SkyRecon Cryptographic Module (SCM)

FIPS 140-2 Documentation: Security Policy

Abstract

This document specifies the security policy for the SkyRecon Cryptographic Module (SCM) as described in FIPS PUB 140-2.

Revision History

Version	Modification Date	Modified By	Description of Changes
1	01/09/2009	Alexandre Buge	Create document

Content

1 Introduction

SkyRecon Cryptographic Module (SCM) is a software-based dynamically linked cryptographic library containing several validated cryptographic algorithms. Software developers can link the SkyRecon Cryptographic Module (SCM) into their applications to provide FIPS 140-2 compliant cryptographic support. For more information about the linkage procedure see appendix A

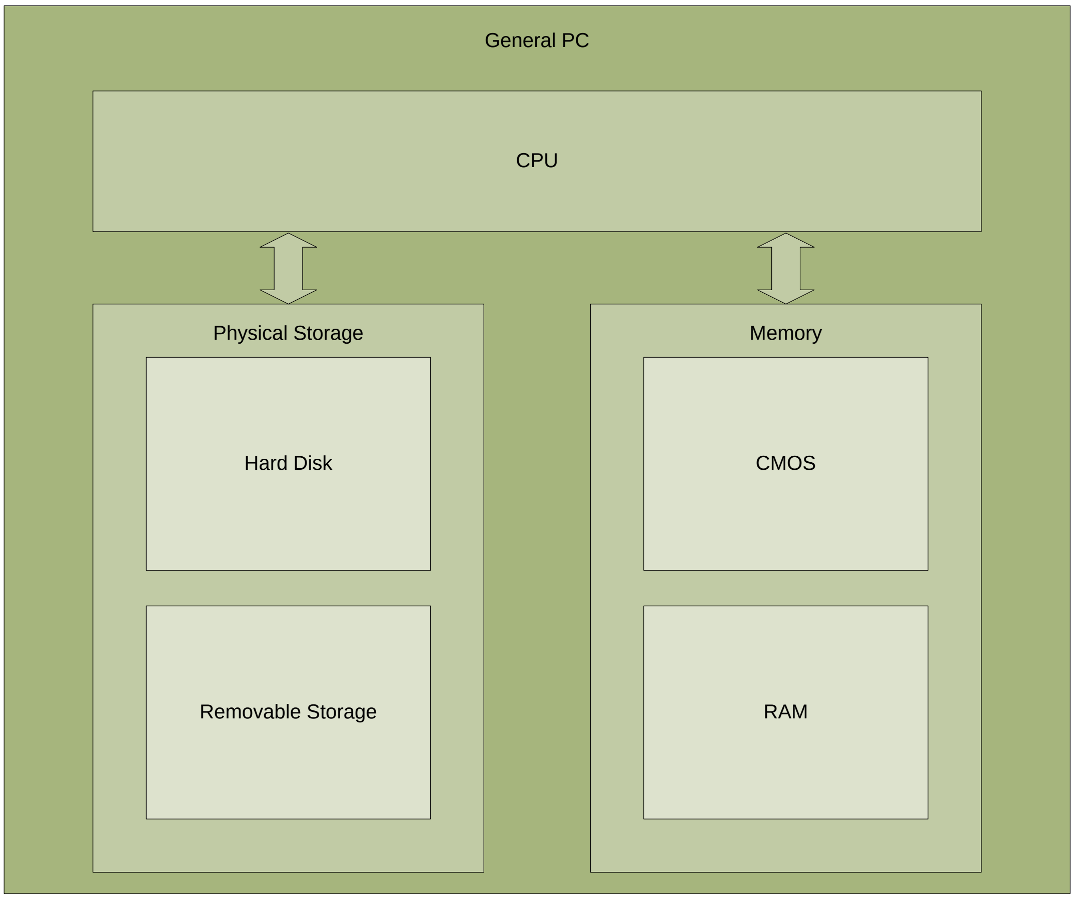
1.1 Cryptographic Boundary

The SkyRecon Cryptographic Module (SCM) consists of a single dynamic link library (.DLL) designed for Windows 32 bits and named libscm.dll (Software version 1.04) tested on x86 processors with Windows XP SP3, which comprises the modules logical boundary. The cryptographic boundary for SCM is defined as the enclosure of the computer system on which the finite state cryptographic module is to be executed. The physical configuration of the module, as defined in FIPS PUB 140-2, is Multi-Chip Standalone. Applications interfacing with the SCM library are outside of the cryptographic boundary.

2 Security Policy

- SCM is supported on Windows XP.
- SCM provides no user authentication. Roles are assumed implicitly. The host process and/or Windows XP OS are responsible for authentication.
- SCM provides only an Approved mode of operation. Therefore, the services of SCM are operating only in FIPS mode.
- All the services provided by SCM are available to the User and Crypto-officer roles.
- Encrypted data, unencrypted data, random values or hashes created within SCM by one user process in Random Access Memory are not accessible to any other user or other process via SCM.
- SCM read keys from the system memory, but relies upon the user process for the encryption of the keys prior to storage.
- SCM contain the following FIPS-approved algorithms: AES-EBC128, AES-EBC192, AES-EBC256 (FIPS Certificate number: #931), SHA256, SHA512 (FIPS Certificate number: #914), ANSI X9.31 A.2.4 AES128, ANSI X9.31 A.2.4 AES192, ANSI X9.31 A.2.4 AES256 (FIPS Certificate number: #532) and HMAC-SHA512 (FIPS Certificate number: #613).
- SCM was tested using the following machine configuration: 32-bits Windows (x86 running on General PC).

The following diagram illustrates the master components of the module



3 Specification of roles

SCM module supports both a User and Cryptographic Officer roles (as defined in FIPS PUB 140-2). Both roles may access all the services implemented in the cryptographic module.
When an application requests the crypto module to generate keys for a user, the keys are generated, used, and deleted as requested by applications. There are no implicit keys associated with a user, and each user may have numerous keys, both signature and key exchange, and these keys are separate from other users' keys.

3.1 Maintenance Roles

Maintenance roles are not supported by SCM.

3.2 Multiple Concurrent Operators

Because the module is a library, each process requesting access is provided its own instance of the module. As such, each process has full access to all information and keys within the module. Note that no keys or other information are maintained upon detachment from the library, thus an instantiation of the module will only contain keys or information that the process has placed in the module. It is the process responsibility to ensure the concurrent access to the manipulated data passed to the SCM functions. In this case only, SCM support multiple concurrent operators.

3.3 Data Access

Because an operator is provided a separate instance of the module, the operator has complete access to all of the security data items within the module.

4 Specification of services

The following list contains all services available to an operator. All services are accessible by all roles.

4.1 Initializing and Self-Tests Services

The following function provides interfaces to the SCM library manual initializing and self-tests launch.

scm_self_test_set_allocator

This function initializes the SCM library and must be called before using all others SCM functions. It computes the first library image signature for software integrity test. This function takes an allocation strategy needed to be used for further self-tests.

Because the library SCM is a dynamic library, the operating system calls implicitly DllMain entry point who calls scm_self_test_set_allocator during the library loading. No explicit initialization need to be performed by the loading process.

The allocation strategy SCM concept is a pair of dynamic memory related functions pointer.
The function pair point to a dynamic memory allocator and the corresponding memory de-allocator.

scm_self_test

This function manually launches a self-test.

scm_self_test_failed

This function informs the caller that last self-tests have passed or failed.
To inspect self-test failure origin, the user application can use scm_sha256_self_test, scm_sha512_self_test, scm_ansi_x9_31_self_test and scm_aes_self_test functions tools.

scm_self_test_get_state

This function returns the status of the SCM library to the caller.

It can return the following values:
scm_state_power_on (1), SCM is loaded into memory and API calls may be made.
scm_state_init (2), SCM initialization functions are performed and the library has not yet run any self-test.
scm_state_self_test (3), SCM is performing self-tests.
scm_state_error (4), SCM is in the error state; all calls to SCM FIPS interfaces return error.
scm_state_operational (5), SCM is in the operational state and all interfaces may be used.

There is also a default state not returned by scm_self_test_get_state :
scm_state_power_off (0), SCM is not linked to any application. It means the library is not loaded into main memory.

4.2 Key Services

The following functions provide interfaces to the SCM's key generation and exchange functions.

scm_aes_key_create

Create a cipher key in internal representation from a sized consecutive binary AES key input.
The internal cipher key storage memory location is specified by an input allocation strategy.

scm_aes_key_destroy

Destroy a cipher key created with scm_aes_key_create by zeroing it and calling the previously defined allocation strategy for freeing memory.

scm_aes_get_size

Get the size of an SCM AES key in byte.

scm_aes_key_get_allocator

Get the allocation strategy associated to a specified SCM AES key.

4.3 Randomization Services

Approved Random Number Generators are used for all Key generation. The following functions provide interfaces to the SCM key generation or zeroing user data.

scm_ansi_x9_31_create

Create an ANSI X9.31 pseudo random number generator context with a specified input allocation strategy, a specified input AES key and a specified timer function callback. After creation, ANSI X9.31 context need to be seeded before using it.

scm_ansi_x9_31_destroy

Destroy an ANSI X9.31 pseudo random number generator context by zeroing it and calling the previously defined de-allocation strategy.

scm_ansi_x9_31_seed

Seed the specified ANSI X9.31 pseudo random number generator context with free user's values. This function returns an error if scm_ansi_x9_31_create has not been called before.

scm_ansi_x9_31_rand

Generate pseudo random number with desired bytes size using the specified ANSI X9.31 context. This call returns an error if scm_ansi_x9_31_create and scm_ansi_x9_31_seed have not been called before.

4.4 Data Encryption and Decryption Services

The following functions provide interfaces to the SCM's data encryption and decryption functions.

scm_aes_get_encrypt_size

Get the needed allocated size to encrypt a buffer with a specified size in byte.

scm_aes_encrypt_block

Encrypt the specified consecutive block in AES-ECB with the specified key. It works on data which size is modulo of the AES block size (128 bit).

scm_aes_decrypt_block

Decrypt the specified AES-ECB consecutive blocks with the specified key. It works on data which size is modulo of the AES block size (128 bit).

scm_aes_encrypt

Encrypt the specified consecutive bytes and randomized padding if needed in AES-ECB with the specified key. Output buffer must have a scm_aes_get_encrypt_size size. This function calls scm_aes_encrypt_block and uses a pseudo random padding to complete data and ensure its size being always a modulo of the AES block size. It allows the caller to encrypt data of any size.

scm_aes_decrypt

Decrypt the specified consecutive bytes in AES-ECB with the specified key. The output size must be given by caller. This function calls scm_aes_decrypt_block.

4.5 Hashing Services

The following functions provide interfaces to the SCM's hashing functions.

scm_sha256

Compute the SHA256 of the specified input buffer.

scm_sha256_init

Initialize a SHA256 context for multiple pass buffers hash.

scm_sha256_update

Update a multiple pass SHA256 context with a specified buffer to hash.

scm_sha256_done

Do the multiple pass SHA256 operation by returning the hash.
The context is zeroized.

scm_sha512

Compute the SHA512 of the specified input buffer.

scm_sha512_init

Initialize a SHA512 context for multiple pass buffers hash.

scm_sha512_update

Update a multiple pass SHA512 context with a specified buffer to hash.

scm_sha512_done

Do the multiple pass SHA512 operation by returning the hash.
The context is zeroized.

The following functions provide interfaces to the SCM's keyed hash functions.
SCM HMAC implementation always uses 1024 bits keys with SHA512 hash algorithm and output 512 bits keyed hash.

scm_hmac_sha512

Compute the HMAC of the specified input buffer.

scm_hmac_sha512_init

Initialize a HMAC context for multiple pass buffers keyed hash.

scm_hmac_sha512_update

Update a multiple pass HMAC context with a specified buffer to get the keyed hash.

scm_hmac_sha512_done

Do the multiple pass HMAC operation by returning the keyed hash.
The context is zeroized.

4.6 Data Input and Output Interfaces

The data input interface for SCM consists of the SCM export functions. Data and options are passed to the interface as input parameters to the SCM export functions. Data Input is kept separate from Control Input by passing Data Input in separate parameters from Control Input.
The Data Output Interface for SCM also consists of the SCM export functions.

4.7 Control Input Interface

The Control Input Interface for SCM also consists of the SCM export functions. Options for control operations are passed as input parameters to the SCM export functions.

4.8 Status Output Interface

The Status Output Interface for SCM also consists of the SCM export functions. For each function, the status information is returned to the caller as the return value from the function.

5 Cryptographic key management

The SCM library manages keys in the following manner.

5.1 Key Material

SCM can create or use keys for the following algorithms: AES128 AES192 and AES256. Each time an application links with SCM, the library is instantiated and no keys exist within. As an exception, keys used for power up self-testing are stored in the cryptographic module. The user application is responsible for importing keys into SCM. The input AES keys format is a contiguous binary buffer with a size depending of the chosen algorithm. The internal cipher key format is in unrolled encryption and decryption key pair for speed considerations.

5.2 Key Generation

Random AES keys can be generated by calling the scm_ansi_x9_31_rand function.

5.3 Key Entry and Output

SCM read keys from the system memory, but relies upon the user process for the encryption of the keys prior to storage.
The input contiguous binary buffer associated to the AES key is directly converted to an internally unrolled cipher key pair format by the scm_aes_key_create function.

5.4 Key Storage

SCM does not provide persistent storage of keys. The task of protecting (or encrypting) the keys prior to storage in the file system is delegated to the user process. The user process is a separate component that is outside the boundaries of the library but relies upon SCM for all cryptographic functionality.

5.5 Key Archival

SCM does not directly archive cryptographic keys. The management of the secure archival of keys is the responsibility of the user process.

5.6 Key Destruction

Internal cipher keys are zeroized from their memory location when the operator calls scm_aes_key_destroy function on the scm_aes_key_create returned value.

6 Self-Tests

SCM provides all of the FIPS 140-2 required self-tests. As required, the module performs some of its self-tests upon power up and other self-tests upon encountering a specific condition (user process running self-test manually).

6.1 Power-up

The following self-tests are initiated upon power-up

- SIT: Software Integrity Test (via a HMAC-SHA512 keyed hash verification of the library file, see 7.3.1 Process Security for details)
- KAT: Known Answer Test

6.2 Conditional

The following is initiated at random number generation

- PST: continuous Pseudo random number generator Stuck Test

6.3 Additional

- CKT: Corrupted Known Answer Test. This test consists in modifying the "known question" input parameter of an algorithm, and ensures that the output is different from the waited "known answer" stored in our known answer test (KAT) code. This test ensures that attackers do not replace our algorithm with a function returning our known answer for each input value and force the KAT success.
- SUT: Successfully Unencrypted Test. This test consists in checking if data encrypted then decrypted with the same key returns the original data. We do this test to ensure that attacker didn't modify our algorithm to corrupt data.
- RTMSIT: Runtime Memory Software Integrity Test (via a HMAC-SHA512 keyed hash verification of the library in random access memory, see 7.3.2 Process Security for details). We perform this test to ensure that an attacker didn’t modify our algorithm at runtime without modifying the .DLL file.

6.4 Details

SMC Algorithm / tests:	SIT	KAT	CKT	SUT	PST	RTMSIT
AES EBC 128	X	X	X	X		X
AES EBC 192	X	X	X	X		X
AES EBC 256	X	X	X	X		X
SHA 256	X	X	X			X
SHA 512	X	X	X			X
HMAC-SHA512	X	X	X			X
ANSI X9.31 A.2.4 AES128	X	X			X	X
ANSI X9.31 A.2.4 AES192	X	X			X	X
ANSI X9.31 A.2.4 AES256	X	X			X	X
Self-tests	X					X

7 Miscellaneous

The following items address requirements not addressed above.

7.1 Cryptographic Bypass

A cryptographic bypass is not supported in SCM.

7.2 Operator Authentication

- No authentication is provided by the SCM module. The host process and/or Windows XP OS are responsible for authentication.

7.3 Process Security

The SCM library is intended to be linked on a process running on Windows XP. The SCM library is not shared between processes.

Each process requesting access is provided its own instance of the module. As such, each process has full access to all information and keys within the module. Note that no keys or other information are maintained upon detachment from the library, thus an instantiation of the module will only contain keys or information that the process has placed in the module.

7.3.1 SIT: Software Integrity Test:

The SCM library performs a software integrity test on its own file. To allow this, SCM embeds a secret 1024 bits HMAC-SHA512 key and a known answer of the HMAC-SHA512 of the libscm.dll file.

During the SCM library build operation, a space is reserved in the libscm.dll file to store its own HMAC-SHA512. This space is referenced by the SCM SIT self test code and exposed by a named exported symbol.

After the library build operation, a "hash tool" is built, using the same secret 1024 bits HMAC-SHA512 key as the SCM library's one. Like the SCM library source code, the "hash tool" is not distributed.

The "hash tool" seeks the libscm.dll named exported symbol and patches it with the correct value.

The value to set is the HMAC-SHA512 of libscm.dll file, without the region containing the HMAC-SHA512 result pointed by the named exported symbol.

At runtime, the SCM SIT, finds its own library file location, opens it and computes the HMAC-SHA512 excluding the named exported symbol. Then the SCM SIT compares the value stored in the library with the dynamically computed HMAC-SHA512.

7.3.2 RTMSIT: Runtime Memory Software Integrity Test:

When a process initializes SCM, the library generates a pseudo random 1024 bits key and computes the HMAC-SHA512 keyed hash on the SCM library pre-compiled code random access memory segment. The segment location and size are obtained by computation of the first and last SCM functions addresses at runtime. This signature is compared to the signature computed during self-tests. Self-tests will only succeed if the two signatures are equal.

8 For more information

For the latest information on SkyRecon Cryptographic Module, check out our World Wide Web site at http://www.skyrecon.com.

Appendix A Installation and Initialization

Note that applications interfacing with the SCM dynamic library are outside of the cryptographic boundary.

To link the application with the SCM library, two way are possible:

- Link statically with SCM library during application builds operation.
- Link dynamically with SCM library by calling Windows API during application runtime.

In each case the DllMain entry point of the SCM library is implicitly called by the operating system in each application process environment. DllMain call scm_self_test_set_allocator. At runtime the scm_self_test_set_allocator function uses the embedded HMAC-SHA512 digest to compute the keyed hash of the memory mapped contents of the SCM library, specifically, the object code (text segment areas of memory as mapped by the runtime linker/loader from the SCM library file). Then selftest is run implicitly by invocation of cryptographic functions or explicitly by invocation of the scm_self_test function call to check the library integrity.

A.1 Static Link with the runtime executable application

When linking the application with the SCM library, user application needs to link with the libscm.dll file.

- For gcc: set -lscm parameter to the linker and launch make command.

- For Visual Studio: create a def and lib file and add libscm.lib to additional dependency of linker input and build the solution.

A.2 Dynamic Link with runtime executable application

An application can call WINAPI LoadLibrary function of the kernel32.dll library in order to load the libscm.dll library into the client process memory address space. The path of the SCM dll file needs to be known by the LoadLibray caller.