Specification	3
Module Ports and Interfaces	1
Roles, Services and Authentication	1
Finite State Model	1
Physical Security	N/A
Operational Environment	1
Cryptographic Key Management	3
EMI/EMC	3

Self-Tests	1
Design Assurance	3
Mitigation of Other Attacks	N/A

3. Modes of Operation

Approved Algorithms

In FIPS mode, the SecureParser module supports FIPS Approved algorithms as follows. The certificate #'s sited below have all been obtained by SecureParser module algorithm testing with the CAVP.

Version 4.5.0:

▪ AES-CBC/ECB - 128/192/256 bit key	Cert. #594
▪ AES-CTR - 128/192/256 bit key	Cert. #594
▪ HMAC-SHA1 & HMAC-SHA256	Cert. #302
▪ SHA-1 SHA-256 hashing	Cert. #631
▪ DSA sign/verify – 1024 bit key	Cert. #229
▪ RSA sign/verify – 1024/2048/4096 bit key	Cert. #262
▪ PRNG Key Generation ANSI X9.31 with AES	Cert. #330
▪ ECDSA sign/verify – 521 bit key	Cert. #63

Version 4.5.1:

▪ AES-CBC/ECB - 128/192/256 bit key	Cert. #687
▪ AES-CTR - 128/192/256 bit key	Cert. #687
▪ HMAC-SHA1 & HMAC-SHA256	Cert. #366
▪ SHA-1 SHA-256 hashing	Cert. #716
▪ DSA sign/verify – 1024 bit key	Cert. #260
▪ RSA sign/verify NIST Key Wrapping– 1024/2048/4096 bit key	Cert. #321
▪ PRNG Key Generation ANSI X9.31 with AES	Cert. #401
▪ ECDSA sign/verify – 521 bit key	Cert. #77

Key Entry and Output

All Key Entry and Output in FIPS mode must be in encrypted form. Plaintext keys are never entered or output from the module.

NIST Key Wrapping per FIPS 140-2 Annex D using 128/192/256 bit keys.

RSA Key Wrapping per FIPS 140-2 IG 7.1 Acceptable Key Establishment Protocols, Key Transport using asymmetric keys [key wrapping] using k = 4096. (RSA key wrapping, key establishment methodology provides 128 bits of encryption strength).

Configuring the module for FIPS mode

The SecureParser module may be configured for FIPS mode by calling the module's exposed module_initialize( ) API function with the calling parameter "fipsEnabled" set to true. Subsequent SecureParser module API calls that are used to further configure the SecureParser will have their calling parameters checked by the SecureParser based on the value of the "fipsEnabled" calling parameter used in the original module_initialize( ) API function call. These subsequent checks are used to insure that all FIPS mode configuration values are set properly.

Operators can determine if the cryptographic module is running in FIPS versus non-FIPS mode via execution of the module's module_getstatus( ) API function call, which is used to meet the FIPS area 1 requirements to achieve level 3. The module_getstatus( ) API function call equates to the FIPS "show status" service and will indicate if a FIPS mode of operation has been selected. The module_getstatus( ) API call returns two items. (These items are returned as known/documented values to memory locations passed into module_getstatus( ) as pointers):

- Whether the module is in FIPS mode (value set = 1), or non-FIPS mode (value set = 0)
- The current FSM state of the module (MODULE_STATE enum)

Once a FIPS mode of operation has been selected the module cannot transition into a non-FIPS mode of operation during the lifetime of the module instantiation in executable memory. Similarly once a non-FIPS mode of operation has been selected the module cannot transition into a FIPS mode of operation during the lifetime of the module instantiation in executable memory.

Non-FIPS mode of operation

The SecureParser module can be initialized into a non-FIPS Approved mode of operation by setting the "fipsEnabled" flag to 0 during the first call to the API function module_initialize. Additionally in v4.5.0 external applications (outside of the logical boundary of the SecureParser module) are not allowed multi-threaded use of the SecureParser module toolkit library in FIPS mode. The v4.5.0 restriction against multi-threaded use is stated in documentation for external application developers.

Applications cannot transition their use of the module toolkit library to/from FIPS mode and non-FIPS mode while the module is instantiated. The module must be shut down by the calling application and then restarted to transition to/from FIPS mode and non-FIPS mode.

4. Identification and Authentication Policy

Assumption of roles

The SecureParser module has two distinct operator roles - governed by a single operator (the operating system): Cryptographic-Officer role; User role. Operators of the cryptographic module implicitly assume roles each time they call into the SecureParser module via exposed SecureParser API function calls. Each API call into the SecureParser module performs a module service. Consistent with FIPS 140-2 area 3 level 1 requirements, the operators of the SecureParser cryptographic module are not authenticated to the module. Note that the operating systems the module is run on provide functionality to require an operator to be successfully authenticated prior to using any service provided by the module.

Table 2 - Roles and Required Identification and Authentication

Role	Type of Authentication	Authentication Data
Cryptographic-Officer	None	None
User	None	None

Table 3 – Strengths of Authentication Mechanisms

Authentication Mechanism	Strength of Mechanism
None	None

5. Access Control Policy

Roles and Services

Table 4 – Services Authorized for Roles

Role	Authorized Services
Cryptographic-Officer: The Cryptographic Officer role is assumed when applications call the module's exposed API functions that perform initialization, configuration, and administrative services.	• **module_initialize**. Initializes the module, sets FIPS mode or non-FIPS mode, performs self tests, and moves the module into an operational state. • **parser_create**. Allocates the memory for a Parser structure. There can be multiple parser instances within the module – they are all either in FIPS mode, or they are all in non-FIPS mode (determined by the **module_initialize** service). • **parser_destroy**. Deallocates the memory of a Parser structure. • **parser_generateHeaders.** Configures parser context and generates headers. • **parser_restoreHeaders.** Configures parser context based on headers with optional modifications. • **keystore_getImportKey:** This service provides the RSA public key needed for asymmetric key wrapping for all key entry into the specified keystore of the module (note that each keystore will have its own public/private keypair). • **keystore_create.** Allocates memory for a volatile KeyStore structure, and creates a non-persistent RSA public/private

Role	Authorized Services
	encryption keypair to be used for key import (note that each keystore will have its own public/private keypair). • **keystore_destroy.** Deallocates the memory of a volatile KeyStore structure. • **keystore_addKeyFromBuffer.** Imports a key into the specified volatile keystore structure. All imported keys will be RSA key wrapped and will need to be unwrapped by the module. Note: x509 certificates can be in the buffer, their public keys will be imported. • **keystore_removeKey.** Removes a key from the volatile keystore. • **keystore_getKeyType.** Returns the key type for the requested key. • **keystore_getkeylength.** Returns the key length for the requested key. • **keystore_keyexists.** True or False, the requested key exists or does not exist within the specified volatile keystore. • **module_getstatus**: This service provides the current status of the cryptographic module including whether or not a FIPS Approved mode of operation has been selected. • **module_destroy:** Zeroization**,** called by the application prior to (graceful) application termination. Zeroizes non-persistent CSPs including the RSA import public/private keypair, and the volatile Keystores. ALL keystores and ALL parsers in memory are zeroized. • **Self tests**: Power cycle
User: The User role is assumed when applications call the module's exposed API functions that perform general cryptographic services.	• **parser_parseData.** Parses data from the input buffer gather list into the output buffers. • **parser_restoreData.** Restores data from the output buffers into the input buffers. • **parser_getHeaderInfo.** Processes the header and returns information about specific header fields. • **parser_getParsedLength.** Returns the number of bytes needed for each output share when parsing. • **parser_getRestoredLength.** Returns the number of bytes needed for the original share when restoring. • **module_getstatus**: This service provides the current status of the cryptographic module including whether or not a FIPS Approved mode of operation has been selected. • **Self tests**: Power cycle

Table 5 - Specification of Service Inputs & Outputs

Service	Control Input	Data Input	Data Output	Status Output
module_initialize	int fipsEnabled	N/A	N/A	Success or ERROR_TYPE
parser_create	Parser **ret	N/A	Parser **ret	Success or ERROR_TYPE
parser_destroy	Parser *p	N/A	N/A	Success or ERROR_TYPE
parser_ generateHeaders	Parser *p KeyStore *ks int L int M int N IDA_TYPE idaMode ENC_TYPE encMode HASH_TYPE hashMode char *workgroupKeyId size_t workgroupKeyIdMem size_t workgroupKeyIdSize, AUTH_TYPE postAuthMode char *postAuthKeyId size_t postAuthKeyIdMem size_t postAuthKeyIdSize uint8 **outputBuffers size_t *outputBufferLengths size_t *outputBufferMems	N/A	uint8 **outputBuffers size_t *outputBufferLengths	Success or ERROR_TYPE
parser_restoreHeaders	Parser *p KeyStore *ks HASH_TYPE hashMode char *workgroupKeyId size_t workgroupKeyIdMem size_t workgroupKeyIdSize AUTH_TYPE postAuthMode HASH_TYPE postHashMode char *postAuthKeyId size_t postAuthKeyIdMem size_t postAuthKeyIdSize uint8 **inputBuffers size_t * inputBufferLengths size_t * inputBufferMems int inputBuffersCount int trustedShareNumber	uint8 **inputBuffers	N/A	Success or ERROR_TYPE
keystore_ getImportKey	KeyStore *ks size_t bufferMem size_t *bufferLength	N/A	uint8 *buffer size_t *bufferLength	Success or ERROR_TYPE
keystore_create	KeyStore **ret int minimumKeyCount	N/A	KeyStore **ret	Success or ERROR_TYPE
keystore_destroy	KeyStore *ks	N/A	N/A	Success or ERROR_TYPE
keystore_	KeyStore *ks	uint8* buffer	N/A	Success or

Service	Control Input	Data Input	Data Output	Status Output
addKeyFromBuffer	uint8 *buffer size_t bufferMem size_t bufferLength char *id size_t idMem size_t idLength char *passphrase size_t passphraseMem size_t passphraseLength IMPORT_TYPE importType			ERROR_TYPE
keystore_removeKey	KeyStore *ks char *id size_t idMem size_t idLength	N/A	N/A	Success or ERROR_TYPE
keystore_getKeyType	KeyStore *ks char *id size_t idMem size_t idLength KEY_TYPE *keyType	N/A	KEY_TYPE *keyType	Success or ERROR_TYPE
keystore_ getKeyLength	KeyStore *ks char *id size_t idMem size_t idLength size_t *keyLength	N/A	size_t *keyLength	Success or ERROR_TYPE
keystore_keyExists	KeyStore *ks char *id size_t idMem size_t idLength int *keyExists	N/A	int *keyExists	Success or ERROR_TYPE
parser_parseData	Parser *p uint8*inputBuffer size_t inputBufferLength size_t inputBufferMem uint8 **outputBuffers size_t *outputBufferLengths size_t *outputBufferMems int outputBuffersCount	uint8 *inputBuffer	uint8**outputBuffers size_t *outputBufferLengths	Success or ERROR_TYPE
parser_restoreData	Parser *p uint8 *outputBuffer size_t *outputBufferLength size_t outputBufferMem uint8 **inputBuffers size_t *inputBufferLengths size_t *inputBufferMems int inputBuffersCount int trustedShareNumber	uint8 **inputBuffers	uint8 *outputBuffer size_t *outputBufferLength	Success or ERROR_TYPE
parser_getHeaderInfo	DATAFIELD_TYPE t uint8 *header size_t headerMem size_t headerLength uint8 *ret size_t retMem	uint8 *headerbuffer	uint8 *ret size_t *retLength	Success or ERROR_TYPE

Service	**Control Input**	**Data Input**	**Data Output**	**Status Output**
	size_t *retLength			
parser_ getParsedLength	Parser *p size_t inputLength size_t *ret	size t inputLength	size t *ret	Success or ERROR_TYPE
parser_ getRestoredLength	Parser *p size t inputLength size_t *ret	size_t inputLength	size t *ret	Success or ERROR_TYPE
module_getStatus	int *fipsEnabled MODULE_STATE *state	N/A	int *fipsEnabled MODULE_STATE *state	Success or ERROR_TYPE
module_destroy (Zeroization)	N/A	N/A	N/A	Success or ERROR_TYPE
Self-Tests (Power cycle)	N/A	N/A	N/A	

Definition of Critical Security Parameters (CSPs)

The following are CSPs contained within the module:

- **Private_Import_Key_RSA_Unwrap**: Used by the SecureParser module to unwrap encrypted keys sent to it by applications. All keys sent to the SecureParser will be RSA key wrapped by applications with CSP **Public_Import_Key_RSA_Wrap**. Note that each SecureParser keystore will have its own associated RSA public/private import keypair.
- **Workgroup_Key_AES:** Used to NIST key wrap internally generated Session keys (Session_Key_AES, Session_Data_Integrity_Key_HMAC) before they are split and output into shares, also used to unwrap key shares in headers being restored.
- **Session_Key_AES**: Used to encrypt all plaintext data prior to data splitting. Encrypted with Workgroup_Key_AES during NIST Key wrapping and then placed into share headers.
- **Session_Data_Integrity_ Key_HMAC**: SHA1 or SHA 256 used for ciphertext data integrity prior to data splitting, and also for the first share data integrity after data splitting. Encrypted with Workgroup_Key_AES during NIST Key wrapping and then placed into share headers.
- **Share_Integrity_Key_HMAC:** Optional HMAC-SHA1 or HMAC-SHA256 key used for additional ciphertext share data integrity after data splitting + the first share data integrity. Never output.
- **Share_Integrity_Key_ DSA_Sign:** Optional DSA Private Key (PEM or ANSI) used to sign ciphertext share data after the data splitting process. Never output.
- **Share_Integrity_Key_ RSA_Sign:** Optional RSA Private Key used to sign ciphertext share data after the data splitting process. Never output.
- **Share_Integrity_Key_ ECDSA_Sign:** Optional ECDSA Private Key (PEM or ANSI) used to sign ciphertext share data after the data splitting process. Never output.

- **PRNG_Seed_Key**: Imported from standard operating system services within the physical boundary of the general purpose computer.. Used to seed the module's own FIPS ANSI X9.31 pseudo random number generator. **Rationale of strength** follows PRNG_Seed_Value description.
- **PRNG_Seed_Value**: Imported from standard operating system services within the physical boundary of the general purpose computer.. Used to seed the module's own FIPS ANSI X9.31 pseudo random number generator. Must not be identical to PRNG_Seed_Key. **Since the PRNG seed comes from the operating system, which is outside the logical boundary of the module, for the purposes of FIPS 140-2, the entropy of this seed may be assumed to be equal to the length of the seed. The seed length is 128 bits.**
- **SecureParser PRNG_State**: Internal state of the SecureParser's PRNG (v4.5.0 Cert. #330, v4.5.1 Cert. #401).

Definition of Public Keys

The following are the public keys contained in the module:

- **Public_Import_Key_RSA_Wrap**: Used by applications to wrap keys they are sending to the SecureParser module. All keys sent to the SecureParser must be RSA wrapped. Note that each SecureParser keystore will have its own public/private keypair.
- **SW_Integrity_Key_DSA_Verify**: Used for verification of the signed module executable during power-on self-tests. Hard coded in the module.
- **Share_Integrity_Key_DSA_Verify**: Optional DSA Public Key (PEM or ANSI) used to verify ciphertext share data during the restoration process. Can be imported into the module from an X509 certificate.
- **Share_Integrity_Key_RSA_Verify**: Optional RSA Public Key used to verify ciphertext share data during the restoration process. Can be imported into the module from an X509 certificate.
- **Share_Integrity_Key_ECDSA_Verify**: Optional ECDSA Public Key (PEM or ANSI) used to verify ciphertext share data during the restoration process. Can be imported into the module from an X509 certificate.

Definition of CSPs Modes of Access

Table 6 defines the relationship between access to CSPs and the different module services. The modes of access shown in the table are defined as follows:

- G = Generate CSP
- R = Read CSP
- W = Write CSP
- Z = Zeroize CSP

Table 6 – CSP Access Rights within Roles & Services
Ref. SecureParser Specification: 4.4 Critical Security Parameters

Role	Service	Cryptographic Keys and CSPs Access Operation

C.O.	**User**		
X		**module_initialize**	PRNG_Seed_Key, **G-Z** PRNG_Seed_Value, **G-Z** PRNG_State, **W**
X		**parser_create**	N/A
X		**parser_destroy**	Session_Key_AES, **Z** Session_Data_Integrity_ Key_HMAC, **Z**
X		**parser_generateHeaders**	Session_Key_AES, **G-R-W** Session_Data_Integrity_ Key_HMAC, **G-R-W** Workgroup_Key_AES, **R** PRNG_State, **R-W**
X		**parser_restoreHeaders**	Session_Key_AES, **R-W** Session_Data_Integrity_ Key_HMAC, **R-W** Workgroup_Key_AES, **R**
X		**keystore_create**	Private_Import_Key_RSA_ Unwrap, **G** Public_Import_Key_RSA_ Wrap, **G**
X		**keystore_destroy**	All CSPS in the keystore, **Z**
X		**keystore_addKeyFromBuffer**	All keys that are imported, **R-W** Private_Import_Key_RSA_ Unwrap, **R**
X		**keystore_removeKey**	Specified key in volatile keystore structure **Z**
X		**keystore_getKeyType**	Specified key in volatile keystore structure **R**
X		**keystore_getkeylength**	Specified key in volatile keystore structure **R**
X		**keystore_keyexists**	Specified key in volatile keystore structure **R**

Role		**Service**	**Cryptographic Keys and CSPs Access Operation**
C.O.	**User**		
	X	**parser_parseData**	Session_Key_AES, **R** Session_Data_Integrity_ Key_HMAC, **R** Workgroup_Key_AES, **R** PRNG_State, **R-W**
	X	**parser_restoreData**	Session_Key_AES, **R** Session_Data_Integrity_ Key_HMAC, **R** Workgroup_Key_AES, **R**
	X	**parser_getHeaderInfo**	N/A
	X	**parser_getParsedLength**	N/A
	X	**parser_getRestoredLength**	N/A
X	X	**module_getstatus**	N/A
X		**Zeroization: module_destroy**	All CSPs (includes imported public keys and everything in the volatile keystore), **Z**
		Self tests (power cycle)	SW Integrity: Digital signature using Security First Corp. public DSA key SW_Integrity_Key_ DSA_Verify, **R**

6. Operational Environment

The FIPS 140-2 Area 6 Operational Environment requirements are applicable because the SecureParser module operates in a modifiable operational environment on a general purpose computer. See the description of the operational environment in section “1. Module Overview” above.

7. Security Rules

The SecureParser cryptographic module’s design corresponds to the module’s security rules. This section documents the security rules enforced by the cryptographic module to implement the security requirements of this FIPS 140-2 Level 1 software module.

1. The SecureParser module interfaces shall be logically distinct from each other as defined by the SecureParser API for the following interfaces: Data Input; Data

Output; Control Input; Status Output.

2. Status information shall not contain CSPs or sensitive data that if misused could lead to a compromise of the module.
3. Data output shall be inhibited during self-tests, and while in error states.
4. Data output shall be disconnected from the module processes that perform key generation, and plaintext CSP zeroization (the module will not support manual key entry).
5. Two independent internal actions will be required to output data via the output interface through which sensitive restored plaintext share data is output.
6. Plaintext secret/private key output is not supported. No SecureParser API calls will permit secret/private key output.
7. The SecureParser module shall provide two distinct operator roles. These are the User role, and the Cryptographic-Officer role.
8. The SecureParser module shall not support concurrent operators.
9. The SecureParser module shall not support a maintenance role.
10. The SecureParser module shall not support a bypass capability.
11. The SecureParser module does not provide any operator authentication.
12. Explicit service API calls into the SecureParser module shall allow the implicit assumption of operator roles.
13. The SecureParser module includes the following operational and error states: Power on/off state; Crypto officer state; User state; Self-test state; Error state; Key/CSP Entry state.
14. Recovery from error states shall be possible by power cycling the module.
15. Secret keys, private keys, and CSPs shall be protected within the cryptographic module from unauthorized disclosure, modification, and substitution.
16. Public keys shall be protected within the cryptographic module against unauthorized modification and substitution.
17. An Approved RNG (ANSIX9.31 with AES) shall be used for the generation of AES cryptographic keys within the module.
18. An Approved RNG (ANSIX9.31 with AES) shall be used for the generation of RSA cryptographic keypairs within the module.
19. The PRNG seed and seed key shall not have the same value.
20. Compromising the security of the key generation method (e.g., guessing the seed value to initialize the deterministic RNG) shall require as least as many operations as determining the value of the generated key.
21. The SecureParser module shall associate all cryptographic keys (secret, private, or public) stored within the module with the correct entity (KeyID) to which the key is assigned.
22. The SecureParser module shall provide a method to zeroize all plaintext secret and private cryptographic keys and CSPs within the module in a time that is not sufficient to compromise the plaintext secret and private keys and CSPs (service: module_destroy).
23. Power-on Self-tests will not require operator intervention, they will be performed automatically when the module is initialized.
24. The cryptographic module shall perform the following self-tests:
    a. Power up Self-Tests:

        i. Cryptographic algorithm tests:
            1. PRNG KAT, covers AES Encrypt
            2. AES Decrypt KAT (ECB mode with 256-bit key)
            3. HMAC KATS using SHA-256, covers SHA-256 hashing
            4. DSA sign/verify using SHA-1, covers SHA-1 hashing.
            5. RSA-PSS sign/verify
            6. RSA encrypt/decrypt
            7. ECDSA sign/verify
        ii. Software Integrity Test – DSA public key verification of a private key signature.
    b. Conditional Self-Tests:
        i. Continuous Random Number Generator (PRNG) test – performed on each sample from the PRNG (each sample will be 128 AES bits).
        ii. Pairwise consistency test – performed each time an RSA "import" keypair is generated inside the module.

25. If the SecureParser module fails a self-test, the module shall enter an error state and output an error indicator via the status output interface.
26. The SecureParser module shall not perform any cryptographic operations while in an error state.
27. When the power-up tests are completed, the results (i.e., indications of success or failure) shall be output via the "status output" interface.
28. The operator shall be capable of commanding the module to perform the power-up self-tests at any time by power cycling the cryptographic module.

This section documents the security rules imposed by the vendor:

1. For v4.5.0 to run in a FIPS Approved mode external calling applications cannot use the cryptographic module in a multi-threaded manner. (Note: v4.5.1 does allow multi-threaded use in FIPS Approved mode.)
2. An approved encryption mode and an approved integrity mechanism must be requested by calling applications to run the SecureParser module in FIPS Approved mode.
3. Workgroup keys shall be mandatory for the SecureParser module to run in a FIPS Approved mode.
4. Workgroup keys shall not be placed in data shares.
5. The SecureParser module shall encrypt all share data using AES session keys.
6. The SecureParser module shall provide for the integrity of encrypted data shares using HMAC-SHA1 or HMAC-SHA256. In addition an optional configurable second layer of integrity will be provided using either, HMAC-SHA1 or HMAC-SHA256, DSA, ECDSA, or RSA.
7. All Secret & Private Key Entry/Output is encrypted using RSA key wrapping.

8. Physical Security

FIPS 140-2 Area 5 Physical Security requirements are not applicable because the SecureParser cryptographic module is a software only module.

9. Mitigation of Other Attacks Policy

The SecureParser module has not been designed to mitigate any specific attacks.

10. Definitions and Acronyms

Share

A partition of data created after the SecureParser is enacted to parse data.

Mandatory Share

A mandatory share is a share that must be present for the proper recovery of data. In other words, all mandatory shares must be available. The number of mandatory shares is denoted by L.

Non-mandatory Share

The SecureParser allows for the reconstruction of data with a subset of non-mandatory shares. The number of non-mandatory shares is denoted by N and the number of non-mandatory shares that must be available to restore is denoted by M.

M of N

In this document, we refer to M of N shares, which is intended to mean M of N non-mandatory shares and L mandatory shares. For example, “without M of N shares...” means without at least M non-mandatory shares and L mandatory shares.

Trusted

Something that is trusted is known to meet its security assumptions. For example, a trusted share is known to be valid, untampered with, and otherwise uncompromised by any adversaries.

Workgroup key

This can be any encryption key that can be used to encrypt or decrypt data. Often it is a shared key between users of the application working together.

Integrity Authentication key:

This can be any key used for generating or verifying a MAC or signature of data.