SecureParser®

Version 4.7.0

Security Policy

Revision 1.34

23 September 2010

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	11/8/2007	Security First Corp. (ISE input)	V4.5.1 revision for submission. Updated API section, removed references to single-threaded requirement, added references for libparser4.sys.
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.
1.1	08/15/2008	Security First Corp. (ISE Input)	V4.6 revision for submission. Updated API, added algorithms, added operating systems.
1.2	02/02/2009	Security First Corp. (ISE input)	V4.7.0 revision for submission. Updated API section, added description of RPU. Removed all operating systems but Ubuntu and the Windows kernel.
1.21	02-17-2009	Security First Corp. (ISE input)	Removed function get_errorlog, it is disabled in FIPS mode.
1.22	02-20-2009	Security First Corp. (ISE input)	Prior Track Changes accepted. Prior comments removed. Table 1: Level of Physical Security NA → 1. Real picture for Figure 2 – Image of the Accelium
1.23	02-20-2009	Security First Corp.	Added real picture of the RPU for Figure 2.

		(ISE input)	
1.24	02-23-2009	Security First Corp. (ISE input)	Updated photos in Figure 2. Adjusted verbiage in the physical RPU section.
1.25	02-26-2009	Security First Corp. (ISE input)	Clarified key wrapping description under Security Rules.
1.26	04-01-2009	Security First Corp. (ISE input)	Final edits before CMVP submission.
1.30	08-06-2009	Security First Corp. (ISE input)	Added Windows Server 2003 references in anticipation of update submission. Responses to CMVP Comments.
1.31	08-06-2009	Security First Corp. (ISE input)	Minor formatting changes.
1.32	09-02-2009	Security First Corp. (ISE input)	Updated for SW-only release.
1.33	11-18-2009	Security First Corp. (ISE input)	More updates for SW-only release.
1.34	09-23-2010	Security First Corp (ISE input)	Updated Module Overview and Modes of Operation Responses to CMVP Comments

TABLE OF CONTENTS

1. Module Overview

The SecureParser® (SW Version 4.7.0) a FIPS software-only module (hereafter known as "the module"). The module is a security and 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 plaintext is transmitted across a network or is placed on a single storage device. During the parse 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. The shares can be stored in geographically disbursed nodes providing for continuous access to online information.

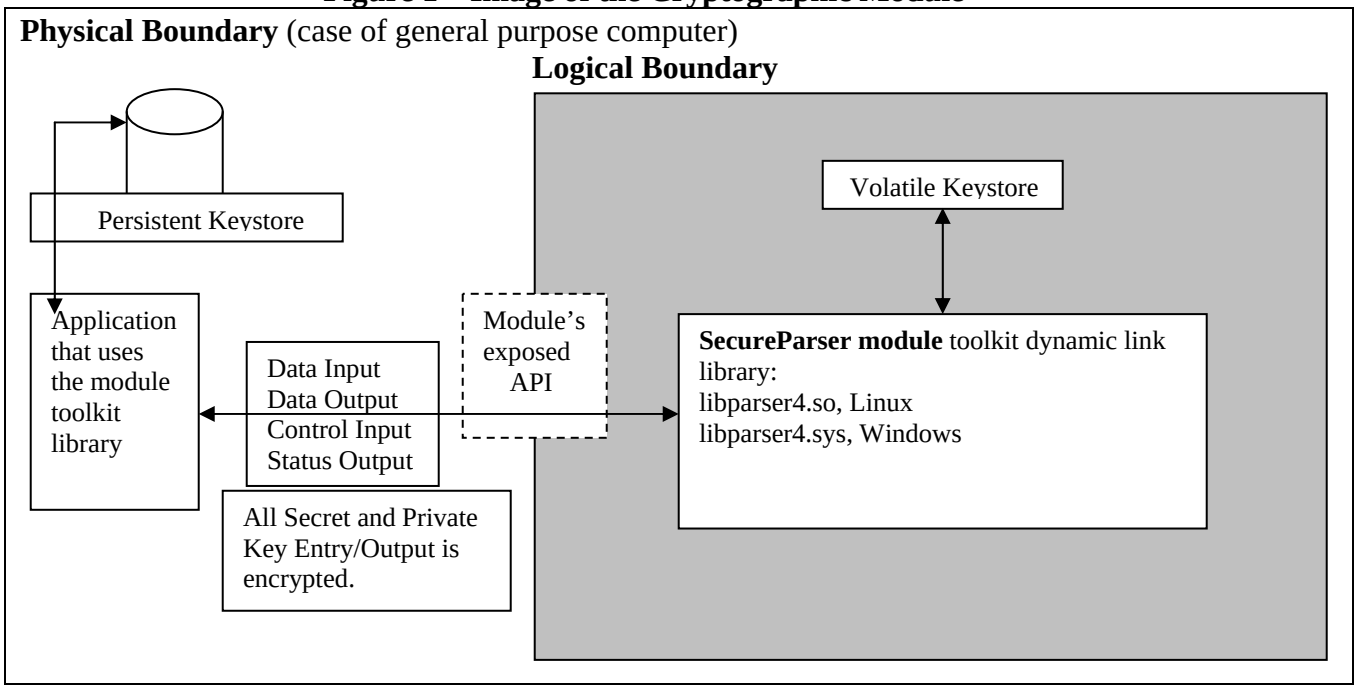
Each share contains a cryptographically strong integrity check that prevents tampering with the stored data and is immediately recognized by the other shares. Any change to the data in a share precludes that share from being used in the data rebuild process. The encryption, integrity and Information Dispersal Algorithm (IDA) session keys are encrypted with long-term, external workgroup keys, and a per-session key encrypting key that is shared and stored with the data.

Data availability through redundant shares allows for a return to operations in the face of lost or corrupted shares due to environmental, malicious 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



Logical Boundary

When operating on the Linux operating system Ubuntu 8, the SecureParser cryptographic logical boundary is defined as containing the SecureParser libparser4.so. Seed values for the SecureParser's random number generator are imported from standard operating system services within the physical boundary of the general purpose computer.

When operating on Windows XP and Windows Server 2003, the SecureParser module cryptographic logical boundary is defined as containing the SecureParser libparser4.sys. Seed values for the SecureParser's random number generator are 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: Ubuntu 8, Windows XP, and Windows Server 2003.

Operational testing was performed on all the above operating systems.

Additionally, the module runs without recompilation on other GPC's equipped with x64 compatible processors running kernels compatible with Ubuntu 8 and Windows Server 2003.

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	1
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:

Software (CPU) Algorithms:

- AES-CBC/ECB - 128/192/256 bit key Cert. #1222
- AES-CTR - 128/192/256 bit key Cert. #1222
- AES-GCM 128/192/256 bit key Cert. #1223
- HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512 Cert. #714
- SHA-1, SHA-256, SHA-384, SHA-512 hashing Cert. #1124
- DSA sign/verify – 1024 bit key Cert. #405
- RSA sign/verify – 1024/2048/4096 bit key Cert. #590
- RNG Key Generation ANSI X9.31 with AES Cert. #678
- ECDSA sign/verify – 521 bit key Cert. #144

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. *AES (Cert. #1222 key wrapping; key establishment methodology provides 128, 192 or 256 bits of encryption strength) for NIST Key Wrapping per FIPS 140-2 Annex D)*

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).*

In FIPS mode the SecureParser module does not support any non-allowed FIPS algorithms.

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:

- 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( ).

In non-FIPS mode, the module supports the option to not use encryption or message authentication coding.

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. Note that the module does not have any persistent CSPs. All CSPs are zeroed when the module is shut down or transitions between modes.

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.

Role	Authorized Services
	• **parser_regenerateHeaders.** Produces headers associated with an already-configured parser context. • **parser_recoverHeaders.** Recovers missing headers and places them in the output buffers that are unused. • **parser_setWorkgroupKeys.** Changes the workgroup keys in an existing parser context. • **keystore_getImportKey.** 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 ephemeral public/private keypair). • **keystore_create.** Allocates memory for a volatile KeyStore structure, and creates a non-persistent RSA public/private 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. Zeroes non-persistent CSPs including the RSA import public/private keypair, and the volatile Keystores. ALL keystores and ALL parsers in memory are zeroed. • **Self-tests**: Power cycle
User: The User role is assumed when applications call the module's exposed API functions that perform	• **parser_parseData.** Parses data from the input buffer into the output buffers. • **parser_restoreData.** Restores data from the output buffers into the input buffer. • **parser_parseDataEx.** Parses an array of input buffers into

Role	Authorized Services
general cryptographic services.	the output buffers. • **parser_recoverData.** Rebuilds all N data shares given only M input shares. • **parser_getFieldOffsets.** Returns an array of {share number, offset, length} tuples necessary to create M of N shares for a given M value and a set of "N of N" parsed shares. • **parser_setFaultTolerance.** Sets a new fault tolerance value (M). Designed for use with parser_getFieldOffsets(). • **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	N/A	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 uint8 *encWgKeyId uint32 encWgKeyIdMem uint32 encWgKeyIdLength uint8 *macWgKeyId uint32 macWgKeyIdMem uint32 macWgKeyIdLength uint8 *idaWgKeyId uint32 idaWgKeyIdMem uint32 idaWgKeyIdLength	N/A	uint8 **outputBuffers uint32 *outputBufferLengths	Success or ERROR_TYPE

Service	**Control Input**	**Data Input**	**Data Output**	**Status Output**
	AUTH_TYPE postAuthMode HASH_TYPE postHashMode uint8 *postAuthPubKeyId uint32 postAuthPubKeyIdMem uint32 postAuthPubKeyIdLength uint8 *postAuthPrivKeyId uint32 postAuthPrivKeyIdMem uint32 postAuthPrivKeyIdLength uint32 *outputBufferMems int outputBuffersCount			
parser_ restoreHeaders	Parser *p KeyStore *ks HASH_TYPE hashMode uint8 *encWgKeyId uint32 encWgKeyIdMem uint32 encWgKeyIdLength uint8 *macWgKeyId uint32 macWgKeyIdMem uint32 macWgKeyIdLength uint8 *idaWgKeyId uint32 idaWgKeyIdMem uint32 idaWgKeyIdLength AUTH_TYPE postAuthMode HASH_TYPE postHashMode uint8 *postAuthPubKeyId uint32 postAuthPubKeyIdMem uint32 postAuthPubKeyIdLength uint8 *postAuthPrivKeyId uint32 postAuthPrivKeyIdMem uint32 postAuthPrivKeyIdLength uint32 * inputBufferMems uint32 * inputBufferLengths int inputBuffersCount int trustedShareNumber	uint8 **inputBuffers	N/A	Success or ERROR_TYPE
parser_ regenerateHeaders	Parser *p uint32 *outputBufferMems int outputBuffersCount	N/A	uint8 **outputBuffers uint32 *outputBufferLengths	Success or ERROR_TYPE
parser_ recoverHeaders	KeyStore *ks HASH_TYPE hashMode char *workgroupKeyId uint32 workgroupKeyIdMem uint32 workgroupKeyIdSize, AUTH_TYPE postAuthMode char *postAuthKeyId	uint8 **outputBuffers	uint8 **outputBuffers uint32 *outputBufferLengths	Success or ERROR_TYPE

Service	Control Input	Data Input	Data Output	Status Output
	uint32 postAuthKeyIdMem uint32 postAuthKeyIdSize uint32 *outputBufferMems uint32 *outputBufferLengths			
parser_ setWorkgroupKeys	Parser * p char * encWgKeyId uint32 encWgKeyIdMem uint32 encWgKeyIdLength char * macWgKeyId uint32 macWgKeyIdMem uint32 macWgKeyIdLength char * idaWgKeyId uint32 idaWgKeyIdMem uint32 idaWgKeyIdLength	N/A	N/A	Success or ERROR_TYPE
keystore_ getImportKey	KeyStore *ks uint32 bufferMem	N/A	uint8 *buffer uint32 *bufferLength	Success or ERROR_TYPE
keystore_create	int minimumKeyCount	N/A	KeyStore **ret	Success or ERROR_TYPE
keystore_destroy	KeyStore *ks	N/A	N/A	Success or ERROR_TYPE
keystore_ addKeyFromBuffer	KeyStore *ks uint32 bufferMem uint32 bufferLength char *id uint32 idMem uint32 idLength char *passphrase uint32 passphraseMem uint32 passphraseLength IMPORT_TYPE importType	uint8* buffer char *id	N/A	Success or ERROR_TYPE
keystore_ removeKey	KeyStore *ks char *id uint32 idMem uint32 idLength	N/A	N/A	Success or ERROR_TYPE
keystore_ getKeyType	KeyStore *ks char *id uint32 idMem uint32 idLength	N/A	KEY_TYPE *keyType	Success or ERROR_TYPE
keystore_ getKeyLength	KeyStore *ks char *id uint32 idMem uint32 idLength	N/A	uint32 *keyLength	Success or ERROR_TYPE
keystore_keyExists	KeyStore *ks char *id uint32 idMem uint32 idLength	N/A	int *keyExists	Success or ERROR_TYPE
parser_parseData	Parser *p uint32 inputBufferLength uint32 inputBufferMem uint32 *outputBufferMems int outputBuffersCount	uint8 *inputBuffer	uint8**outputBuffers uint32 *outputBufferLengths	Success or ERROR_TYPE

Service	Control Input	Data Input	Data Output	Status Output
parser_parseDataEx	Parser * p uint32 * inputBufferMems uint32 * inputBufferLengths uint32 inputBuffersCount uint32 * outputBufferMems uint32 outputBuffersCount	uint8 ** inputBuffers	uint8**outputBuffers uint32 *outputBufferLengths	Success or ERROR_TYPE
parser_restoreData	Parser *p uint32 outputBufferMem uint32 *inputBufferLengths uint32 *inputBufferMems int inputBuffersCount int trustedShareNumber	uint8 **inputBuffers	uint8 *outputBuffer uint32 *outputBufferLength	Success or ERROR_TYPE
parser_recoverData	Parser *p uint32 *outputBufferLengths uint32 *outputBufferMems int outputBuffersCount int trustedShareNumber	uint8 *outputBuffers	uint8**outputBuffers uint32 *outputBufferLengths	Success or ERROR_TYPE
parser_ getHeaderInfo	uint32 headerMem uint32 headerLength uint32 retMem	DATAFIELD_TYPE t uint8 *header	void *ret uint32 *retLength	Success or ERROR_TYPE
parser_ getParsedLength	Parser *p	uint32 inputLength	uint32 *ret	Success or ERROR_TYPE
parser_ getRestoredLength	Parser *p	uint32 inputLength	uint32 *ret	Success or ERROR_TYPE
parser_ getFieldOffsets	Parser * p uint32 plaintextLength int intendedM	N/A	FieldOffsets * o	Success or ERROR_TYPE
parser_ setFaultTolerance	Parser *p int newM	N/A	N/A	Success or ERROR_TYPE
module_getStatus	N/A	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_Enc:** Used to NIST key wrap internally generated encryption session key (Session_Key_Enc), also used to unwrap encryption session key when headers are being restored.

- **Workgroup_Key_Mac:** Used to NIST key wrap internally generated integrity session key (Session_Key_Mac), also used to unwrap integrity session key when headers are being restored.
- **Workgroup_Key_Ida:** Used to NIST key wrap internally generated IDA session key (Session_Key_Ida), also used to unwrap IDA session key when headers are being restored.
- **Session_Key_Enc**: Used to encrypt all plaintext data prior to data splitting. Encrypted by Workgroup_Key_Enc using the NIST Key wrap and then placed into share headers.
- **Session_Key_Mac**: HMAC-SHA1, SHA256, SHA384, or SHA512 key used for ciphertext data integrity once splitting is complete. Encrypted by Workgroup_Key_Mac using the NIST Key wrap and then placed into share headers.
- **Session_Key_Ida**: Random seed used as input to IDAs for adding randomness. Encrypted by Workgroup_Key_Mac using the NIST Key wrap and then placed into share headers.
- **Share_Integrity_Key_HMAC:** Optional HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, or HMAC-SHA512 key used for additional ciphertext share data integrity after data splitting. 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 (Cert. #584).

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 key pair.
- **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 = Zero 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_Enc, **Z** Session_Key_Mac, **Z** Session_Key_Ida, **Z**
X		**parser_generateHeaders**	Session_Key_Enc, **G-R-W** Session_Key_Mac, **G-R-W** Session_Key_Ida, **G-R-W** Workgroup_Key_Enc, **R** Workgroup_Key_Mac, **R** Workgroup_Key_Ida, **R** Share_Integrity_Key_HMAC, **R** Share_Integrity_Key_DSA_Sign, **R** Share_Integrity_Key_RSA_Sign, **R** Share_Integrity_Key_ECDSA_Sign, **R** PRNG_State, **R-W**
X		**parser_restoreHeaders**	Session_Key_Enc, **R-W** Session_Key_Mac, **R-W** Session_Key_Ida, **R-W**

Role		Service	Cryptographic Keys and CSPs Access Operation
C.O.	**User**		
			Workgroup_Key_Enc, **R** Workgroup_Key_Mac, **R** Workgroup_Key_Ida, **R** Share_Integrity_Key_HMAC, **R**
X		**parser_regenerateHeaders**	Session_Key_Enc, **R** Session_Key_Mac, **R** Session_Key_Ida, **R** Workgroup_Key_Enc, **R** Workgroup_Key_Mac, **R** Workgroup_Key_Ida, **R** Share_Integrity_Key_HMAC, **R** Share_Integrity_Key_DSA_Sign, **R** Share_Integrity_Key_RSA_Sign, **R** Share_Integrity_Key_ECDSA_Sign, **R**
X		**parser_recoverHeaders**	Session_Key_Enc, **R** Session_Key_Mac, **R** Session_Key_Ida, **R** Workgroup_Key_Enc, **R** Workgroup_Key_Mac, **R** Workgroup_Key_Ida, **R** Share_Integrity_Key_HMAC, **R** Share_Integrity_Key_DSA_Sign, **R** Share_Integrity_Key_RSA_Sign, **R** Share_Integrity_Key_ECDSA_Sign, **R**
X		**parser_setWorkgroupKeys**	Session_Key_Enc, **R** Session_Key_Mac, **R** Session_Key_Ida, **R** Workgroup_Key_Enc, **W** Workgroup_Key_Mac, **W** Workgroup_Key_Ida, **W**
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

Role		Service	Cryptographic Keys and CSPs Access Operation
C.O.	**User**		
			structure **R**
X		**keystore_getImportKey**	Public_Import_Key_RSA_Wrap, **R**
	X	**parser_parseDataEx**	Session_Key_Enc, **R** Session_Key_Mac, **R** Session_Key_Ida, **R** Share_Integrity_Key_HMAC, **R** Share_Integrity_Key_DSA_Sign, **R** Share_Integrity_Key_RSA_Sign, **R** Share_Integrity_Key_ECDSA_Sign, **R** PRNG_State, **R-W**
	X	**parser_parseData**	Session_Key_Enc, **R** Session_Key_Mac, **R** Session_Key_Ida, **R** Share_Integrity_Key_HMAC, **R** Share_Integrity_Key_DSA_Sign, **R** Share_Integrity_Key_RSA_Sign, **R** Share_Integrity_Key_ECDSA_Sign, **R** PRNG_State, **R-W**
	X	**parser_restoreData**	Session_Key_Enc, **R** Session_Key_Mac, **R** Session_Key_Ida, **R** Share_Integrity_Key_HMAC, **R**
	X	**parser_recoverData**	Session_Key_Mac, **R** Session_Key_Ida, **R** Share_Integrity_Key_HMAC, **R** Share_Integrity_Key_DSA_Sign, **R** Share_Integrity_Key_RSA_Sign, **R** Share_Integrity_Key_ECDSA_Sign, **R**
	X	**parser_getHeaderInfo**	N/A
	X	**parser_getFieldOffsets**	N/A
	X	**parser_setFaultTolerance**	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**

For each service listed in Table 6 above, the following identifies all CSPs that are entered into and output from the module during execution of each service.

- module_initialize:
  - Entry: N/A.
  - Output: N/A.
- parser_create:
  - Entry: N/A.
  - Output: N/A.
- parser_destroy:
  - Entry: N/A.
  - Output: N/A.
- parser_generateHeaders:
  - Entry: N/A.
  - Output:
    - Session_Key_Enc (encrypted with Workgroup_Key_Enc).
    - Session_Key_Mac (encrypted with Workgroup_Key_Mac).
    - Session_Key_Ida (encrypted with Workgroup_Key_Ida).
- parser_restoreHeaders:
  - Entry:
    - Session_Key_Enc (encrypted with Workgroup_Key_Enc).
    - Session_Key_Mac (encrypted with Workgroup_Key_Mac).
    - Session_Key_Ida (encrypted with Workgroup_Key_Ida).
  - Output: N/A.
- parser_regenerateHeaders:
  - Entry: N/A.
  - Output:
    - Session_Key_Enc (encrypted with Workgroup_Key_Enc).
    - Session_Key_Mac (encrypted with Workgroup_Key_Mac).
    - Session_Key_Ida (encrypted with Workgroup_Key_Ida).
- parser_recoverHeaders:
  - Entry:
    - Session_Key_Enc (encrypted with Workgroup_Key_Enc).
    - Session_Key_Mac (encrypted with Workgroup_Key_Mac).
    - Session_Key_Ida (encrypted with Workgroup_Key_Ida).
  - Output:
    - Session_Key_Enc (encrypted with Workgroup_Key_Enc).
    - Session_Key_Mac (encrypted with Workgroup_Key_Mac).
    - Session_Key_Ida (encrypted with Workgroup_Key_Ida).
- parser_setWorkgroupKeys:
  - Entry: N/A.
  - Output: N/A.
- keystore_create:
  - Entry: N/A.
  - Output: N/A.
- keystore_destroy:
  - Entry: N/A.

    - Output: N/A.
- keystore_addKeyFromBuffer:
    - Entry:
        - Workgroup_Key_Enc (encrypted with Public_Import_Key_RSA_Wrap).
        - Workgroup_Key_Mac (encrypted with Public_Import_Key_RSA_Wrap).
        - Workgroup_Key_Ida (encrypted with Public_Import_Key_RSA_Wrap).
        - Share_Integrity_Key_HMAC (encrypted with Public_Import_Key_RSA_Wrap).
        - Share_Integrity_Key_DSA_Sign (encrypted with Public_Import_Key_RSA_Wrap).
        - Share_Integrity_Key_RSA_Sign (encrypted with Public_Import_Key_RSA_Wrap).
        - Share_Integrity_Key_ECDSA_Sign (encrypted with Public_Import_Key_RSA_Wrap).
    - Output: N/A.
- keystore_removeKey:
    - Entry: N/A.
    - Output: N/A.
- keystore_getKeyType:
    - Entry: N/A.
    - Output: N/A.
- keystore_getKeyLength:
    - Entry: N/A.
    - Output: N/A.
- keystore_keyExists:
    - Entry: N/A.
    - Output: N/A.
- keystore_getImportKey:
    - Entry: N/A.
    - Output: Public_Import_Key_RSA_Wrap (plaintext).
- parser_parseDataEx:
    - Entry: N/A.
    - Output: N/A.
- parser_parseData:
    - Entry: N/A.
    - Output: N/A.
- parser_restoreData:
    - Entry: N/A.
    - Output: N/A.
- parser_recoverData:
    - Entry: N/A.
    - Output: N/A.
- parser_getHeaderInfo:
    - Entry: N/A.
    - Output: N/A.
- parser_getFieldOffsets:

    - Entry: N/A.
    - Output: N/A.
- parser_setFaultTolerance:
    - Entry: N/A.
    - Output: N/A.
- parser_getParsedLength:
    - Entry: N/A.
    - Output: N/A.
- parser_getRestoredLength:
    - No key Entry or Output.
- module_getstatus:
    - Entry: N/A.
    - Output: N/A.
- Zeroization: module_destroy:
    - Entry: N/A.
    - Output: N/A.
- Self tests (power cycle):
    - Entry: N/A.
    - Output: N/A.

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-only 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 zeroing (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 (ANSI X9.31 with AES) shall be used for the generation of AES cryptographic keys within the module.
18. An Approved RNG (ANSI X9.31 with AES) shall be used for the generation of RSA cryptographic key pairs 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 zero 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. Software cryptographic algorithm tests:
            1. RNG KAT, covers AES Encrypt
            2. AES Decrypt KAT (ECB mode with 256-bit key)
            3. AES Encrypt and Decrypt KAT (GCM mode with 128,192, and 256-bit keys)
            4. HMAC KATS using SHA-256, SHA-384, SHA-512, covers SHA-256, SHA-384, and SHA-512 hashing
            5. DSA sign/verify using SHA-1, covers SHA-1 hashing
            6. RSA-PSS sign/verify
            7. RSA encrypt/decrypt
            8. ECDSA sign/verify
        ii. Software/Firmware Integrity Test – DSA public key verification of a private key signature. Covers all module executables listed in **Figure 1 - Image of the Cryptographic Module**.
    b. Conditional Self-Tests:
        i. Continuous Random Number Generator (RNG) test – performed on each sample from the RNG (each sample will be 128 AES bits).
        ii. Pairwise consistency test – performed each time an RSA “import” key pair 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.
29. None of the mentioned hardware, software, or firmware components will be excluded from the module.

This section documents the security rules imposed by the vendor:

1. An approved encryption mode and an approved integrity mechanism must be requested by calling applications to run the SecureParser module in FIPS Approved mode.
2. Workgroup keys shall be mandatory for the SecureParser module to run in a FIPS Approved mode.
3. Workgroup keys shall not be placed in data shares.
4. The SecureParser module shall encrypt all share data using AES session keys.
5. The SecureParser module shall provide for the integrity of encrypted data shares using HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512, or GCM. In addition an optional configurable second layer of integrity will be provided using either HMAC-SHA1, HMAC-SHA256, HMAC-SHA384, HMAC-SHA512, DSA, ECDSA, or RSA.
6. All Secret and Private Keys entered via the module's keystore_addKeyFromBuffer( ) service are encrypted using RSA key wrapping. All Secret and Private Keys entered/output via the module's parser_parseData( ) and parser_restoreData( ) services are encrypted using NIST Key Wrapping.

8. Physical Security

The requirements of Section 4.5 of FIPS 140-2 Physical Security are not applicable as the SecureParser is a software-only module which operates on a general purpose computer.

9. Mitigation of Other Attacks Policy

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

10. References

OpenSSL: This product includes software developed by the OpenSSL Project for use in the OpenSSL Toolkit. (http://www.openssl.org)

11. 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.

Information Dispersal Algorithm (IDA):

An algorithm, possibly keyed, that divides information into multiple components. An IDA may add redundancy to allow the information to be recovered if some of the components are unavailable. Each IDA has an inverse algorithm that, when given some or all of the aforementioned components, produces the original information.