SureWave Mobile Defense Security Kernel

JP Mobile®
www.jpmobile.com
FIPS 140-2 Non-Proprietary Security Policy
Version 5.0.050623

1. Table of Contents

2. Introduction

2.1 Purpose

This document is JP Mobile' non-proprietary security policy for the SureWave Mobile Defense Security Kernel. This security policy describes how the SureWave Mobile Defense Security Kernel 5.0.050107 conforms to the level 1 security requirements imposed by the Federal Information Processing Standard (FIPS) 140-2. The SureWave Mobile Defense Security Kernel controls the cryptographic functions of various versions of the SureWave Mobile Defense 4.0 software for Palm, Pocket PC, and Symbian OS enabled devices. Although the same kernel is used in all versions of SureWave Mobile Defense 4.0, it has only been tested and validated for use on the Pocket PC 2003 Premium OS. The major role of this kernel is the implementation of the AES, DES, 3-DES, Blowfish, SHA-1, and MD5 algorithms.

2.2 References

For more information on the FIPS 140 Cryptographic Module Validation Program, please visit the National Institute of Standards and Technology (NIST) web site at: http://csrc.nist.gov/cryptval/

For more information about SureWave Mobile Defense's parent company, JP Mobile please visit:
http://www.jpmobile.com/

To find out more about the SureWave Mobile Defense product, please visit:
http://www.pdadefense.com/

2.3 Supplemental Documentation Listing

The following documents must accompany this Security Policy for the FIPS 140-2 validation package. These documents are proprietary to JP Mobile and are available only under appropriate non-disclosure agreements.

- Design Documents
- FIPS Finite State Model

The following Documents are available from JP Mobile upon request:

- SureWave Mobile Defense Administrator Guide
- SureWave Mobile Defense End User Guide
- SureWave Mobile Defense Installation Walkthrough
- SureWave Mobile Defense Upgrade Instructions
- SureWave Mobile Defense Installation Flowchart

3. Module Description and Details

3.1 Cryptographic Module Specification

Specification of Cryptographic Module

The SureWave Mobile Defense Security Kernel is a specific set of instructions that perform all cryptographic functions, approved and unapproved, of the SureWave Mobile Defense software. The SureWave Mobile Defense Security Kernel has been designed to meet the criteria for FIPS-140-2 Level 1 validation.

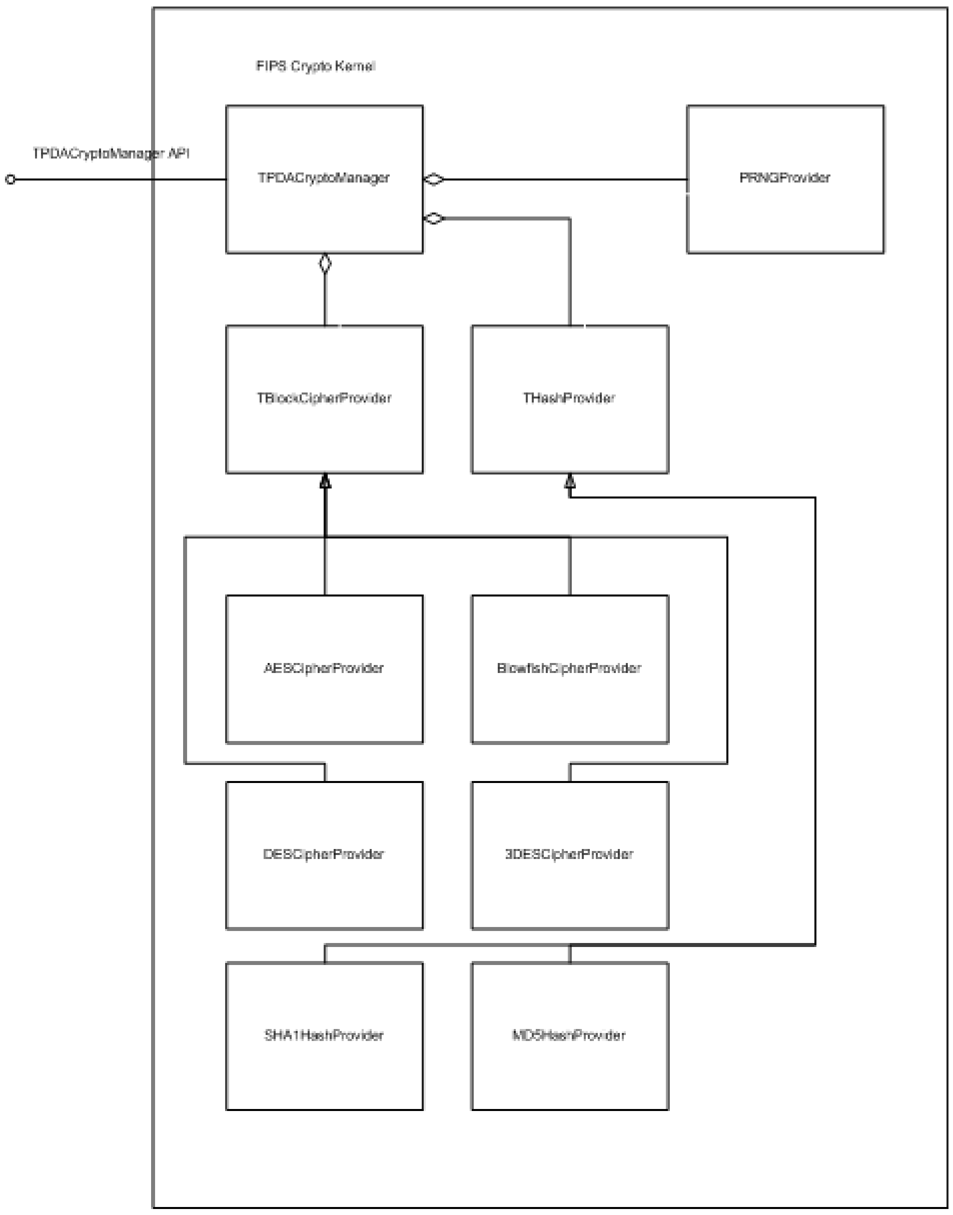
The SureWave Mobile Defense Security Kernel has a specific mode in which will disable all unapproved security functions such as MD5 and Blowfish. This specific mode is the FIPS Mode. FIPS Mode is initiated by the method `BOOL SetFIPSMode(BOOL bMode).` The actual state of the SureWave Mobile Defense Security Kernel may be queried with the method `BOOL GetFIPSMode(BOOL bMode).`

The temporary keys that are used for data ciphering/deciphering could compromise the security of the console. To ensure that the security of the module is not compromised, the TPDACryptoManager class has a method called WipeAllInternalKeys which erases all internal keys. The class object performs this action automatically each time the object is destroyed.

When tested on the specified device, the SureWave Mobile Defense Security Kernel is considered by FIPS140-2 to be a “Multi-Chip Standalone cryptographic module” and has been tested as such.

The MD5HashProvider and BlowfishCipherProvider Classes are disabled while the module has been set to FIPS validated mode.

Block Diagram:



The black line around the edge of the Block Diagram is representative of the module's Cryptographic Boundary. All input or output from the module must go through the TPDACryptoManager class using the TPDACryptoManager API as shown.

Each function of the SureWave Mobile Defense Security Kernel is contained within a specific class noted in the block diagram. These classes are TDPACryptoManager, PRNGProvider, TBlockCipherProvider, THashProvider, AESCipherProvider, BlowfishCipherProvider, DESCipherProvider, 3DESCipherProvider, SHA1HashProvider, MD5HashProvider.

DES (for legacy use only) (transitional phase only – valid until May 19, 2007) Details
DES accepts either 64-bit key with parity check bits according DES standard FIPS 46-3 or 56-bit key without parity check bits.
Supported modes: ECB and CBC.

3DES Details
3DES accepts either combined 168-bit key as a bundle of three DES 56-bit keys or 192-bit key as a bundle of three DES 64-bit keys according to FIPS 46-3.
The following three keying options can be used with 3DES

- Keying option 1: K1, K2, and K3 are independent keys.
- Keying option 2: Supported if combined key is constructed from K1, K2, and K3 = K1.
- Keying option 3: Supported if combined key is constructed from K1, K2 = K1, and K3 = K1. In this keying option, the strength of encryption is 56 bits at best.

Supported modes: ECB and CBC

AES Details
AES supports 128, 192, or 256-bit keys according to FIPS-197.
Supported modes: ECB and CBC.

PRNG Details
The PRNG implementation present in the SureWave Mobile Defense Security Kernel is based on FIPS 186-2 appendix 3.1.

3.2 Cryptographic Module Ports and Interfaces

All logical input and output is done through the TPDACryptoManager API. Physical input and output is done through the handheld device's standard I/O ports. The API function calls represent the Control Input Interface. The Data Input Interface consists of the parameters sent to the API and the Data Output Interface consists of the parameters returned from the API. The status output interface is made up of the return values and error codes provided by each function in the API. More specific information is defined below.

Interface	Logical Interface	Physical Port
Data Input	API parameters	IrDA port, Key Pad controller, LCD controller, USB port
Data Output	Parameters returned from API	IrDA port, controller, LCD controller, USB port
Control Input	API parameters and API calls	IrDA port, Key Pad controller, LCD controller, USB port
Status Output	Error codes and return	LCD controller

	values from API	
Power	Not available	Battery or DC port

Interface	**Parameters**
Data Input	Key, key length, algorithm ID, plain text, cipher text, encode/decode flag, IV, IV length, plain text length, cipher text length, CBC mode flag, hash input data, hash input data length, buffer memory pointer, buffer size
Data Output	Cipher text block, plain text block, algorithm ID, key length, hash digest, filled memory block*
Control Input	EnumBlockCipherProvidersL, GetBlockCipherProviderName, EnumHashProvidersL, GetHashProviderName, GetBlockCipherSupportedKeyLengthL, SetCipherScrambledKeyL, GetCipherScrambledKeyL, SetCipherPlainKeyL, GetCipherPlainKeyL, WipeAllInternalKeys, ProcessMemoryBlockL, GenerateHashL, GetHashLen, RandSeed, Random, PowerUpTestL, GetPowerTestState, IsNotBlocked, SetFIPSMode, SetFIPSMode**
Status Output	TEST_OK, TEST_FAIL, TEST_NOT_DONE, FALSE/TRUE as result of operation

* Memory block is filled with pseudo random values
** Control input methods parameters depend on particular method. It may be encryption key vector, key length, etc. See detailed description of each method below.

3.3 Roles, Services, and Authentication

The SureWave Mobile Defense Kernel does not perform user role authentication because it conforms to the level 1 security requirements imposed by the Federal Information Processing Standard (FIPS) 140-2. FIPS 140-2 Level 1 validation does not require authentication.

The operator assumes roles (User or Crypto Officer) implicitly when invoking these services.

3.4 Approved Mode of Operation

The module provides a specific FIPS approved mode of operation. The Crypto Officer may initiate this approved mode of operation via the Security kernel method `SetFIPSMode(BOOL)`. Only FIPS 140-2 approved algorithms are available in this mode. Unapproved algorithms such as Blowfish and MD5 will be disabled in the approved mode of operation. Please refer to the Secure Installation and Setup section for detailed information on how to place the module in FIPS mode.

3.5 Finite State Model

The Finite State Model is in a document folder labeled "Finite State Model." This folder contains finite state model of the module. For a copy of this Finite State Model, please contact JP Mobile.

3.6 Physical Security

As a software product, the Physical Security requirements proposed by FIPS 140-2 are not applicable to SureWave Mobile Defense Security Kernel. The iPAQ 2215 used to test the SureWave Mobile Defense Security Kernel uses production grade components.

3.7 Operational Environment

The SureWave Mobile Defense Security Kernel has been tested using a HP iPAQ 2215 running the Pocket PC 2003 OS Premium (version 4.20.0). This single operator device runs on the 400 MHz PXA255 Intel Xscale processor. This is an ARM-based processor.

3.8 Cryptographic Key Management

Key generation and management mechanisms
The module does not generate cryptographic keys. Keys come outside of the module.

Key Input/Output
Keys are input by the Administrator through the Data Input Interface. Keys can be retrieved only in encrypted form from the module.

Key Storage and Zeroization

The module provides temporary storage for keys that are used by the algorithms. Keys may be retrieved only via the Data Output interface. Keys are zeroized when the module is unloaded or by request from an operator.

CSP	CSP type	Generation	Storage location	Key usage	Key zeroization
AES key	**Symmetric**	**External**	**Process space of the module**	**Encryption/ Decryption**	**WipeAllInternalKeys***
DES key (For Legacy Use Only)	**Symmetric**	**External**	**Process space of the module**	**Encryption/ Decryption**	**WipeAllInternalKeys***

3-DES key	**Symmetric**	**External**	**Process space of the module**	**Encryption/ Decryption**	**WipeAllInternalKeys***
Triple-DES-MAC key	**Message Authentication code**	**External**	**Hard coded in the module**	**Calculation of Triple-DES-MAC for software integrity test**	**Hard reset the iPAQ**

*WipeAllInternalKeys is the method which zeroizes keys.

3.9 Services

Services	**Cryptographic Keys and CSPs**	**Role (CO, User or both)**	**Access (R,W,Z)**
AES Encryption	AES cryptographic key	CO, User	R
AES Decryption	AES cryptographic key	CO, User	R
AES Encryption/Decryption Key Entry	AES cryptographic key	CO	W
AES Encrypted Key Output	AES cryptographic key	CO	R
DES Encryption	DES cryptographic key	CO, User	R
DES Decryption	DES cryptographic key	CO, User	R
DES Encryption/Decryption Key Entry	DES cryptographic key	CO	W
DES Encrypted Key Output	DES cryptographic key	CO	R
3DES Encryption	3DES cryptographic key	CO, User	R
3DES Decryption	3DES cryptographic key	CO, User	R
3DES Encryption/Decryption Key Entry	3DES cryptographic key	CO	W
3DES Encrypted Key Output	3DES cryptographic key	CO	R
Generate SHA-1 hash	N/A	CO, User	
Seed PRNG	N/A	CO, User	
Generate pseudo random bytes, PRNG	N/A	CO, User	
Wipe internal keys	Cryptographic key AES, DES, 3DES	CO, User	Z
Perform self test	N/A	CO, User	

Put the module in FIPS mode	N/A	CO	
Get module status	N/A	CO, User	

Access:

- R – Read
- W – Write
- Z – Zeroize

3.10 EMI/EMC

The iPAQ 2215 that the SureWave Mobile Defense Security Kernel has been tested with for FIPS Validation meets applicable Federal Communication Commission (FCC) Electromagnetic Interference and Electromagnetic Compatibility requirements for business use.

3.11 Self-Tests

Power-up Self Tests
To ensure secure and successful operation, the SureWave Mobile Defense Security Kernel performs the following algorithmic self-tests upon module power-up.

a. Calculates and compares the TDES-MAC 8 bytes value of the module – self integrity check.
b. SHA-1 Known Answer Test
c. AES Known Answer Test
d. 3-DES Known Answer Test
e. DES Known Answer Test (For Legacy Use Only)
f. Approved PRNG (FIPS 186-2, Appendix 3.1) Known Answer Test

Conditional Self Tests
The SureWave Mobile Defense Crypto Kernel performs a continuous random number generator (CRNG) test upon use of the implemented Pseudo Random Number Generator (PRNG) according FIPS 186-2, Appedix 3.1. This test is done by generating 40 bytes on each round and comparing the generated block with previously saved block. The first block is generated during the power-up self-test and its value is used for CRNG testing only. Also, the module implements a CRNG test for the non-Approved PRNG.

3.12 Design Assurance

Configuration Management
To ensure consistent configuration management, version 3.0 of an application known as SourceOffsite is used. Each new build is uniquely identified using the date that the

file was compiled. The date used for this build number is of the format YYMMDD, which corresponds to the Year, Month, and Day of the build.

Secure Installation and Setup
Administrator should place two files *FIPSExtDLL.dll* and *FIPSExtDLL.sig* into folder "\Windows" on target PocketPC device. Required steps to place the module into FIPS mode:

1. Load FIPSExtDLL.DLL;
2. Create instance of TPDACryptoManager object;
3. Perform power-up self test via call of PowerUpTestL method;
4. Call method TPDACryptoManager::SetFIPSMode(TRUE);
5. If method returned TRUE value then FIPS mode was enabled successfully, FALSE indicates error state.

Steps 1, 2, 3 are essential part of module initialization.

Guidance Documents
For more information please see the SureWave Mobile Defense Enterprise User Guide and SureWave Mobile Defense Enterprise Administrator Guide documentation. These guides are available from JP Mobile upon request.

3.13 Mitigation of Other Attacks

No mitigations against attacks occur within the cryptographic boundary of module.

4. TPDACryptoManager methods

4.1 NewL and NewLC

Prototype
static TPDACryptoManager* NewL();
static TPDACryptoManager* NewLC();

Parameters
None.

Return value
Pointer to created object. NULL – fail.

Description

These methods construct and return initialized object of class TPDACryptoManager. Method ConstructL is called for created class object. There is no difference between NewL and NewLC on PocketPC. These methods are different for SymbianOS only. Method NewL cleanup the Symbian object stack, but NewLC does not cleanup the stack.

4.2 EnumBlockCipherProvidersL

Prototype

BOOL EnumBlockCipherProvidersL(CipherType** pBlockCiphersID, int* pNumCiphers);

Parameters

pBlockCiphersID – pointer to address of pointer that receives pointer to allocated array of cipher algorithms ID. This array should be deallocated later using of *delete* operator.

pNumCiphers – pointer to address of pointer that receives number of available cipher algorithms. This array should be deallocated later using of *delete* operator.

Return value

TRUE indicates success, FALSE indicates fail.

Description

This method queries for list of available cipher algorithms. Method returns number of available cipher algorithms and array of cipher IDs. These IDs should be used in methods which expect cipher algorithms ID as parameter.

4.3 GetBlockCipherProviderName

Prototype

BOOL GetBlockCipherProviderName(CipherType cipherID, TString<Character>& sName);

Parameters

cipherID – ID of cipher algorithm. There are several predefined IDs:

- CIPHER_AES
- CIPHER_BLOWFISH
- CIPHER_DES
- CIPHER_3DES

sName – reference to string object which receives cipher algorithm name.

Return value

TRUE indicates success, FALSE indicates fail.

Description
Method allows getting of printable name of specified cipher algorithm ID.

4.4 EnumHashProvidersL

Prototype
BOOL EnumHashProvidersL(HashType** pHashID, int* pNumHash);

Parameters
pHashID – pointer to address of pointer that receives pointer to allocated array of hash algorithms ID. This array should be deallocated later using of *delete* operator.

pNumHash – pointer to address of pointer that receives number of available hash algorithms. This array should be deallocated later using of *delete* operator.

Return value
TRUE indicates success, FALSE indicates fail.

Description
This method queries for list of available hash algorithms. Method returns number of available hash algorithms and array of cipher IDs. These IDs should be used in methods which expect hash algorithms ID as parameter.

4.5 GetHashProviderName

Prototype
BOOL GetHashProviderName(HashType hashID, TString<Character>& sName);

Parameters
hashID – ID of cipher algorithm. There are several predefined IDs:
- HASH_MD5
- HASH_SHA1

sName – reference to string object which receives cipher algorithm name.

Return value
TRUE indicates success, FALSE indicates fail.

Description
Method allows getting of printable name of specified hash algorithm ID.

4.6 GetBlockCipherSupportedKeyLengthL

Prototype
BOOL GetBlockCipherSupportedKeyLengthL(CipherType nCipherType, int** ppKeyArray, int* pKeyNum);

Parameters

nCipherType – ID of cipher algorithm
ppKeyArray – pointer to address of pointer which receives address of allocated array with supported key length in bytes of specified cipher algorithm. This array should be deallocated later using of *delete* operator.

Return value

TRUE indicates success, FALSE indicates fail.

Description

Method allows getting of array of supported key length for specified cipher algorithm. For example actual DES implementation supports 7 bytes key length (key without parity bits) and 8 bytes key (expanded key with parity bits).

4.7 SetCipherScrambledKeyL

Prototype

BOOL SetCipherScrambledKeyL(BYTE* pScrambledMasterKey, size_t nMasterKeyLen, CipherType idDesiredAlg, CipherType idScrambledAlg, BYTE* pScrambleKey, size_t nScrambleKeyLen, BYTE* pIV, size_t nIVSize);

Parameters

pScrambledMasterKey – pointer to cipher key
nMasterKeyLen – length of cipher key
idDesiredAlg – cipher type ID
idScrambledAlg – scramble cipher ID
pScrambleKey – pointer to cipher key of scrambling cipher algorithm
nScrambleKeyLen – length of scramble cipher key
pIV – pointer to initializing vector
nIVSize – length of initialization vector

Return value

TRUE indicates success, FALSE indicates fail.

Description

Methods initializes cipher with specified cipher algorithm ID *idDesiredAlg* with cipher key *pScrambledMasterKey* using of length *nMasterKeyLen*. Method expects the cipher key in encrypted form. It decrypts the cipher key using of cipher algorithm with ID *idScrambledAlg* using of key *pScrambleKey* with length *nScrambleKeyLen*. Master key is encrypted using of CBC mode of scrambling cipher. *pIV* is pointer to initialization vector.

4.8 GetCipherScrambledKeyL

Prototype

BOOL GetCipherScrambledKeyL(BYTE* pScrambledMasterKey, size_t* pMasterKeyLen, CipherType* pIDSelectedAlg, CipherType idScrambledAlg, BYTE* pScrambleKey, size_t nScrambleKeyLen, BYTE* pIV, size_t nIVSize);

Parameters

pScrambledMasterKey – pointer to cipher key
nMasterKeyLen – length of cipher key
pIDSelectedAlg – cipher type ID
idScrambledAlg – scramble cipher ID
pScrambleKey – pointer to cipher key of scrambling cipher algorithm
nScrambleKeyLen – length of scramble cipher key
pIV – pointer to initializing vector
nIVSize – length of initialization vector

Return value

TRUE indicates success, FALSE indicates fail.

Description

Methods retrieves cipher key from actual initialized cipher and encrypts it using of the same or another cipher algorithm. Method retrieves the cipher key in encrypted form. It encryots the cipher key using of cipher algorithm with ID *idScrambledAlg* using of key *pScrambleKey* with length *nScrambleKeyLen*. Master key is encrypted using of CBC mode of scrambling cipher. *pIV* is pointer to initialization vector.

4.9 SetCipherPlainKeyL

Prototype

BOOL SetCipherPlainKeyL(BYTE* pMasterKey, size_t nMasterKeyLen, CipherType idDesiredAlg);

Parameters

pMasterKey – pointer to cipher key
nMasterKeyLen – length of cipher key
idDesiredAlg – cipher type ID

Return value

TRUE indicates success, FALSE indicates fail.

Description

Methods initializes specified cipher using of *idDesiredAlg* and set cipher key in plain form *pMasterKey*.

4.10 IsCipherMasterKeyAssigned

Prototype
BOOL IsCipherMasterKeyAssigned();

Parameters
None.

Return value
TRUE indicates that cipher key is assigned. FALSE indicates that there is no cipher key.

Description
Method returns status of presence of assigned cipher key.

4.11 WipeAllInternalKeys

Prototype
BOOL WipeAllInternalKeys();

Parameters
None.

Return value
TRUE indicates success, FALSE indicates fail.

Description
Method zeroizes and deallocates cipher key and all security relevant values.

4.12 ProcessMemoryBlockL

Prototype
BOOL ProcessMemoryBlockL(BYTE* pInMem, BYTE* pOutMem, size_t nMemLen, CIPHER_DIRECTION eDirection, BOOL bCBCMode, BOOL bFirstBlock, BOOL bLastBlock, BYTE* pIV, size_t nSizeIV);

Parameters
pInMem – pointer to input plain text
pOutMem – pointer to output cipher text buffer
nMemLen – length of input plain text in bytes
eDirection – required operation:

- CIPHER_ENCRYPT
- CIPHER_DECRYPT
  bCBCMode – CBC mode:
- TRUE – using of CBC mode
- FALSE – using of EBC mode

bFirstBlock – indicates is the ciphered block the first (for CBC mode only)

bLastBlock – indicates is the ciphered block the last (for CBC mode only)
pIV – pointer to initialization vector (CBC mode only)
nSizeIV – size of initialization vector

Return value

TRUE indicates success, FALSE indicates fail.

Description

Method performs encryption of specified plain text buffer and places resulting cipher text into output buffer or it decrypts specified cipher text and places resulting plain text into output buffer. Input text buffer may intersect with output text buffer. Method supports ECB or CBC mode. Parameter bCBCMode is used for indication of desired mode. If plain text or cipher text is too large that it may be processed using of sequence of method calls and parameters bFirstBlock and bLastBlock are used to specify is the block first and/or last. There are possible combinations of parameters in CBC mode. If the last block length is less than cipher block length then the last block is

1. bFirstBlock = TRUE and bLastBlock = TRUE
Method processes the only block which is the first block and the last block at the same time.
Initialization vector is initialized using of *pIV* and *nSizeIV* parameters.

2. bFirstBlock =TRUE and bLastBlock = FALSE
Method processes the first block and initialization vector is initialized using of *pIV* and *nSizeIV* parameters.

3. bFirstBlock = FALSE and bLastBlock = TRUE
Method processes the last block. Initialization vector is not changed.

4. bFirstBlock = FALSE and bLastBlock = FALSE
Method processes on of the blocks. Initialization vector is not changed.

4.13 GenerateHashL

Prototype

BOOL GenerateHashL(BYTE* pMemBlock, size_t nMemLen, HashType idHashType, BYTE* pMemHash, BOOL bFirst = TRUE, BOOL bLast = TRUE);

Parameters

pMemBlock – address of input message block
nMemLen – length of input message block
idHashType – id of hash algorithm
pMemHash – address of output message hash buffer
bFirst – TRUE indicates that processed block is the first block, FALSE indicates that processed block is not first.

bLast – TRUE indicates that processed block is the last block, FALSE indicates that processed block is not first.

Return value

TRUE indicates success, FALSE indicates fail.

Description

This method generates message hash digest for specified input message. Actually SHA-1 and MD5 is supported. The method may generate hash for very long message which is splitted on several messages block. In this case parameters *bFirst* and *bLast* should be used. If *bLast* is FALSE that method will not update hash value until *bLast* will become TRUE.

4.14 GetHashLen

Prototype

BOOL GetHashLen(HashType idHashType, size_t* pHashLen);

Parameters

idHashType – id of hash algorithm;
pHashLen – address of variable which receives length of message hash

Return value

TRUE indicates success, FALSE indicates fail.

Description

Method retrieves length of generated hash of selected hash algorithm.

4.15 RandSeed

Prototype

BOOL RandSeed(BYTE* pMemBlock, size_t nMemLen);

Parameters

pMemBlock – address of seed value;
nMemLen – length of seed value

Return value

TRUE indicates success, FALSE indicates fail.

Description

Method seed random generator with specified value. nMemLen should be equal to 20, otherwise method will fail.

4.16 Random

Prototype

BOOL Random(BYTE* pMemBlock, size_t nMemLen);

Parameters

pMemBlock – address of output buffer which should be filled with pseudo random numbers;
nMemLen – length of output buffer in bytes

Return value

TRUE indicates success, FALSE indicates fail.

Description

Method generates specified number of pseudo random bytes and places them into output buffer. The method uses FIPS approved PRNG which is described in FIPS 186-2, appendix 3.1 using of G function SHA-1. The method performs continuous check of generated values – on each cycle of generation of 20 bytes x_j value it compares the value with the previously saved value. Method fails if new value is identical to the previous value.
nMemLen should be multiplication of 40 bytes, otherwise method will fail.

4.17 RandXKey

Prototype

BOOL RandXKey(BYTE* pMemBlock, size_t nMemLen);

Parameters

pMemBlock – address of XKEY value
nMemLen – length of XKEY value in bytes

Return value

TRUE indicates success, FALSE indicates fail.

Description

Method sets XKEY value for PRNG according FIPS 186-2, appendix 3.1. Length of XKEY is arbitrary because methods construct “big number” value from specified binary string. nMemLen should be equal to 20, otherwise method will fail.

4.18 PowerUpTestL

Prototype

BOOL PowerUpTestL(TString<Character>& moduleName, HashType hashID, BYTE* pCheckSig, size_t nSigSize);

Parameters

moduleName – name of checked module
hashID – id of hash algorithm
pCheckSig – address of hash signature
nSigSize – length of hash signature

Return value

TRUE indicates success, FALSE indicates fail.

Description

Method performs initial self integrity check of crypto kernel and performs tests of FIPS approved cipher algorithms (AES, DES, TDES), hash algorithms (SHA-1) and PRNG. Methods calculates hash value of specified module and compares it with etalon hash value. Method fails if calculated hash value and etalon hash value are different. Method fails if one of tested algorithms do not pass test.
If method fails then crypto module blocks all hash and cipher related output functions.
Module blocks all hash and cipher related output functions until power-up test is completed.

4.19 GetPowerTestState

Prototype

FIPS_TEST_STATE GetPowerTestState();

Parameters

None.

Return value

- TEST_OK – test is ok.
- TEST_FAIL – test failed.
- TEST_NOT_DONE – test is in progress or was not preformed yet.

Description

Method returns state of power-up test.

4.20 IsNotBlocked

Prototype

BOOL IsNotBlocked();

Parameters

None.

Return value

TRUE indicates module can perform cipher or hash output methods, FALSE indicates that hash and cipher output methods are disabled.

Description

Method retrieves is crypto module disabled functionality of hash and cipher output methods or not. It checks status of power-up test state and returns appropriate value.

4.21 SetFIPSMode

Prototype

BOOL SetFIPSMode(BOOL bFIPSMode);

Parameters

bFIPSMode – TRUE indicates FIPS mode is on, FALSE indicates FIPS mode is off.

Return value

TRUE indicates success, FALSE indicates fail.

Description

Methods triggers FIPS mode on/off. When FIPS mode is on – all non-FIPS approved algorithms are disabled.

4.22 GetFIPSMode

Prototype

BOOL GetFIPSMode();

Parameters

None.

Return value

State of FIPS mode – TRUE indicates FIPS mode is on, FALSE indicates FIPS mode is off.

Description

Method retrieves state of FIPS mode – TRUE indicates FIPS mode is on, FALSE indicates FIPS mode is off.

4.23 TestBlockCipher

Prototype

BOOL TestBlockCipher(CipherType idCipher);

Parameters

idCipher – id of cipher algorithm

Return value
TRUE indicates success, FALSE indicates fail.

Description
Method performs self test of specified cipher algorithm.

4.24 TestHashProvider

Prototype
BOOL TestHashProvider(HashType idHash);

Parameters
idHash – ID of hash algorithm

Return value
TRUE indicates success, FALSE indicates fail.

Description
Method performs self test of specified hash algorithm.

4.25 TestPRNG

Prototype
BOOL TestPRNG();

Parameters
None.

Return value
TRUE indicates success, FALSE indicates fail.

Description
Method performs self test of PRNG using of Known Answer Test.