F-Secure Corporation

F-Secure Cryptographic Service Provider (Windows DLL) FIPS 140-1 Validation Security Policy

Author: Alexey Kirichenko

Module version: 1.1

Date: May 2001

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

Introduction

The F-Secure Cryptographic Service Provider (the Module) is a software module, implemented as a 32-bit Windows™ 'NT/2000/95/98 compatible DLL, which provides an assortment of cryptographic services that are accessible from C and C++ language programs through an Application Program Interface (API). This API is documented in the ***F-Secure Cryptographic Service Provider (Windows DLL) Application Program Interface*** document. To use the services provided by the Module from C and C++ language programs, "SSHCRYPT.H" and, optionally, "SSHMATH.H" header files need to be included in code of such programs to obtain constants, data types, and routine definitions required to call the Module API functions. Preferable way of getting access to the module functionality is linking your application program with Import Library file "SSHCRYPT.LIB".

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

The module supports the FIPS approved DSA, RSA PKCS#1 Digital Signing, DES and TDES Modes, and SHA-1 algorithms. It also provides non-FIPS approved RSA Encryption/Decryption, IDEA, Blowfish, CAST-128, Rijndael (AES), Arcfour, MD5, RIPEMD-160, HMAC-SHA-1, HMAC-MD5, HMAC-RMD160, Diffie-Hellman key agreement algorithms.

The F-Secure Cryptographic Service Provider is designed and written in the C language, with some small performance-critical sections being written in assembly language. (Assembly language portions include certain modular arithmetic operations with multi-precision integers and core transformation functions of certain symmetric ciphers and hash functions.) The Module is identical, at the source code level, for all identified hardware platforms and operating systems.

The cryptographic module and cryptographic boundary

In FIPS140-1 terms, the Module is a “multi-chip standalone module.” The F-Secure Cryptographic Service Provider runs as a DLL under Windows™ ‘NT Operating System (OS) installed in a commercially available IBM Compatible PC. In terms of FIPS 140-1, a “secure cryptographic boundary” is defined as those applicable software and hardware components internal to a host IBM-compatible PC that is running the Windows™ ‘NT Operating System (OS).
The Windows™ ‘NT OS separates user processes into protected memory spaces called “process spaces.” Any process space belongs to a single user and can not be shared with any other user. The OS and the underlying central processing unit (CPU) hardware control access to each process space in such a way that other users cannot write to or read from the process’ memory. Every instance of the F-Secure Cryptographic Service Provider resides in a particular “process space”, namely the space of a process that attached the instance. Multiple instances of the Module may reside inside a cryptographic boundary, however such instances operate completely independently and each of them belongs to a single process. One process can not attach more than one instance of the Module. 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 assumption which we make about the operating environment of the Module is that it is installed and initialized by following the rules described below in section “Roles and Services.”

The cryptographic module was internally tested on the following hardware computing platform and operating system:
A Dell OptiPlex GX1 Personal Computer system with:
· an Intel Pentium III 450 MHz processor,
· 128 MB system RAM (DIMM),
· 2 serial ports and 1 parallel port,
· 4.3 GB hard drive,
· the Windows NT 4.0 Workstation Operating System, Service Pack 6a (in single user mode).

Roles and Services

The F-Secure Cryptographic Service Provider implements the following two roles: Crypto-Officer role and User role. The Module does not support user identification or authentication for these roles. Only a single operator assuming a particular role may operate the Module at any particular moment in time.
The two roles are defined per the FIPS140-1 standard as follows:
A **User** is any entity that can operate a client process (typically, an application program) that uses the Module API.
A **Crypto-Officer** is any entity that can operate a client process that uses the Module API, install the Module in a computer system, and configure the computer system to ensure proper operating of the Module in the FIPS 140-1 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 cryptographic 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. All public keys (effectively, DSA and RSA public keys) entered into the module must be verified as being legitimate and belonging to the correct entity by code of a process which attached the module instance in question.
3. Virtual memory that exists in the computer system running the module must be configured to reside on a local drive, not a network one.

It is a responsibility of the Crypto-Officer to configure the operating system to operate securely and prevent remote login. The following action is required from the Crypto-Officer:
· disable “Server” and “RunAsService” services in the computer system (unless these are not installed or have already been disabled). To do that, the Crypto-Officer needs to run “Services” applet in the Control Panel of the computer system, select the services to be disabled from the list, one by one, push “Startup” button and select “Disabled” as a “Startup Type” in the dialog box. Note that the Crypto-Officer must have administrative privileges in the computer system being configured.

It is also recommended that the Crypto-Officer sets value of “ClearPageFileAtShutdown” to 1 under “HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Control\Session Manager\Memory Management” key and sets “Interactive:Read” ACL for “HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\Perflib” key (as opposed to “Everyone:Read” ACL) in the Registry.

Viewing the User as an application program process that attaches the Module DLL, the services provided by the Module 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 an application program process attempts to attach an instance of the Module DLL, integrity test and a number of cryptographic functionality self-tests are run by the Module. If all the tests pass successfully, the Module makes a transition to "User Service" state, where the API calls can be used by the application program to obtain needed cryptographic services. Otherwise, the Module enters the error state and the calling process will have to terminate.

The following FIPS-approved services are provided by the F-Secure Cryptographic Service Provider:
1. Cryptographic data hashing using FIPS PUB 180-1 SHA-1.
2. Symmetric data encryption and decryption using FIPS PUB 46-2 DES and TDES.
3. Signing and signature verification using FIPS PUB 186-1 DSA and RSA PKCS#1.
4. Random number generation using a software-based algorithm as specified in FIPS 186-2, *Digital Signature Standard (DSS*), Appendix 3.1.

Other services provided by the Module include:
5. Cryptographic data hashing using MD5 and RIPEMD-160 algorithms and MAC computation and verification using the HMAC SHA-1, HMAC-MD5 and HMAC-RMD160 algorithms.
6. Symmetric data encryption and decryption using Rijndael (AES), CAST-128, IDEA, Blowfish, and stream cipher Arcfour.
7. Key wrapping and unwrapping using RSA.
8. Key agreement using Diffie-Hellman.

Non-FIPS-approved services can not be selected if the Module is operating in accordance with FIPS 140-1, that is, in so-called FIPS mode of operation. The only exception to this is RSA encryption which can be used in the FIPS mode of operation but solely for the purpose of encrypting/decrypting keys of symmetric ciphers. It is a responsibility of the Module clients to ensure that RSA encryption is not used in the FIPS mode of operation for any other purposes.

Key Management

The F-Secure Cryptographic Service Provider API contains a number of functions that can be combined to meet FIPS140-1 Level 1 requirements for Key Management.

Key Generation

The Module provides services for generation of the DSA and RSA key pairs. A FIPS-approved key generation methods are used for key pairs generation.
Keys for symmetric ciphers and HMAC algorithms can be generated by simply requesting the module Pseudo-Random Number Generator (PRNG) to return a desired number of bytes. The PRNG employs FIPS-approved algorithm as specified in FIPS 186-2, *Digital Signature Standard (DSS*), Appendix 3.1.

Key Distribution and Storage

The Module supports import and export of electronic keys in both encrypted and plaintext forms. It should be noted, however, that all keys are processed, stored, and used in the Module only on behalf of and for immediate use by a process, typically, an application program, that attaches an instance of the Module.
The Module can be used for electronic key distribution in the frames of a NIST-approved key distribution protocol and for implementing key exchange protocols. This usually involves symmetric ciphers, RSA encryption/decryption, Diffie-Hellman key agreement, and digital signatures algorithms all of which are provided by the Module.
Implementing key distribution and key exchange mechanisms and protocols requires developing application program code which uses the Module's API calls to achieve desired functionality. At run-time, the process of such an application program attaches an instance of the Module DLL, thus, all keys generated and/or otherwise processed by the Module reside in the same Windows 'NT "process space". This effectively means that the application program process has a full control over all such keys, and it is the application program developers responsibility to ensure FIPS140-1 compliance of protocols and algorithms they implement.

The Module does not provide long-term cryptographic key storage. If an application program makes use of the Module services to implement cryptographic key storage functionality, it is a responsibility of the application program developers to ensure FIPS140-1 compliance of key storing techniques they implement.

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 application program process that attaches an instance of the Module. The Module takes care of zeroizing all its internal keys and critical security parameters (such as the PRNG internal state) when those are not needed any more, when the application program process detaches the

instance of the Module, and when the Module enters an error state. For the other group, when the application program 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.

Protection of Keys

We rely on the Windows ™ 'NT memory management mechanism to ensure that process space of a particular process, including its memory, can not be accessed by any other process. It is a responsibility of application program developers to protect keys exported from the Module and validate keys imported in 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.

Module Interfaces

Being a software module, the F-Secure Cryptographic Service Provider defines its interfaces in terms of the API it provides. We define Data Input Interface as all those API calls that accept, as their arguments, data to be 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 Cryptographic Service Provider 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 the Module starts loading (for the reason that some process attempts to attach it), 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 enters the error state with an appropriate associated error code which can be checked by the calling process. This error state is unrecoverable.
The following known answer tests are implemented in the Module:
· DES KAT
· TDES KAT
· SHA-1 KAT
· HMAC-SHA-1 KAT
· RSA encryption/decryption and signing/verification tests

Note: No DSA KAT is implemented. Instead the pairwise consistency test is performed for each DSA key pair generated. See NIST Implementation Guidance 11.2.

On-Demand Self-Testing
The Module exports an API routine which can be called to initiate power-up self-tests plus statistical testing of the PRNG. Calling this routine results in the following tests being run: Software Integrity test, Known answer test of DES implementation, Known Answer test of TDES implementation, Known Answer test of SHA-1 implementation, Known Answer test of HMAC-SHA-1 implementation, Known Answer test of RSA implementation, Statistical PRNG tests (the four tests defined in the FIPS Publication 140-1).

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 newly generated PRNG output block matches the previously generated block. In such a case, the Module enters the unrecoverable error state.

Pairwise Consistency Self-Testing
The test is run whenever private key is generated or imported by the Module. The private key structure of the Module always contains either the data of the corresponding public key or information sufficient for computing the corresponding public key. Thus, generating or importing private key is equivalent to generating or importing key pair.

Depending on key type, the test generates and verifies digital signatures for a piece of data under private and public keys of the key pair being tested and/or applies encryption and decryption operations under the same key pair. If the test fails for generated key pair, the Module enters the unrecoverable error state. If imported key pair does not pass the test, the corresponding function returns an appropriate error code but the Module does not enter the error state. This is a correct behavior since corruption or inconsistency in imported key pair does not mean malfunction of the Module.

Note: No DSA KAT is implemented. Instead the pairwise consistency test is performed for each DSA key pair generated. See NIST Implementation Guidance 11.2.

Operating Modes and List of the API Functions

The F-Secure Cryptographic Service Provider supports two modes of operation: FIPS 140-1 mode and non-FIPS140-1 mode. The module's API includes calls for selecting a desired mode of operation and for requesting current mode of operation. Only FIPS-approved algorithms are available to the caller in FIPS 140-1 mode. Any attempt to use non-FIPS-approved algorithms in FIPS 140-1 mode results in an appropriate error code returned by the Module. It is a responsibility of application program developers to design their products in such a way that they function properly in the both modes of operation. We recommend to avoid schemes and protocols which are based on non-selectable non-FIPS-approved algorithms in any part.

(Note that RSA encryption can be used in the FIPS mode of operation, but solely for the purpose of encrypting/decrypting keys of symmetric ciphers. It is a responsibility of the Module clients to ensure that RSA encryption is not used in the FIPS mode of operation for any other purposes.)

The Module API is described in the ***F-Secure Cryptographic Service Provider (Windows DLL) Application Program Interface*** document. In general, all the routines exported by the Module can be called in the both modes of operation. However, certain functions, if called with parameters requesting non-FIPS-approved services in FIPS 140-1 mode, will only return appropriate error code and no services will be delivered. There are also functions whose output depends on the mode of operation at the time of calling.

In the following list, the API functions are divided into two categories: those whose behavior does not depend on the mode of operation in which they are called, and those which may either refuse to provide certain services in FIPS 140-1 mode or produce different results in the two modes.

Operation mode-independent functions

ssh_crypto_status_message
fscl_get_dll_mode
fscl_set_fips_mode
fscl_set_normal_mode
fscl_get_error_code
fscl_get_module_status
on_demand_self_tests
fscl_crypt_mem_free
FSCryptDllGetVersion
FSCLDelayLoadInitialize
ssh_name_list_get_name
ssh_name_list_step_forward
ssh_name_list_intersection
fscl_DeriveSymmetricKey

ssh_random_add_noise

ssh_random_stir
ssh_random_get_byte
fscl_random_get_nbytes
fscl_random_get_xkey_size
fscl_random_set_xkey_size

fscl_verify_dsa_params

ssh_encode_pubkeyblob
ssh_decode_pubkeyblob
ssh_pubkeyblob_type
ssh_decode_pubkeyblob_ssh1_style

ssh_mp_malloc
ssh_mp_free
ssh_mp_init
ssh_mp_clear
ssh_mp_get_ui
ssh_mp_get_size
ssh_mp_set
ssh_mp_set_ui
ssh_mp_set_si
ssh_mp_set_str
ssh_mp_get_str
ssh_mp_init_set
ssh_mp_init_set_ui
ssh_mp_init_set_si
ssh_mp_neg
ssh_mp_cmp
ssh_mp_cmp_ui
ssh_mp_cmp_si
ssh_mp_get_bit
ssh_mp_set_bit
ssh_mp_and
ssh_mp_or
ssh_mp_xor
ssh_mp_abs
ssh_mp_add
ssh_mp_sub
ssh_mp_add_ui
ssh_mp_sub_ui
ssh_mp_mul
ssh_mp_square
ssh_mp_pow
ssh_mp_div
ssh_mp_mod
ssh_mp_mod_2exp
ssh_mp_mod_ui
ssh_mp_get_buf
ssh_mp_set_buf
ssh_mp_gcd
ssh_mp_gcdext
ssh_mp_invert
ssh_mp_is_probable_prime
ssh_mp_mul_ui

ssh_mp_div_ui
ssh_mp_div_2exp
ssh_mp_mul_2exp
ssh_mp_ powm_bsw_mont

Functions whose behavior depends on mode of operation

ssh_hash_get_supported
ssh_hash_supported
ssh_hash_allocate
ssh_hash_free
ssh_hash_reset
ssh_hash_update
ssh_hash_final
ssh_hash_of_buffer
ssh_hash_input_block_size
ssh_hash_digest_length
ssh_name_list_intersection_hash
ssh_hash_asn1_oid
ssh_hash_iso_identifier

ssh_mac_get_supported
ssh_mac_supported
ssh_mac_allocate
ssh_mac_free
ssh_mac_start
ssh_mac_update
ssh_mac_final
ssh_mac_length
ssh_mac_of_buffer
ssh_name_list_intersection_mac
ssh_mac_info_derive_from_hash
ssh_mac_allocate_with_info
ssh_mac_info_free

ssh_cipher_get_supported
ssh_cipher_get_supported_native
ssh_cipher_supported
ssh_cipher_get_native_name
ssh_cipher_allocate
ssh_cipher_allocate_with_passphrase
ssh_cipher_allocate_and_test_weak_keys
ssh_cipher_free
ssh_cipher_get_key_length
ssh_cipher_get_block_length
ssh_cipher_get_iv_length
ssh_cipher_set_iv
ssh_cipher_get_iv
ssh_cipher_transform
ssh_cipher_transform_with_iv
ssh_cipher_get_name
ssh_name_list_intersection_cipher

ssh_name_list_intersection_public_key
ssh_public_key_get_predefined_groups

ssh_public_key_get_supported
ssh_public_key_name
ssh_private_key_name
ssh_private_key_generate
ssh_private_key_select_scheme
ssh_private_key_derive_public_key
ssh_private_key_get_info
ssh_private_key_copy
ssh_private_key_free
ssh_public_key_define
ssh_public_key_select_scheme
ssh_public_key_get_info
ssh_public_key_copy
ssh_public_key_free

ssh_private_key_import
ssh_private_key_import_with_passphrase
ssh_private_key_import_plain
ssh_private_key_export
ssh_private_key_export_with_passphrase
ssh_private_key_export_plain
ssh_public_key_import
ssh_public_key_export
ssh_public_key_export_canonical

ssh_private_key_sign
ssh_private_key_sign_digest
ssh_private_key_derive_signature_hash
ssh_private_key_max_signature_input_len
ssh_private_key_max_signature_output_len
ssh_public_key_verify_signature
ssh_public_key_verify_signature_with_digest
ssh_public_key_derive_signature_hash

ssh_public_key_max_encrypt_input_len
ssh_public_key_max_encrypt_output_len
ssh_public_key_encrypt
ssh_private_key_max_decrypt_input_len
ssh_private_key_max_decrypt_output_len
ssh_private_key_decrypt

ssh_pk_group_key_name
ssh_private_key_derive_pk_group
ssh_public_key_derive_pk_group
ssh_pk_group_generate
ssh_pk_group_free
ssh_pk_group_select_scheme
ssh_pk_group_get_info
ssh_pk_group_export
ssh_pk_group_import

ssh_pk_group_count_randomizers
ssh_pk_group_generate_randomizer
ssh_pk_group_export_randomizers
ssh_pk_group_import_randomizers

ssh_pk_group_diffie_hellman_setup_max_output_length
ssh_pk_group_diffie_hellman_agree_max_output_length
ssh_pk_group_unified_diffie_hellman_agree_max_output_length
ssh_pk_group_diffie_hellman_setup
ssh_pk_group_diffie_hellman_agree
ssh_pk_group_diffie_hellman_secret_free

ssh_private_key_set_size
ssh_private_decrypt
ssh_public_get_n
ssh_encrypt_ssh1_session_key