F-Secure Corporation

F-Secure Pocket PC Cryptographic Library FIPS 140-2 Validation Security Policy

Author: Alexey Kirichenko

Module version: 1.1

Document version:
F-Secure,FSCLM,FSCLM_PPC_Security_Policy.rtf,00000010

Created: January 2002

Abstract: This document describes the F-Secure Pocket PC Cryptographic Library Security Policy submitted for validation, in accordance with the FIPS publication 140-2, level 1.

Introduction

The F-Secure Pocket PC Cryptographic Library (the Module) is a software module, implemented as a 32-bit Windows? CE compatible DLL for Pocket PC and Pocket PC 2002 platforms. The Module provides an assortment of cryptographic services to any client process that attaches an instance of the Module DLL. The services are accessible for the client through an Application Program Interface (API). This API is documented in the ***F-Secure Cryptographic Library Application Program Interface*** document.

As Pocket PC and Pocket PC 2002 devices are growing in popularity, more and more people use them for creating, storing and processing sensitive information every day. They are small enough to fit in one's pocket and offer an appealing alternative to laptop computers. However, they are also much easier to lose, and if the device is lost or stolen, the information is at risk of being disclosed. To address this and a number of other security-related concerns, data security applications are required on every device. While the use of cryptographic techniques in such applications is often necessary, designing and implementing cryptographic services in a secure, efficient and reliable way is quite challenging. Therefore we believe that the availability of the F-Secure Pocket PC Cryptographic Library, validated to meet security and functional requirements of FIPS 140-2, may bring considerable value to software vendors developing data security products for Pocket PC and Pocket PC 2002 platforms.

It deserves mentioning that we made an effort to design and write the Module's code in such a way that it relies only on generic Windows CE 3.0 services without making use of any functionality specific to Pocket PC and Pocket PC 2002 platforms. Due to this approach, using the Module's code for building cryptographic libraries for other Windows CE-based platforms, such as Smartphone 2002 and Handheld PC 2000, should be very straightforward.

Overall Design and Functionality

The Module is designed and implemented to meet the level 1 requirements of FIPS publication 140-2.

The Module is written in the "C" programming language. The F-Secure Pocket PC Cryptographic Library is a dynamically linked cryptographic library for Pocket PC and Pocket PC 2002 platforms. Our cryptographic libraries are built for a number of different platforms, operating systems, and linkage options. Other examples of our cryptographic library "instances" are: kernel mode export driver and statically linked library for Windows NT/2000/XP; user mode DLL and statically linked library for Windows NT/2000/XP/95/98; DLL for Symbian OS.

Two versions of the Module submitted for FIPS 140-2 validation are the full version and the compact one, with the only difference being between the sets of supported algorithms. The full version supports the FIPS approved AES, DES and Triple DES (TDES), SHA-1 and HMAC-SHA-1 algorithms. It also provides non-FIPS approved Blowfish, MD5, HMAC-MD5, and passphrase-based key derivation (PBKDF2 as specified in PKCS#5) algorithms. The compact version supports the FIPS approved AES, SHA-1 and HMAC-SHA-1 algorithms and provides passphrase-based key derivation (PBKDF2 as specified in PKCS#5) algorithm.
The Module implements a high-quality cryptographically strong Pseudorandom Number Generator (PRNG), which is compliant with the algorithm specified in Section 3.1, Appendix 3 of the **FIPS PUB 186-2** document.

Since the F-Secure Pocket PC Cryptographic Library is a software module that runs on general-purpose Pocket PC and Pocket PC 2002 hand-held devices, no special effort was taken to mitigate power analysis, timing analysis, fault induction and similar attacks.

Use of an appropriate synchronization technique in the Module helps ensure that it functions correctly when simultaneously accessed by multiple threads. We also want to note that performance considerations were an important criterion for the synchronization objects choice.

The Cryptographic Module and Cryptographic Boundary

In FIPS140-2 terms, the Module is a “multi-chip standalone module.” The F-Secure Pocket PC Cryptographic Library runs as a dynamically linked export library in any commercially available Pocket PC or Pocket PC 2002 device under Windows CE 3.0 Operating System (OS). A “cryptographic boundary” for the Module is defined as those applicable software and hardware components internal to a Pocket PC or Pocket PC 2002 device.

The OS separates user processes into memory spaces called “process spaces.” When a client process attaches the Module DLL, the DLL code is mapped to the address space of the process and a copy of the process-specific DDL data is allocated in the client process space. We informally call this mapping and process-specific data *an instance of the Module*. Multiple instances of the Module may reside inside a cryptographic boundary, however such instances are completely independent and each of them belongs to a single process. Any data passed between the Module and its client never leave the client’s process space and, therefore, never leave the cryptographic boundary. The OS is responsible for multi-tasking operations so that only one instance of the Module is active at any particular moment in time.

The module provides no physical security beyond that of the physical enclosure of a “hosting” computer system.

The assumption, which we make about the operating environment of the Module, is that it is installed, initialized and used by following the rules described below in section “Roles and Services.”

The Module was internally tested by the vendor (F-Secure Corporation), on the following computing platforms:

· Pocket PC 2000 (Compaq iPAQ H3630 device),
· Pocket PC 2002 (Compaq iPAQ H3870, HP Jornada 565, Casio E-200 devices)

under the Windows CE 3.0 operating system (in single user mode).

Additionally, the Module was test by a CMVP laboratory on the following computing platform:

· Pocket PC 2000 (Compaq iPAQ H3650 device),

Roles and Services

The F-Secure Pocket PC Cryptographic Library implements the following two roles: Crypto-Officer role and User role. The OS provides functionality to require any user to be successfully authenticated prior to using any system services. However, the Module does not support user identification or authentication that would allow for distinguishing users between the two supported roles. Only a single operator assuming a particular role may operate the Module at any particular moment in time. The OS authentication mechanism must be enabled to ensure that none of the Module's services are available to users who do not assume an authorized role.

The two roles are defined per the FIPS140-2 standard as follows:
A **User** is any entity that can access services implemented in the Module.
A **Crypto-Officer** is any entity that can access services implemented in the Module, install the Module in a device, and configure the device to ensure proper operating of the Module in the FIPS 140-2 mode of operation.
There is no **Maintenance** role.

An operator performing a service within any role can read and write security-relevant data only through the invocation of a service by means of the Module API.

The following operational rules must be followed by any user of the Module.
1. The Module is to be used by a single human operator at a time and may not be actively shared among operators at any period of time.
2. The OS authentication mechanism must be enabled in order to prevent unauthorized users from being able to access system services.

It is the responsibility of the Crypto-Officer to configure the operating system to operate securely and ensure that only a single operator may operate the Module at any particular moment in time.

The services provided by the Module to the User are effectively delivered through the use of appropriate API calls. In this respect, the same set of services is available to both the User and the Crypto-Officer.

When a client process attempts to load an instance of the Module into memory, the Module runs an integrity test and a number of cryptographic functionality self-tests. If all the tests pass successfully, the Module makes a transition to "User Service" state, where the API calls can be used by the client to obtain desired cryptographic services. Otherwise, the Module returns to "Uninitialized" state and the OS reports failure of the attempt to load it into memory.

The Module provides the following FIPS-approved services:
1. Cryptographic data hashing using FIPS PUB 180-1 SHA-1.
2. Symmetric data encryption and decryption using FIPS PUB 197 AES, FIPS PUB 46-2 DES and TDES. (DES and TDES are available only in the full version of the Module.)
3. Random number generation using a software-based algorithm as specified in FIPS 186-2, *Digital Signature Standard (DSS*), Appendix 3.1.

4. MAC computation and verification using FIPS PUB 198 HMAC-SHA-1 algorithm (when key size is at least half of the algorithm output size).

Other services provided by the Module include:
5. Cryptographic data hashing using MD5 algorithm (available only in the full version of the Module).
6. MAC computation and verification using HMAC-MD5 algorithm (available only in the full version of the Module).
7. Symmetric data encryption and decryption using Blowfish block cipher (available only in the full version of the Module).
8. Passphrase-based key derivation (PBKDF2 as specified in PKCS#5) algorithm.

Non-FIPS-approved services cannot be selected if the Module is operating in accordance with FIPS 140-2, that is, in FIPS mode of operation. The exception to this is the Passphrase-based key derivation service, which is based on the FIPS-approved SHA-1 hash function and HMAC-SHA-1 algorithm and provides functionality that is not properly covered by any of the FIPS-approved algorithms at present time.

Key Management

The Module implements a number of functions that are either used internally or exposed in the API to meet the FIPS140-2 Level 1 requirements for Key Management.

Key Generation

Keys for symmetric ciphers and HMAC algorithms can be generated by simply requesting the PRNG implemented in the Module to return a desired number of bytes. The PRNG employs a FIPS-approved algorithm as specified in FIPS 186-2, *Digital Signature Standard (DSS*), Appendix 3.1. No other RNGs are used by the Module.

Intermediate key generation values are never output from the Module.

Key Distribution and Storage

All keys are processed, stored, and used in the Module only on behalf of and for immediate use by its clients, typically, application programs that attach instances of the Module.
Since the current version of the Module does not support any public key methods, there is no easy way to use it for electronic key distribution in the frames of a NIST-approved key distribution protocol or for implementing standard key exchange protocols.
If, nevertheless, someone wants to use the Module API for implementing a key distribution/exchange algorithm, it is their responsibility to ensure FIPS 140-2 compliance of protocols and algorithms they implement.

The Module does not provide long-term cryptographic key storage.

Zeroization of Keys

Keys and critical security parameters in the Module can be divided into two groups: those used by the Module internally and the ones that actually belong to its clients.
The Module takes care of zeroizing all its internal keys and critical security parameters (such as the PRNG internal state): (1) when those are not needed any more, (2) when the client process detaches the instance of the Module, and (3) when the Module enters the error state.
For the other group, when a client requests the Module to destroy a data object containing keys or critical security parameters, the Module always zeroizes all such data objects prior to freeing their memory. Also, the Module performs so-called “objects clean-up at exit.” If the Module’s “DllMain” function is called because the client process is attempting to detach the instance of the Module, we check if there are any objects (like cipher or HMAC contexts) allocated and not freed by any of the client, and we zeroize and free all such objects. This is especially important if a fatal error occurs in the Module, or the client does not have a chance to take proper care of cleaning up objects possibly containing sensitive information.

Protection of Keys

Keys created within or passed into the Module for one user are not accessible to any other user via the Module. It is a responsibility of its clients to protect keys exported from the Module and validate keys passed into the Module.

The Module takes care of never exposing its own internal keys and critical security parameters outside, and of zeroizing those prior to exiting or freeing corresponding portions of memory. In particular, we mention the PRNG state and intermediate generation values, whose disclosure or modification may compromise the security of the Module.

Module Interfaces

Being a software module, the F-Secure Pocket PC Cryptographic Library defines its interfaces in terms of the API that it provides. We define Data Input Interface as all those API calls that accept, as their arguments, data to be used or processed by the Module. The API calls that return, by means of return value or arguments of appropriate types, data generated or otherwise processed by the Module to the caller constitute Data Output Interface. Control Input Interface is comprised of the call used to initiate the Module and the API calls used to control the operation of the Module. Finally, Status Output Interface is defined as the API calls which provide information about the status of the Module.

Self-Testing

The F-Secure Pocket PC Cryptographic Library implements a number of self-tests to check proper functioning of the Module. This includes power-up self-tests (which are also callable on-demand) and conditional self-tests.

Power-up Self-Testing

When an instance of the Module starts loading into memory, power-up self-testing is initiated automatically. It is comprised of the software integrity test and known answer tests of cryptographic algorithms. If any of the tests fail, the Module returns to "Uninitialized" state and the OS reports failure of the attempt to load it into memory.
The following known answer tests are implemented in the Module:
· AES KAT
· DES KAT (runs only in the full version of the Module)
· TDES KAT (runs only in the full version of the Module)
· Blowfish KAT (runs only in the full version of the Module)
· SHA-1 KAT
· HMAC-SHA-1 KAT
· MD5 KAT (runs only in the full version of the Module)

The software integrity test computes DAC value by applying the DES-CBC MAC method, FIPS 113 (in the full version) or the HMAC-SHA-1 method, FIPS 198 (in the compact version) to data of all sections of disk image of the Module, except for "FSC_EDC" data segment. The test fails if the DAC value computed on the disk image of the Module does not match the original value computed on the Module by a special utility at the vendor's site (F-Secure Corporation) and stored in "FSC_EDC" data segment inside the Module.

On-Demand Self-Testing

The Module exports an API routine, "fsclm_Selftest", which can be called to initiate the power-up self-tests plus statistical testing of the PRNG (the four tests defined in the FIPS Publication 140-2). If any of the tests fail, the Module enters the error state. This error state is unrecoverable; upon entering it, the Module stops providing cryptographic services to the client.

Conditional Self-Testing

This includes continuous PRNG testing. The very first output block generated by the PRNG is never used for any purpose other than initiating the continuous PRNG test, which compares every newly generated block with the previously generated block. The test fails if the newly generated PRNG output block matches the previously generated block. In such a case, the Module enters the unrecoverable error state.

List of the API Functions, Operating Modes, Important Technical Considerations

In this section, we briefly describe the services that the Module provides and related security and usage considerations. In order to guarantee secure and robust functioning of the Module, it is important that the clients follow our recommendations as fully and precisely as possible.

The following list presents the Module API functions split into a number of groups in accordance with their functionality.

Mode of operation and Information functions

fsclm_GetModuleVersion
This routine provides the callers with the Module version information.

fsclm_GetModuleMode
This routine returns the current mode of operation of the Module.
The F-Secure Pocket PC Cryptographic Library supports two modes of operation: FIPS 140 mode and non-FIPS mode. Only FIPS-approved algorithms are available to the caller in FIPS 140 mode. Any attempt to use non-FIPS-approved algorithms in FIPS 140 mode results in an appropriate error code returned by the Module. It is a responsibility of client application developers to design their products in such a way that they function properly in the both modes of operation. We recommend avoiding schemes and protocols which are based on non-selectable non-FIPS-approved algorithms in any part.

fsclm_SetModuleMode
This routine sets the mode of operation of the Module. The two options are:
FSCLM_MODE_NONFIPS - all methods included in the Module are available to the caller;
FSCLM_MODE_FIPS140 - only FIPS-approved methods are available to the caller.
Use of "fsclm_SetModuleMode" makes it easy to ensure that non-FIPS-approved algorithms are unavailable, no matter what cryptographic services the client application requests from the Module.

fsclm_GetModuleStatus
This routine returns the current status of the Module. There are five states defined in the Module Finite State Machine (FSM):
FSCLM_STATUS_UNINITIALIZED
FSCLM_STATUS_SELF_TESTING
FSCLM_STATUS_USER_SERVICE
FSCLM_STATUS_UNLOADING
FSCLM_STATUS_ERROR

fsclm_GetErrorCode
This function returns "fatal" error code if the Module is in the error state, or FSCLM_ERROR_FATAL_NONE otherwise.

Symmetric encryption functions

The Module implements a number of symmetric ciphers, including FIPS-approved AES, DES, and TDES modes. In the code, we use a layered approach based on the internal “cipher API”, which makes it very easy to exclude existing or add new ciphers if desired. The cipher modes of operation are implemented as a generic layer, so each newly included cipher can immediately be used in any of the supported modes. (The Module supports the standard ECB, CBC, CFB, and OFB modes as well as Counter and IWEC modes.)

All the encryption and decryption functions support “in-place” operations, which means that the same buffer may be used as both source and destination parameters.

fsclm_CipherInfo
Provides information about the specified cipher. This makes it possible to learn if the cipher is supported by the Module, if it is FIPS-approved, and what key and block sizes are supported for it.

fsclm_CipherAlloc
Allocates and initializes the cipher context object for the specified cipher in the specified mode of operation and with the specified key. Any allocated cipher object must eventually be freed by calling "fsclm_CipherFree". The Module takes care of never exposing contents of cipher objects outside and of proper zeroizing their memory when appropriate.

fsclm_CipherFree
Zeroizes and frees the memory of the specified cipher object. This routine is always available to the caller, even if the Module is in the error state.

fsclm_CipherEncrypt
This encrypts the given input buffer and writes the resulting ciphertext to the given output buffer. Encryption mode and other parameters are taken from the given cipher context object.

fsclm_CipherDecrypt
This decrypts the given input buffer and writes the resulting plaintext to the given output buffer. Mode of operation and other parameters are taken from the given cipher context object.

fsclm_CipherEncryptIV
This encrypts the given input buffer and writes the resulting ciphertext to the given output buffer. The only difference between this routine and "fsclm_CipherEncrypt" is that the latter takes IV/counter information from the cipher object and updates it appropriately, while the former uses "iv" value passed to it as a parameter and updates that value (leaving IV/counter information in the cipher object intact).

fsclm_CipherDecryptIV
This decrypts the given input buffer and writes the resulting plaintext to the given output buffer. The only difference between this routine and "fsclm_CipherDecrypt" is that the latter takes IV/counter information from the cipher object and updates it appropriately, while the former uses "iv" value passed to it as a parameter and updates that value (leaving IV/counter information in the cipher object intact).

fsclm_CipherSetIV

This sets encryption or decryption IV/counter value in the specified cipher object. This value will then be used for the subsequent encryption ("fsclm_CipherEncrypt") or decryption ("fsclm_CipherDecrypt") operation respectively.
Note that the same cipher object can be used for both encryption and decryption operations, thus we maintain separate encryption and decryption IV/counter information in the cipher object.

fsclm_CipherGetIV
This copies the current encryption or decryption IV/counter value in the specified cipher object to the caller-supplied buffer.

fsclm_CipherComputeIV
Certain modes of operation of block ciphers make use of counter value. In such modes, processing of a particular block of input depends on the initial value of counter and index (or offset) of the block. (Two examples supported by the Module are Counter and IWEC modes.) If you want to perform encryption or decryption operation starting with the *n*-th block, you would need to know the corresponding counter value, and this is what this routine helps you do: given the initial counter value and the block index, it computes and writes to the caller-supplied buffer the counter value for the block.
Note that counter-based modes provide you with a random read-write access to large streams of encrypted data, the property that CBC, CFB, and OFB modes do not enjoy.

fsclm_CipherEncryptBuffer
This routine performs one-pass encryption of a given buffer, which can be a useful shortcut in certain cases. It encapsulates a number of other API calls to save the application developer effort. This call is equivalent to the following sequence:
fsclm_CipherAlloc
fsclm_CipherEncryptIV
fsclm_CipherFree

fsclm_CipherDecryptBuffer
This routine performs one-pass decryption of a given buffer, which can be a useful shortcut in certain cases. It encapsulates a number of other API calls to save the application developer effort. This call is equivalent to the following sequence:
fsclm_CipherAlloc
fsclm_CipherDecryptIV
fsclm_CipherFree

Hash functions

The Module currently implements two hash functions: FIPS-approved SHA-1 and non-FIPS-approved MD5. In the code, we use a layered approach based on the internal “hash API”, which makes it very easy to exclude existing or add new hash functions if desired.

fsclm_HashInfo
Provides information about the specified hash function. This makes it possible to learn if the hash function is supported by the Module, if it is FIPS-approved, and what its output (digest) and block sizes are.

fsclm_HashAlloc
Allocates and initializes the hash context object for the specified hash function. Any allocated hash object must eventually be freed by calling "fsclm_HashFree".
Hash objects may contain confidential information. The Module takes care of never exposing contents of hash objects outside and of proper zeroizing their memory when appropriate.

fsclm_HashFree
Zeroizes and frees the memory of the specified hash object. This routine is always available to the caller, even if the Module is in the error state.

fsclm_HashReset
This resets the given hash context object so that it would look like a newly allocated and initialized one. It is useful when you want to use the same hash function for computing hash values (also called *digests*) of multiple data blocks.
The "reset" operation also zeroizes all remnants of the previous processing.

fsclm_HashUpdate
This updates the given hash context with the given input.
When you need to compute digest of a data stream which comes in a number of portions (or when you want to split a very long stream in a number of pieces), you can simply feed such portions to "fsclm_HashUpdate" one by one. The resulting digest value will be identical to what you would get if passing the entire stream as a single buffer.
Note that in order to obtain digest value of your data, any sequence of calls to "fsclm_HashUpdate" must eventually be followed by a call to "fsclm_HashFinal".

fsclm_HashFinal
This function completes computation of hash value of a data stream, which has been processed by calls to "fsclm_HashUpdate" function. The resulting digest is written to a caller-supplied buffer.
Note that after "fsclm_HashFinal" has been called for a hash object, the object should not be used for any further operations until you call "fsclm_HashReset" for it. After resetting, you may start computation of hash value for a new data stream.

fsclm_HashOfBuffer
This routine computes digest of a given buffer, which can be a useful shortcut in certain cases. It encapsulates a number of other API calls to save the application developer effort. This call is equivalent to the following sequence:
fsclm_HashAlloc
fsclm_HashUpdate
fsclm_HashFinal
fsclm_HashFree

HMAC functions

The Module clients can use HMAC methods based on any hash function that is implemented in the Module. By specifying the ID of a hash function of your choice, you fully specify the HMAC

algorithm that you want to use. To obtain information about parameters of a particular HMAC algorithm, simply call "fsclm_HashInfo" for the corresponding hash function.

fsclm_HMACAlloc
Allocates and initializes the context object for the HMAC algorithm based on the specified hash function, and with the specified key. Any allocated HMAC object must eventually be freed by calling "fsclm_HMACFree".
The Module takes care of never exposing contents of HMAC objects outside and of proper zeroizing their memory when appropriate.

fsclm_HMACFree
Zeroizes and frees the memory of the specified HMAC object. This routine is always available to the caller, even if the Module is in the error state.

fsclm_HMACReset
This resets the given HMAC context object so that it would look like a newly allocated and initialized one. It is useful when you want to use the same HMAC function, possibly with a different key, for computing message authentication code (MAC) values of multiple data blocks.
The "reset" operation also zeroizes all remnants of the previous processing.

fsclm_HMACUpdate
This updates the given HMAC context with the given input.
When you need to compute MAC of a data stream which comes in a number of portions (or when you want to split a very long stream in a number of pieces), you can simply feed such portions to "fsclm_HMACUpdate" one by one. The resulting MAC value will be identical to what you would get if passing the entire stream as a single buffer.
Note that in order to obtain MAC value of your data, any sequence of calls to "fsclm_HMACUpdate" must eventually be followed by a call to "fsclm_HMACFinal".

fsclm_HMACFinal
This function completes computation of MAC value of a data stream, which has been processed by calls to "fsclm_HMACUpdate" function. The resulting MAC is written to a caller-supplied buffer.
Note that after "fsclm_HMACFinal" has been called for an HMAC object, the object should not be used for any further operations until you call "fsclm_HMACReset" for it. After resetting, you may start computation of MAC value for a new data stream (possibly using a different key).

fsclm_HMACOfBuffer
This routine computes MAC value of a given buffer, which can be a useful shortcut in certain cases. It encapsulates a number of other API calls to save the application developer effort. This call is equivalent to the following sequence:
fsclm_HMACAlloc
fsclm_HMACUpdate
fsclm_HMACFinal
fsclm_HMACFree

PRNG functions

The PRNG implemented in the Module is based on a hybrid architecture. It uses a one-way output function on top of the well-known "entropy pool" scheme. The design is FIPS-compliant as the output algorithm is the one specified in Section 3.1, Appendix 3 of **FIPS PUB 186-2** document, with the function G constructed from the SHA-1 as specified in Section 3.3, Appendix 3 of the same document.

The PRNG is initialized when the Module gets loaded into memory. During the initialization phase, various system and hardware parameters and statistics are collected and mixed in the PRNG pool with the SHA-1 transform function to achieve a good diffusion of "entropy" bits. Seeding/reseeding code for each supported platform resides in the respective platform-specific source file.

fsclm_PrngDeepPoll
Invokes platform-specific "deep" polling for entropy (i.e., hard-to-predict bits) to achieve good-quality seeding of the PRNG. This deep polling gets called automatically occasionally during the entire lifetime of the Module. However, for Pocket PC and Pocket PC 2002 platforms, this function is not called at the PRNG initialization time (due to the performance considerations). Thus, we recommend our clients calling it on their own prior to requesting the PRNG to generate any critical security values. The main purpose of this function is to help maintain the PRNG pool in a state which is infeasible to guess for the adversary.

fsclm_PrngAddNoise
This exclusive-ORs bytes from the given buffer with the PRNG pool content and serves the purpose of adding unpredictability to the PRNG state. (We leave it up to the client whether to use this function or not as the automatic PRNG seeding in the Module should be good enough to prevent the adversary from guessing the PRNG state or any of the output values.)
The exclusive-OR operation cannot force the PRNG in a weaker state because it obviously cannot reduce the pool data entropy.

fsclm_PrngMixPool
Mixes (i.e., cryptographically processes) the PRNG pool. The mixing operation is based on the SHA-1 transform function. It provides good "entropy" diffusion and is irreversible.
This function gets called automatically at the initialization time and then regularly during the entire lifetime of the Module.

fsclm_PrngGetBytes
This routine writes to the caller-supplied buffer the requested number of PRNG-produced bytes. Although what the generated bytes will be used for is entirely up to the caller, we recommend calling this function if you need to generate:
- any keying material (in both symmetric and asymmetric settings)
- IV or initial counter values used in many popular methods (e.g., modes of operation of block ciphers)
- padding bytes for various cryptographic schemes
- random nonces and challenges required in many cryptographic protocols (e.g., authentication protocols)
- salts to be combined with passphrases in passphrase-based key derivation algorithms
- random values for probabilistic cryptographic algorithms (e.g., signing with DSA)

We stress that it is a responsibility of the client to protect bytes provided by the Module PRNG (in particular, from being exposed to the adversary).

fsclm_PrngGetParameters
Fills in the fields of a caller-supplied structure with the current values of the PRNG object parameters.

fsclm_PrngSetParameters

This function lets the caller change the PRNG parameters used in the algorithms for generating output and updating the PRNG pool.

Other functions

fsclm_Selftest
Calling this routine makes the Module run a number of self-tests. This on-demand self-testing includes self-integrity test, Known Answer Tests of cryptographic algorithms, and the set of PRNG statistical tests (as specified in the FIPS 140-2 document). If any of the tests fail, the Module enters the error state, which means that its cryptographic services become unavailable to the clients. To use the services again, the user will usually need to restart the client application or reload the Module in some other way.

fsclm_DeriveSymmetricKey
This routine implements the passphrase-based key derivation function specified in PKCS#5 (PBKDF2). The implementation uses HMAC-SHA1 as a PRF.
The two main goals of this key derivation algorithm are:
- preventing the adversary from compiling a universal dictionary of passphrases and precomputing the corresponding keys (achieved by using so-called “salt”, whose presence in the algorithm results in a very large number of keys that correspond to each passphrase)
- making exhaustive search attacks much more computationally expensive, which is especially important in the case of “weak” passphrases (achieved by iterating the key derivation function many times and recursively)
We stress that it is a responsibility of the client to protect keys derived by this routine (in particular, from being exposed to the adversary).

fsclm_OverwriteMemory
This function can be used for overwriting a given block of memory with a bit stream that enjoys good statistical properties (i.e., appears as a Binary Symmetric Source output).
We use it internally to overwrite portions of memory that may contain confidential data.
Also, this function can (and should !) be used instead of the PRNG to produce random-looking bits when we do not care about “cryptographic quality”. A typical example is generating “witnesses” for probabilistic primality testing.

Detailed description of the Module API can be found in the ***F-Secure Cryptographic Library Application Program Interface*** document.

We conclude this section by listing a couple of recommendations aimed at helping the Module clients avoid security-related and technical problems when implementing data security products for Pocket PC and Pocket PC 2002 platforms.

- Prior to freeing any memory blocks that may contain critical security parameters or other confidential data, take care of zeroizing them properly.

- It is a responsibility of the clients to ensure they work with cryptographic objects allocated by the Module in a multi-threading safe way. Please keep in mind that the Module provides no synchronisation for accessing such objects concurrently by multiple threads of the client applications.