Pragma Systems Cryptographic Module Version 2.0 FIPS 140-2 Security Policy

Version 2.0
April 11, 2018

Revision History

Date	Author	Notes
12/15/2017	**D. Kulwin**	**Initial Draft**
4/3/2018	**D. Kulwin**	**CMVP Review Changes (5CM1)**
4/11/2018	**D. Kulwin**	**CMVP Review Changes (5CM2)**

Table of Contents

Table of Figures

Table of Tables

1 Module Description

The Pragma Systems Cryptographic Module is a windows dynamically linked library that is intended to provide cryptographic functions to applications implementing the Secure Shell (SSH) Transport Layer (RFC 2453). The module is bound to the Microsoft Windows Cryptographic Primitives Library CMVP Cert. #2937. Additionally, the Code Integrity (ci.dll) module is bound to perform integrity checking (Cert #2935). The Pragma Systems Cryptographic Module and Microsoft Cryptographic Primitives Library together encapsulate several different cryptographic algorithms in an easy-to-use cryptographic module accessible via the Microsoft CNG (Cryptography, Next Generation) API. It can be dynamically linked into applications by software developers to permit the use of general-purpose FIPS 140-2 Level 1 compliant cryptography, though the only algorithms required by SSH are provided. For example, asymmetric key algorithms can perform encrypt/decrypt operations generally, but are not included since there is no need for those operations in SSH. The Pragma Systems Cryptographic Module provides cryptographic services to Pragma Systems Fortress SSH servers, clients, applications and tools. Additionally, Pragma Systems Telemote and SecureFactors products use this cryptographic module.

Both the Pragma Systems Cryptographic Module and the dependent Microsoft Windows Cryptographic Primitives Library are general purpose, software-based cryptographic modules.

For the Pragma Systems Cryptographic Module, the functionality is offered through the PragmaCryptov2.dll. For the dependent module, the Windows Cryptographic Primitives Library functionality is offered through one cryptographic module comprised of two Dynamic Link Library (DLL) files, BCRYPTPRIMITIVES.DLL and NCRYPTSSLP.DLL.

The Pragma Systems Cryptographic Module's Operational Environments (OEs) are:

- Windows 10 Enterprise Anniversary Update (x64) running on a HP Compaq Pro 6305 - AMD A4 without AES-NI and PCLMULQDQ and SSSE 3

- Windows Server 2016 Standard Edition running on a HP Compaq Pro 6305 - AMD A4 without AES-NI and PCLMULQDQ and SSSE 3

The OE includes the validated dependent Binary Executables included in the bound module
- BCRYPTPRIMITIVES.DLL – Version 10.0.14393 for Windows 10 OEs
- NCRYPTSSLP.DLL – Version 10.0.14393 for Windows 10 OEs
- CI.DLL –version 10.0.14393 for Windows 10 OEs (used for integrity checking)

1.1 Validated Platforms

The Pragma Systems Cryptographic Module was tested on:

- Windows 10 Enterprise Anniversary Update (x64) running on a HP Compaq Pro 6305 - AMD A4 without AES-NI and PCLMULQDQ and SSSE 3

- Windows Server 2016 Standard Edition running on a HP Compaq Pro 6305 - AMD A4 without AES-NI and PCLMULQDQ and SSSE 3

The Module is a *cryptographic software application* that operates as a multi-chip standalone cryptographic module. The physical boundary is the hardware platform on which the Module is installed.

The cryptographic module meets the overall requirements applicable to FIPS 140-2 for the specified level.

SECURITY REQUIREMENTS SECTION	LEVEL
Cryptographic Module Specification	1
Module Ports and Interfaces	1
Roles, Services and Authentication	1
Finite State Model	1
Physical Security	N/A

Operational Environment	1
Cryptographic Key Management	1
EMI/EMC	1
Self-Tests	1
Design Assurance	1
Mitigation of Other Attacks	N/A

Table 1 Module Security Level Specification

1.2 Security Policy

This security policy defines all security rules under which the Pragma Systems Cryptographic Module (Module) must operate and enforce. The module operates under several rules that encapsulate its security policy.

1.2.1 Operating System

The following shall be configured for the Pragma Systems Cryptographic Module to operate in the approved mode of operation:

1. The Microsoft Operating System must be configured such that Debug mode is disabled. One way to do this is to run the following command from an Adminstrator Command Prompt:

   ```
   BCDEDIT /debug OFF
   ```

2. The Microsoft Operating System must be configured such that Driver Signing enforcement is enabled. One of the ways to enable this setting is to run the following command from an Administrator Command Prompt:

   ```
   BCDEDIT /set nointegritychecks OFF
   ```

1.2.2 Microsoft Cryptographic Primitives Library (Bound Module Cert. #2937)

The following shall be configured for the bound module for the Pragma Systems Cryptographic Module to operate in the approved mode of operation:

3. The bound module must be configured to operate in the approved mode of operation by setting any one of the following DWORD registry values to 1:

   o \HKLM\System\CurrentControlSet\Control\Lsa\FipsAlgorithmPolicy\Enabled
   o \HKLM\System\CurrentControlSet\Control\Lsa\FipsAlgorithmPolicy

o \HKLM\System\CurrentControlSet\Control\Lsa\FipsAlgorithmPolicy\MDMEnabled
o\HKLM\SYSTEM\CurrentControlSet\Policies\Microsoft\Cryptography\Configuration\SelfTestAlgorithms

Note 1:** ***Changes to the FIPS mode registry setting do not take effect until the Windows OS has been rebooted. The registry security policy settings can be observed with the regedit tool to determine whether the module is in FIPS mode.

Note 2:** ***Instead of editing the registry directly, the FIPS Local/Group Security Policy Flag may be enabled. The Windows operating system provides a group (or local) security policy setting, "System cryptography: Use FIPS compliant algorithms for encryption, hashing, and signing", which when enabled, will in turn enable one of the FIPS mode registry settings listed above. Changes to the FIPS mode security policy setting do not take effect until the Windows OS has been rebooted.

- When properly initialized as per the guidance in this section, the Microsoft Cryptographic Primitives Library will make the determination that it is validated while under the control of the DllMain code block invoked by the OS Loader (as per FIPS 140-2 IG 9.10). When the determination has been made that the module is validated, the module will operate in its FIPS mode of operation. If the module has not been properly initialized as per the guidance in this section, then the module will make the determination that it is not validated.

  More information about the Microsoft Cryptographic Primitives Library's, including the downloadable Security Policy can be obtained from the following link:

  https://csrc.nist.gov/Projects/Cryptographic-Module-Validation-Program/Certificate/2937

1.2.3 Pragma Systems Cryptographic Module

The following shall be configured for the Pragma Systems Cryptographic Module to operate in the approved mode of operation:

- the Pragma Systems Cryptographic Module shall be configured into the approved mode of operation by setting the following registry DWORD value to 1:
  - \HKLM\SOFTWARE\PragmaSystems\PragmaCrypto\FORCE FIPS

1.3 Determining Operating Mode

If the module has been configured according to sections 1.2.1, 1.2.2 and 1.2.3 above, then an operator can determine the operational state via the GetState() api call which will return the following state indicator:

STATE_ENABLED

The following registry flag is used to set the module into approved mode and is also used, in conjuction with the GetState() api call, to determine if the module is operating in approved mode:

\HKLM\SOFTWARE\PragmaSystems\PragmaCrypto\FORCE FIPS

If the above registry DWORD value is set to 1 and if the GetState() api call returns STATE_ENABLED, then the module is operating in approved mode.

The following diagram, Figure 1, illustrates the master components of the Cryptographic Primitives Library module:

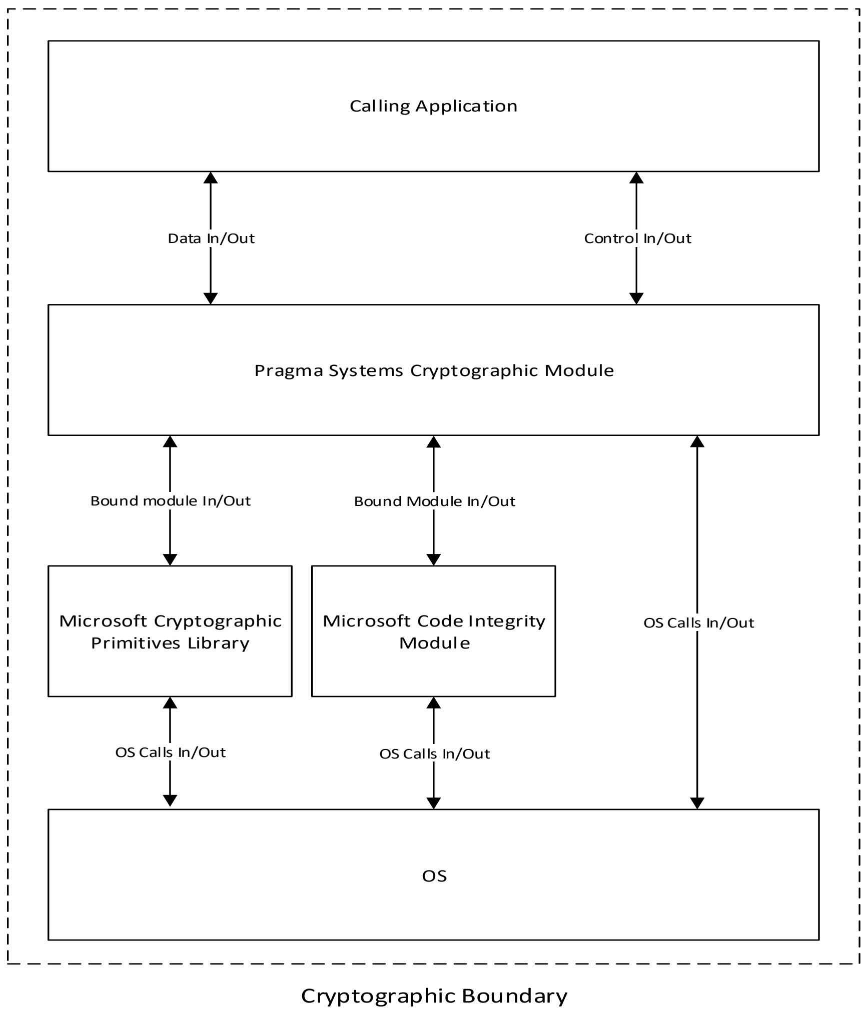

Figure 1 Cryptographic Module Interface Diagram

*NOTE: All of the interfaces (Data IN/OUT, Control IN/OUT) are handled via the Module application interfaces.

1.4 Approved Algorithms

1.4.1 Bound Module Approved algorithms

The Pragma Cryptographic Module relies on the Microsoft Bcryptprimitives contained in the bound module, however only a sub-set of the bound module's cryptography is accessible when bound to the Pragma Systems Cryptographic Module. The table below lists the approved algorithms that support the Pragma Cryptographic Module when they bound together:

The following Approved algorithms are provided by the bound Microsoft Cryptographic Primitives Library module					
Cert. #	**standard**	**Algorithm and Mode/Method**	**Key Lengths, Curves, or Moduli**	**Use**	**Notes**
3347	FIPS 180-4	SHS: SHA-1 SHA-256 SHA-384 SHA-512	NA	Message Digest	NA
2651	FIPS 198-1	HMAC: HMAC-SHA-1 HMAC-SHA-256 HMAC-SHA-384 HMAC-SHA-512	NA	Message Authentication	Prerequisite: SHS #3347
2227	SP 800-67	Triple-DES CBC	3-Key	encrypt/decrypt	CFB64, CFB8, and ECB modes not available when bound to the Pragma Systems Cryptographic Module.
4064	FIPS 197	AES CTR	128 192 256	encrypt/decrypt	AES CBC, CCM, ECB, CFB128, CFB8, CMAC, and XTS not supported when bound to the Pragma Systems Cryptographic Module AES GCM not available in the approved mode when bound with the Pragma Systems Cryptographic Module.
2192 2193	FIPS 186-4	RSA (RSASSA-PKCS1-v1_5 and RSASSA-PSS)	1024 2048 3072	signature generation and verification	1024-bit key size is used for verification only Prerequisites: SHS #3347 DRBG #1217
2195	FIPS 186-4	RSA	2048 3072	key-pair generation	Prerequisites: SHS #3347 DRBG #1217
1098	FIPS 186-4	DSA PQG, SIG	2048 3072	signature generation and verification	Prerequisites: SHS #3347 DRBG #1217
911	FIPS 186-4	ECDSA	P-256 P-384 P-521	KAS	Prerequisites: SHS #3347 DRBG #1217
92	SP 800-56A	ECDH KAS ECC	P-256 P-384 P-521	Key agreement	key establishment methodology provides between 128 and 256-bits of encryption strength

					Prerequisites: SHS #3347, DSA #1098, ECDSA #911, DRBG #1217 HMAC #2651
92	SP 800-56A	DH KAS FFC	2048	Key agreement	key establishment methodology provides at least 112 bits of encryption strength Prerequisites: SHS #3347, DSA #1098, DRBG #1217
1217	SP 800-90A	DRBG AES CTR	256	Derived Random Bit Generation	Prerequisites: AES #4064

Table 2 Bound Module Approved Algorithms

1.4.2 Approved Algorithms not dependent upon bound module

The following approved algorithms are implemented solely within the Pragma Cryptographic Module and do not depend upon the Microsoft Windows Cryptographic Primitives Library:

The following Approved algorithms are provided by the Pragma Systems Cryptographic Library					
Cert. #	**standard**	**Algorithm and mode**	**Moduli/key size**	**Use**	**Notes**
5027	FIPS 197	AES CBC, CTR	128, 192, 256	encrypt/decrypt	NA
3341	FIPS 198-1	HMAC	SHA-1 SHA-256 SHA-384 SHA-512	Keyed Hash Message Authentication	Prerequisites: SHS #4086
4086	FIPS 180-4	SHA	SHA-1 SHA-256 SHA-384 SHA-512	Message digest	NA

Table 3 Pragma CM Approved Algorithms

1.5 Non-Approved/Non-Compliant Algorithms

To operate in FIPS mode, the Operating Environment, Bound Module, and the Pragma Systems shall be configured according to the instructions provided in Section 1.2 above, otherwise the module is operating in the non-approved mode. When configured according to this Security Policy document, the module always operates in the approved mode whereby non-approved algorithms and functions are not accessible.

1.5.1 Bound Module Non-Approved/Non-Compliant algorithms

If the module has not been configured according to this security policy, then the following non-approved algorithms contained in the bound module are available for use:

The following non-approved algorithms are provided by the bound Microsoft Cryptographic Primitives Library module		
Algorithm	**Service**	**Notes**
SHA-1	Message Digest	SHA-1 for digital signature generation: As per SP 800-131-A, SHA-1 may only be used for digital signature generation where specifically allowed by NIST protocol-specific guidance. For all other applications, SHA-1 shall not be used for digital signature generation.
RSA 1024-bits	Signature generation	RSA 1024-bits for digital signature generation RSA Encrypt/Decrypt
DH KAS FFC	Key Agreement	SP 800-56A Key Agreement using Finite Field Cryptography (FFC) with parameter FA (p=1024, q=160), which is disallowed. The key establishment methodology provides 80 bits of encryption strength
AES GCM	Encryption	non-approved algorithm implementation of AES GCM encryption
TDES	Encryption	2-Key Triple-DES
MD5, HMAC-MD5	Hashing, Keyed Hashing	Non-approved
RC2, RC4, MD2, MD4	encrypt/decrypt	Non-approved
DES in ECB, CBC. CFB8 and CFB64 modes	encrypt/decrypt	Non-approved
Legacy CAPI KDF	Key Derivation	Non-approved
IEEE 1619-2007 XTS-AES	encrypt/decrypt	Non-approved

Table 4 Bound Module Non-Approved Algorithms

1.5.2 Pragma Systems Cryptographic Module Non-Approved/Non-Compliant Algorithms

If the module has not been configured according to this security policy, then the following non-approved algorithms contained in the Pragma Systems Cryptographic Module are available for use:

The following non-approved algorithms are provided by the Pragma Systems Cryptographic Module		
Algorithm	**Service**	**Notes**
MD5	Hash function	Non-approved
HMAC-MD5	Message Verification	Non-approved
CHACHA20-POLY1350	Encryption and Decryption	Non-approved
ECDH using Curve 25519	Key Agreement	

Table 5 Pragma CM Non-Approved Algorithms

2 Cryptographic Bypass

Cryptographic bypass is not supported by Cryptographic Primitives Library.

3 Operational Environment

The operational environment for Cryptographic Primitives Library is Windows 10 OEs running on the hardware listed in Section 1.1.

Because the Pragma Cryptographic Module is a DLL, 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 DLL, thus an instantiation of the module will only contain keys or information that the process has placed in the module. The Pragma Cryptographic Module relies on the operating environment to enforce isolation between processes.

4 Ports and Interfaces

The physical ports of the module are provided by the general purpose computer on which the module is installed. The logical interfaces are defined as the API or the cryptographic module. The module's API supports the following logical interfaces:

FIPS 140-2 INTERFACE	LOGICAL INTERFACE
Data Input Interface	Input parameters to all functions that accept input from Crypto-Officer or User entities. For algorithms handled by the bound module, the data may be passed through to the bound module, but this interaction is transparent to the Crypto-Officer or User.
Data Output Interface	Output parameters from all functions that return values from Crypto-Officer or User entities. For algorithms handled by the bound module, the data may be

	passed through from the bound module, but this interaction is transparent to the Crypto-Officer or User.
Control Input Interface	All API functions that are input into the Module by the Crypto-Officer and User entities
Status Output Interface	Information returned via exceptions (return/exit codes) to Crypto-Officer or User entities.

Table 6 Ports and Interfaces

The logical interfaces do not map to the physical ports, and the only entity accessing the logical interfaces is the application that loaded a specific instance of the Module DLL.

The following tables describes the Inputs and Outputs for each api call

HASH

Call	**Inputs**	**Outputs**
HASHInit	Overload1 - Algorithm enum Overload2 - Algorithm enum - Initial Data	Hash context
HASHUpdate	- Hash context - Additional data to hash	Updated hash context
HASHFinal	- Hash context	Final hash digest
HASHFree	- Hash context	none
HASHDigestSize	- Hash algorithm enum	Digest size

Table 7 Hash APIs

HMAC

Call	Inputs	Outputs
HMACSetup	- Algorithm enum	HMAC context
HMACInit	- HMAC context - HMAC key	Updated HMAC context
HMACUpdate	- HMAC context - Additional data to HMAC	Updated HMAC Context
HMACFinal	- HMAC context	Final MAC digest
HMACTearDown	- HMAC context	none

Table 8 HMAC APIs

Cipher

Call	**Inputs**	**Outputs**
CIPHERInit	Overload 1 - None Overload 2 - Algorithm enum - Cipher key blob - Cipher initialization vector	Cipher Context

	- Mode enum	
CIPHEREncrypt	Overload 1 - Cipher context - Data to encrypt Overload 2 - Cipher context - Data to encrypt - Additional auth data - tag HMAC context - HMAC key	Encrypted data
CIPHERDecrypt	Overload 1 - Cipher context - Data to decrypt Overload 2 - Cipher context - Data to decrypt - Additional auth data - tag	Decrypted data
CIPHERSetIV	- Cipher context - Initialization vector	Updated cipher context
CIPHERFree	- Cipher context	none

Table 9 Cipher APIs

Diffie Hellman Key Exchange

Call	Inputs	Outputs
DHInit	- Algorithm enum	Dh context
DHGenerateGroupKey	- Dh context - MODP P,G values	Updated dh context
DHGenerateECCKey	- Dh context - ECC key size (determines curve to use)	Updated dh context
DHGetPublicKey	- Dh context	Public key
DHGetSharedSecret	- Dh context - Remote public key	Shared secret
DHImportECCKey	- Dh context - ECC point parameters - (optional) ECC iterations	Updated dh context
DHImportGroupKey	- Dh context - MODP parameters - Pub key blob - (optional) private key blob	Updated dh context
DHFree	- Dh context	None

Table 10 Diffie-Hellman Key Exchange APIs

Keys

Call	Inputs	Outputs
KEYGen	- Key length - Key type enum	Key pair context
KEYImport	- Key blob	Key pair context
KEYExportPublic	- Key pair context	Public key struct

KEYExportPrivate	- Key pair context	Private key struct
KEYSignData	- Key pair context - Data to sign	signature
KEYVerifySig	- Key pair context - Data over which to verify signature - Signature to verify	Success/fail
KEYFree	- Key pair context	none

Table 11 Key APIs

Utilities

Call	**Inputs**	**Outputs**
GetState	- none	Current State enumeration
GetMode	- none	Current Mode enumeration
SetMode	- Desired Mode enumeration	none
StartSelfCheck	- none	none
MEMFree	- Module allocated memory pointer	none

Table 12 Utility APIs

5 Specification of Roles

The Pragma Cryptographic Module provides User and Cryptographic Officer roles (as defined in FIPS 140-2). These roles share 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. Each user may have numerous keys, and each user's keys are separate from other users' keys.

5.1 Maintenance Roles

Maintenance roles are not supported.

5.2 Multiple Concurrent Interactive Operators

There is only one interactive operator in Single User Mode. When run in this configuration, multiple concurrent interactive operators are not supported. This configuration is set by disabling user switching and disabling remote desktop access.

1. Open the Group Policy Editor by right-clicking the Start menu and selecting Run. Type gpedit.msc in the Run box and click OK.

2. Go to Local Computer Policy >> Computer Configuration >> Administrative Templates >> System >> Logon.

3. On the right side of the window, open “Hide entry points for Fast User Switching” and set the Enabled option. Click the OK button.

4. Open System Properties by right-clicking the Start menu and selecting System. Click the Remote Settings link.

5. Select the “Don’t allow remote connections to this computer” option. Click the OK button.

5.3 Operator Authentication

The module does not provide authentication. Roles are implicitly assumed based on the services that are evoked.

6 Services

All services are accessible to both the User and Crypto Officer roles.

The keys and CSPs saved by an application that loaded the Module DLL are stored outside of the Module memory and are the responsibility of the developer writing the application. The developer is directed to use HASHFree(), CIPHERFree(), HMACTearDown(), DHFree(), and FreeKeys() listed in Table 11 to zeroize and free these saved structures. Inside the Module proper, the keys and CSPs persist only in memory and only for the duration of each API’s execution. If a CSP or key is maintained outside of the Microsoft Bcryptprimitives produced structures but within the Module’s memory, then the memory is zeroed by the Module prior to the key or CSP being destroyed. Zeroization of the keys and CSPs contained in the Microsoft CAPI’s produced structures is performed by the Module via the destruction APIs provided by the FIPS 140-2 validated Microsoft BCrypt APIs’s.

6.1 Critical Security Parameters

CSP Name	Description	How CSP is Established (Service)	Output
RSA SGK	RSA (2048 to 3072 bits) signature generation key	Generated in the bound module (Asymmetric key generation	Can be output from the Pragma Module in plain-text

		service) or imported into the bound module.	
DSA SGK	DSA (2048/3072) Signature generation key	Generated in the bound module (Asymmetric key generation service) or imported into the bound module.	Can be output from the Pragma Module in plain-text
ECDSA SGK	ECDSA (NIST P-256, P-384, P-521) signature generation key	Generated in the bound module (Asymmetric key generation service) or imported into the bound module.	Can be output from the Pragma Module in plain-text
EC DH Private	EC DH (NIST P-256, P-384, P-521) private key agreement key	Generated in the bound module (Key agreement service)	Never output from the Pragma Module
DH Private	DH FFC (2048 to 4096 bit) private key agreement key	Generated in the bound module (Key agreement)	Never output from the Pragma Module
AES EDK	AES (128/192/256 bits) encrypt/decrypt key	Externally generated, imported into the Pragma CM (Key Import Service)	Never output from the Pragma Module
TDES EDK	TDES (3 key – 112 bits) encrypt/decrypt key	Externally generated, imported into the bound module (Key Input Service)	Never output from the Pragma Module
HMAC Key	Keyed hash key (160/256/384/512 bits)	Externally generated, imported into the Pragma CM (Key import service)	Never output from the Pragma Module
DH Shared Secret	Negotiated shared secret FFC 2048-bits – 4096-bits ECC P-256, P-385, P-512	Generated in the bound module (Key Agreement Service)	Output in plain-text from the Pragma Module
DRBG Secret	V and Key secret values.	Exists only in the bound module and supports the bound module's key generation	Never output from the bound module

Table 13 Critical Security Parameters Descriptions

6.2 Services/CSP association

R- read
W- write
D– delete (Zeroization)

E – execute

Service	Roles	Description	CSP and Type of Access
Initialize	User, CO	This service initializes the Pragma Module. Does not access any CSPs	None
Show Status	User, CO	GetState() API call indicates FIPS mode of operation of the Pragma Module with the STATE_ENABLED return indicator. Does not access CSPs	None
Zeroize	User, CO	This will result in zeroization of all Pragma CM and bound module CSPs.	All CSPs (D)
Self Test	User, CO	User callable execution of the Pragma module self-tests using StartSelftests() API. This service does not cause the bound module's self-tests to re-execute which can only be accomplished by unloading and then re-loading of the Pragma CM.	None
Private Key Output	User, CO	Plaintext export of private keys. The Pragma module makes a call to the bound module to export the key, which the Pragma module can then output to the calling application.	RSA SGK (R) DSA SGK (R) ECDSA SGK (R)
Asymmetric key generation	User, CO	The Pragma CM makes a call to the bound module to generate the key. FIPS 186-4 RSA (RSASSA-PKCS1v1_5 and RSASSA-PSS) digital signature generation and verification with 2048 to 16384 modulus; supporting SHA-128, SHA-256, SHA-384, and SHA-512 FIPS 186-4 ECDSA with the following NIST curves: P-256, P384, P-521	RSA SGK (R) DSA SGK (R) ECDSA SGK (R) DRBG Secret (R)
Symmetric encrypt/decrypt	User, CO	If AES encryption/decryption is invoked, the Pragma CM will perform the encryption/decryption functions. If Triple-DES encryption/decryption is invoked the bound module will perform the encryption/decryption functions. Triple-DES with 3 key in CBC mode; AES-128, AES-192, and AES-256 in CBC and CTR modes;	AES EDK (E) TDES EDK (E)
Message Digest	User, CO	Used in the Pragma CM to generate a message digest. Does not access CSPs. FIPS 180-4 SHS SHA-1, SHA-256, SHA-384, and SHA-512;	None

Keyed Hash	User, CO	Used in the Pragma CM to generate or verify data integrity with HMAC. Executes using HMAC Key (passed in by the calling process) FIPS 180-4 SHA-1, SHA-256, SHA384, SHA-512 HMAC;	HMAC Key (E)
Key agreement	User, CO	Used in the bound module to perform key agreement primitives on behalf of the calling process (does not establish keys into the module). Executes using ECDH Private, ECDH Public (passed in by the calling process) SP 800-56A FFC/ECC	EC DH Private (E) DH Private (E) DH shared secret (W)
Key Input	User, CO	Keys imported into either the bound module or the Pragma CM.	DSA SGK (W) ECDSA SGK (W) RSA SGK (W) AES EDK (W) TDES EDK (W) HMAC Key (W)
Digital Signature	User, CO	The bound module performs the signing operations. FIPS 186-4 RSA (RSASSA-PKCS1v1_5 and RSASSA-PSS) digital signature generation and verification with 2048 and 3072 modulus; supporting SHA-1, SHA-256, SHA-384, and SHA-512 FIPS 186-4 DSA SIG(gen/ver) FIPS 186-4 ECDSA with the following NIST curves: P-256, P384, P-521	RSA SGK (R, W) DSA SGK (R, W) ECDSA SGK (R, W)
Utility	User, CO	Miscellaneous helper functions for the Pragma CM. Does not access any CSPs.	None
Random Bit Generation	User, CO	Exists only in the bound module and supports asymmetric and Diffie-Hellman key generation services which are performed by the bound module.	DRBG Secret (WDRX)

Table 14 Service/Algorithms/CSP associations

7 Self Tests

After loading the module the power-up self-tests execute automatically and until completion without any intervention from the operator. The module inhibits data output and access to all cryptographic services while the module is executing the power-up self-tests. The module performs a software integrity check at power-up or on demand by verifying a signed (RSA, 2048-bit key) hash (SHA-256) contained in the module's PKI certificate. Modification of any software component will cause the module to enter the error state with an integrity failure.

7.1 Power-On Self-Tests

The bound Microsoft Cryptographic Primitives Library module performs the following power-up self-tests to verify the correct operation of the algorithms listed in 1.4.1:

1. HMAC Known Answer Tests (includes testing for SHA sizes): HMAC-SHA-1, HMAC-SHA-256, HMAC-SHA-384, and HMAC-SHA-512
2. Triple-DES Known Answer Test
3. AES encrypt/decrypt Known Answer Tests
4. RSA sign/verify Known Answer Tests
5. DSA sign/verify Known Answer Tests
6. ECDSA sign/verify Known Answer Test
7. DH secret agreement Known Answer Test
8. ECDH secret agreement Known Answer Test
9. SP 800-90A AES-256 based counter mode random generator Known Answer Tests (instantiate, generate and reseed)

The Pragma Systems Cryptographic Module performs the following power-up self-tests to verify the correct operation of the algorithms listed in 1.4.2:

1. Module Integrity Check using RSA 2048 signature and SHA-256. The integrity of the module is checked by Code Integrity before it is loaded into memory. This integrity check is based on the verification of an RSA signature over the binary using a 2048-bit key (CAVP Cert. # 2193) and a SHA-256 hash (CAVP Cert. # 3347), and verifying that the signing certificate chains up to a known root authority. The integrity checking is performed by the ci.dll bound module (CMVP Cert # 2935).
2. Bound Module Critical Function Check – ensures the bound module has passed all self-tests
3. HMAC-SHA-1 Known Answer Tests
4. HMAC-SHA-256 Known Answer Tests
5. HMAC-SHA-384 Known Answer Tests
6. HMAC-SHA512 Known Answer Tests
7. SHA-1, SHA-256, SHA-384, and SHA-512 are tested as part of their respective HMAC tests listed above
8. AES-128 encrypt/decrypt CBC Known Answer Tests

9. AES-192 encrypt/decrypt CBC Known Answer Tests
10. AES-256 encrypt/decrypt CBC Known Answer Tests
11. AES-128 encrypt/decrypt CTR Known Answer Tests
12. AES-192 encrypt/decrypt CTR Known Answer Tests
13. AES-256 encrypt/decrypt CTR Known Answer Tests

7.2 Conditional self tests

The bound module performs all of the conditional selftests since all functions that require conditional tests are contained within the boundary of the bound module. The Microsoft Cryptographic Primitives Library Module implements the following applicable conditional for as required by the algorithms in 1.4.1.

- CRNGT for SP 800-90A AES-CTR DRBG
- Pairwise consistency tests for DSA, ECDSA, and RSA key generations
- Pairwise consistency tests for Diffie-Hellman and EC Diffie-Hellman prime value generation
- Assurances for SP 800-56A (According to sections 5.5.2, 5.6.2, and 5.6.3 of the standard)
- DRBG health test for SP 800-90A AES-CT

8 EMI / EMC

The Physical Cryptographic Module component of FIPS 140-2 validated platform is an HP Compaq Pro 6305 GPC which has been tested for and meets applicable Federal Communications Commission (FCC) EMI and EMC requirements for business use as defined in Subpart B of FCC Part.

9 Physical Security

The FIPS 140-2 Area 5 Physical Security requirements are not applicable because the Pragma Systems Cryptographic Module is software only.

10 Mitigation of Other Attacks Policy

The module has not been designed to mitigate any specific attacks outside the scope of FIPS 140-2 requirements.

11 Cryptographic Officer Guidance

In addition to configuring the module according to Section 1.2.3 above, the Module DLL, import library and header files should be protected using OS file permissions so that only authorized personnel can use the files.