Protegrity Cryptographic Module

FIPS 140-2 Non-Proprietary Security Policy

DOCUMENT VERSION – 1.1

DATE – 1/28/2015

Contents

List of figures and tables

1 Introduction

1.1 Summary

This document is a non-proprietary Security Policy for the Protegrity Cryptographic Module. It describes the module and the FIPS 140-2 cryptographic services it provides. This document also defines the FIPS 140-2 security rules for operating the module.

The Protegrity Cryptographic Module is intended to be used in other Protegrity products when compliance with FIPS 140-2 security level 1 requirements is required.

1.2 Purpose of the Security Policy

These are the main reasons why a Security Policy document is required:

- It is required for FIPS 140-2 security level 1 validation.
- It allows individuals and organizations to determine whether the Protegrity Cryptographic Module as implemented satisfies the stated Security Policy.
- It describes the capabilities, protections and access rights provided by the Protegrity Cryptographic Module that will allow individuals and organizations to determine whether it satisfies their security requirements.

1.3 Target Audience

This document will be one of many that are submitted as a package for FIPS validation; it is intended for the following people:

- Developers working on the release.
- The FIPS 140-2 testing lab.
- Cryptographic Module Validation Program (CMVP).
- Consumers.

1.4 Document reference

The enumerated list below shows the documents that are describing the usage guidelines for the Protegrity Cryptographic module.

- Finite state model
- Security Policy
- API Reference

1.5 Naming Conventions used in the documentation

Within the scope of documentation the Protegrity Cryptographic Module can be abbreviated as the “Crypto Module” or “Protegrity module” or “Protegrity Crypto module”.

2 Cryptographic Module Specification

This document is the non-proprietary security policy for the Protegrity Crypto Module and was prepared as part of the requirements for conformance to Federal Information Processing Standard (FIPS) 140-2, Level 1.

The following section describes the module and how it complies with the FIPS 140-2 standard in each of the required areas.

2.1 Description of module

The Protegrity Crypto module is a software only security level 1 cryptographic module that provides general-purpose cryptographic services to any application requiring cryptographic functionality. The Protegrity Crypto module meets the requirements of a multi-chip standalone module.

The module is shipped as a standalone shared library with an Application Programming Interface (API) suitable for the C programming language.

The module's logical boundary is its binary file, which contains the compiled implementation of all supported functionality.

The module contains the following cryptographic functionality:

- Symmetric key encryption and decryption
- Cryptographic hash function

Security Component	**Security Level**
Cryptographic Module specification	1
Cryptographic 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: Security Components and Level

The module has been tested on the following platforms:

Module	OS and Version	Hardware	Processor
Protegrity Crypto Module 1.0	Linux SLES 11 64-bit	IBM x3550 model 7978	Intel x86_64
Protegrity Crypto Module 1.0	Microsoft Windows Server 2008 64-bit	IBM x3550 model 7978	Intel x86_64
Protegrity Crypto Module 1.0	IBM z/OS 2.1	IBM zEC12	IBM zEC12

Table 2: Tested platforms

2.2 Description of modes of operation

The Protegrity Crypto Module supports a FIPS mode of operation (which is the FIPS approved mode) and non-FIPS mode of operation. The Crypto Module turns to FIPS mode of operation after the power-up tests are successfully completed.

In FIPS mode of operation, the Protegrity Crypto Module provides support for the following FIPS-approved functions:

- AES (CBC mode) encryption/decryption
- TDES (CBC mode) encryption/decryption
- SHA-1 (for integrity check only) message digest
- HMAC-SHA1 message authentication code

If the Protegrity Crypto Module is executed in non-FIPS mode of operation the following functions are available:

- DTP2-AES (CBC mode) encryption/decryption
- DTP2-TDES (CBC mode) encryption/decryption
- DTP2-HMAC-SHA1 message authentication code
- CUSP-AES (CBC and ECB mode) encryption/decryption
- CUSP-TDES (CBC and ECB mode) encryption/decryption
- AES (ECB mode) encryption/decryption
- TDES (ECB mode) encryption/decryption
- MD5 message digest
- HMAC-MD5 message authentication code

Note: AES and TDES in ECB mode are non-compliant as they have not been CAVS validated.

See section 10.2 for further instructions on how to set to FIPS mode and non-FIPS mode during the operation of the Protegrity crypto module.

The description of the non-Approved functions DTP2 and CUSP are given below:

- DTP2
  - The DTP2 algorithm is a data type and length preserving symmetric encryption and decryption algorithm.

- The DTP2 algorithm is implemented using regular FIPS approved algorithms, which are:
  - AES (key size 128 and 256),
  - TDES (key size 168), and
  - SHA1.
- The DTP2 algorithm is patented by Protegrity.
- The DTP2 algorithm has not been validated by the CAVP.

- CUSP
  - The CUSP algorithm is a data type and length preserving symmetric encryption and decryption algorithm mainly used in the mainframe (z/OS, OS390) environment.
  - Supported underlying algorithms are: AES key size 128 and 256, TDES key size 168.
  - The CUSP algorithm has not been validated by the CAVP.

No keys or CSP's are shared in between Approved and Non-Approved modes of operations.

2.3 Cryptographic Module Boundary

The module is shipped as a stand-alone shared library with an Application Programming Interface (API) suitable for the C programming language. The logical boundary of the module is the binary code (shared library) of the Protegrity Crypto Module. The binary's name is *pty_crypto.plm*; the same file name is used for all platforms (O/S and hardware) but delivered in different package names depending on each target platform.

Figure 1 shows the boundary of the Protegrity Crypto Module:

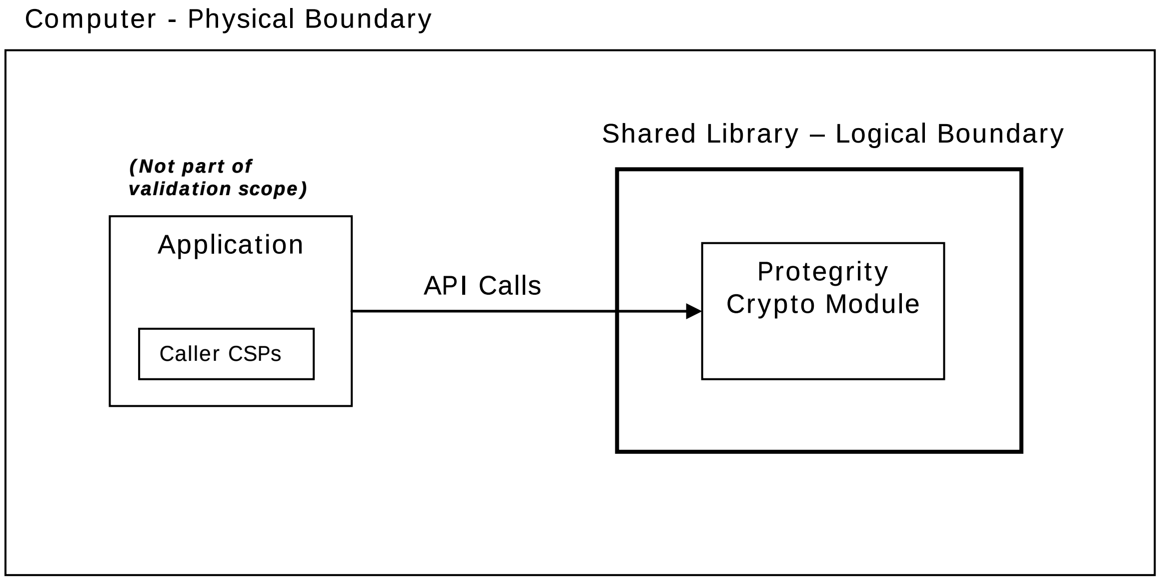
Computer - Physical Boundary
Shared Library – Logical Boundary
(Not part of validation scope)
Application
Caller CSPs
API Calls
Protegrity Crypto Module

FIGURE 1: PHYSICAL BOUNDARY

2.4 Hardware Block Diagram

The physical boundary of the module is the enclosure of the test platform on which the software module executes. Figure 2 shows the physical boundary of the module and hardware components of the platforms on which the module executes.

An image of the Protegrity Crypto Module is stored in persistent storage and loaded into the memory space belonging to the calling application's process when the calling application loads the shared library the cryptographic module.

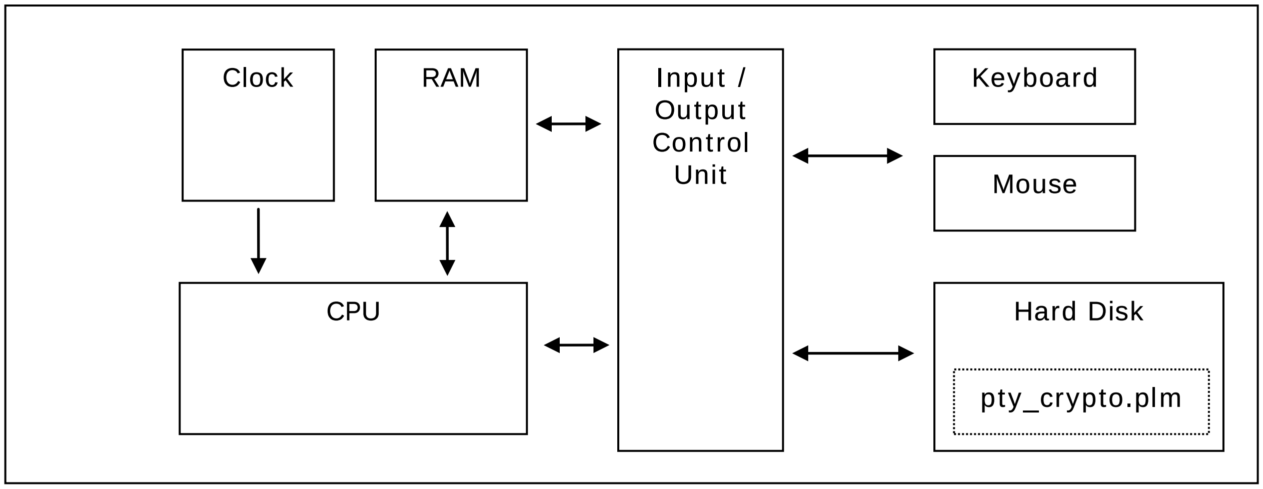

FIGURE 2: HARDWARE BLOCK DIAGRAM

3 Ports and Interfaces

As a software module, it does not have physical ports. Ports and interfaces are interpreted as the module's API, which is a logical interface.

Logical Interface	Description
Data Input	API input parameters
Data Output	API output parameters
Control Input	API function calls
Status Output	API return codes and API return text messages

Table 3: Ports and Interfaces

Control over physical ports is outside of the module's scope since it is a software module.

4 Roles, Services and authentication

4.1 Roles

Based on who is interacting with the module and the nature of the interaction, Crypto Officer Role and User Role are defined for the Module as the following:

- The Crypto-Officer Role is played by the human user who integrates, installs and configures the Module to work with applications. The tasks performed by the Crypto-Officer role is detailed in section 10.2
- User Role is played by the calling application that uses the services provided by the Module as listed in Table 4 in section 4.2.

4.2 Services

Services provided by the Module are requested by the calling application. Cryptographic keys are provided by the calling application as input parameters. Access to internal storage of those keys or any other CSP is not allowed.

Service	**CSP**	**Modes**	**FIPS Approved? If yes, Cert.#**	**Standard**	**API functions**
Service provided via symmetric algorithms					
AES encryption and decryption	128 and 256 bit keys	CBC	#2922, #2923, #2926	FIPS 197	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
TDES encryption and decryption	Key 1, Key 2 and Key 3 (168 bit key size)	CBC	#1735, #1736, #1739	SP 800-67	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
Hash function services					
SHA-1	N/A	N/A	#2458, #2459, #2462	FIPS 180-4	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx

					PTY_DecryptCtx PTY_CleanupCryptoCtx
Message Authentication Code services					
HMAC-SHA1	256-bit HMAC key	N/A	#1849, #1850, #1853	FIPS 198	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
Other security services					
DTP2-AES	128 and 256 bit keys	CBC	No	proprietary	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
DTP2-TDES	Key 1, Key 2 and Key 3 (168 bit key size)	CBC	No	proprietary	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
DTP2-HMAC-SHA1	256-bit HMAC key	N/A	No	proprietary	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
CUSP-AES	128 and 256 bit keys	CBC& ECB	No	proprietary	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx

CUSP-TDES	Key 1, Key 2 and Key 3 (168 bit key size)	CBC & ECB	No	proprietary	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
AES		ECB		FIPS 197	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
TDES		ECB		SP 800-67	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
MD5				RFC1321	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
HMAC-MD5				FIPS 198	PTY_Encrypt PTY_Decrypt PTY_CreateCryptoCtx PTY_EncryptCtx PTY_DecryptCtx PTY_CleanupCryptoCtx
Other non-security services					
Initialization	N/A	N/A			When module is loaded
Set FIPS mode on/off	N/A	N/A			PTY_FIPS_SetMode

Show status	N/A	N/A			PTY_FIPS_GetState PTY_FIPS_GetStateInfo
Get current mode	N/A	N/A			PTY_FIPS_GetMode
Self-test	N/A	N/A			PTY_FIPS_SelfTest
Get Version	N/A	N/A			PTY_FIPS_GetVersion

Table 4: Services

4.3 Operator authentication

There is no operator authentication; assumption of role is implicit by action.

4.4 Mechanism and authentication Strength

Not applicable.

5 Physical security

Due to the fact that this is a software module it does not provide with any physical security.

6 Operational environment

The Module operates in a modifiable operational environment.

6.1 Policy

The operational environment prevents access by other processes to keys and CSPs during the time the Protegrity Crypto Module is running. The Module provides a private context per process for key and CSP storage, which is then destroyed upon request by the process or when the module is powered off (unloaded).

The operating systems segregate user processes into separate process spaces. Each process space is logically separated from all other processes by the operating system software and hardware. The module functions entirely within the process space of the calling application.

The operational environment must be accessed by a single user. No concurrent operators are allowed.

6.2 Operational Rules

The following rules must be followed in order to operate the Protegrity Crypto Module securely:

1. The authentication mechanism provided by the operational environment must be enabled in order to prevent unauthorized users from being able to access system services.
2. The operational environment must prevent unauthorized reading, writing or modification of the Protegrity Crypto Module memory space.
3. Communication between the calling application and the cryptographic module to request services must be performed exclusively through the documented API functions mentioned in section 4.2.
4. It is the responsibility of the calling application to protect keys sent to the cryptographic module.
5. It is the responsibility of the calling application to zeroize the crypto contexts created to request cryptographic services through the corresponding API functions (see section 7.4).
6. Only one user at a time can access the Operational Environment.

7 Cryptographic Key Management

7.1 Key/CSP Generation

The Protegrity Crypto Module neither generates keys in general nor performs key generation for any of its approved algorithms; instead, keys are passed in from clients by way of the module APIs.

7.2 Key Entry and Output

All CSPs enter the Protegrity Crypto Module's logical boundary as cryptographic algorithm API parameters in plaintext. They are associated with memory locations and do not persist across power cycles. The Protegrity Crypto Module does not output intermediate key generation values or other CSPs.

The Protegrity Crypto Module provides a private context per process for key and CSP storage.

7.3 Key Storage

The Protegrity Crypto Module does not provide persistent key storage for keys or CSPs and they also are not stored inside the Protegrity Crypto Module. Instead, pointers to plaintext keys are passed through the Protegrity Crypto Module and keys/CSPs exist only in the volatile memory that is assigned to the process within which the Module runs.

7.4 Key Zeroization

Whenever CSPs are de-allocated, zeroization is done using different kernel memory zeroization APIs, with a value of 0 and a size equal to that of the CSP. The APIs listed in the table below internally call memset() function for performing zeroization.

Table 5 summarizes details regarding what key management the Module provides.

Key/CSP Name	**Details**	**Zeroization**
128 and 256 bit AES keys	Accessible by Roles: User, Crypto Officer Generation: N/A Type: Encrypt and decrypt Entry: API parameter Output: N/A Storage: N/A	Zeroization is done internally by the module when PTY_Encrypt or PTY_Decrypt is called. Zeroization is also done when PTY_CleanupCryptoCtx is called; this requires the functions PTY_CreateCryptoCtx and PTY_EncryptCtx or PTY_DecryptCtx have to be called prior.
TDES 3-Key	Accessible by Roles: User, Crypto Officer Generation: N/A Type: Encrypt and decrypt Entry: API parameter Output: N/A Storage: N/A	Zeroization is done internally by the module when external PTY_Encrypt or PTY_Decrypt is called. Zeroization is also done when PTY_CleanupCryptoCtx is called; this requires the functions PTY_CreateCryptoCtx and PTY_EncryptCtx or PTY_DecryptCtx have to be called prior.

HMAC Keys	Accessible by Roles: User, Crypto Officer Generation: N/A Type: Keyed-Hash Message Authentication Entry: API parameter Output: N/A Storage: N/A	Zeroization is done internally by the module when PTY_Encrypt or PTY_Decrypt is called. Zeroization is also done when PTY_CleanupCryptoCtx is called; this requires the functions PTY_CreateCryptoCtx and PTY_EncryptCtx or PTY_DecryptCtx have to be called prior.
256-bit HMAC Key for module integrity check	Accessible by Roles: Crypto Officer Generation: N/A Type: Keyed-Hash Message Authentication Entry: API parameter Output: N/A Storage: Module Binary	Zeroization not required per FIPS 140-2 IG 7.4

Table 5: Key Zeroization

8 Electro Magnetic Interference/Compatibility

Testing Platform	Product Name / Model	EMI / EMC information
IBM	IBM System x3550 (Type 7978)	Compliant to part 15 of FCC rules, according to "IBM System x3550 type 7978 Installation guide[1]": "This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules."
IBM	zEC12	Compliant to part 15 of FCC rules, according to "zEnterprise EC12 Installation Manual for Physical Planning 2827 All Models[2]": "This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules".

Table 6: EMC/EMI Compatibility

[1] See page 80 in Appendix B of IBM System x3550 type 7978 Installation guide.

[2] See page 146 in Appendix G of zEnterprise EC12 Installation Manual for Physical Planning 2827 All Models (IBM Edition: GC28-6914-01)

9 Self-tests

The Protegrity Crypto Module performs power up self-tests that are executed automatically without requiring any operator intervention when the module gets loaded into the address space of the calling application. While the module is performing the power-up tests no other functions are available and all output is inhibited.

If the power up self-tests fail, the module is set to the ERROR state. If the tests pass, the module is set to FIPS mode and cryptographic services are available. The result of the power-up tests can be obtained by calling the function PTY_FIPS_GetState(), which provides the status indicator (status output interface).

9.1 Integrity test

During the software build process, the Protegrity Crypto Module is used to compute a HMAC-SHA-1 message authentication code (MAC) of the Module binary. The MAC is then stored within the Module, except on the z/OS mainframe platform, where the MAC is stored in a separate file.

During process of loading the Protegrity Crypto Module, the HMAC-SHA-1 MAC of the binary is computed again and compared to the original MAC calculated for the module. If the comparison passes, the Module is loaded and the known-answer tests are run; if all tests pass, the Module enters the FIPS-Approved mode. If any of the known-answer tests fail, error messages are returned to the calling application, the module is set in an ERROR state and no further operations are allowed.

9.2 Known-Answer tests

Once the integrity test passes, the Protegrity Crypto Module performs the following known-answer tests (also referred to cryptographic algorithm tests):

- AES Encryption/Decryption tested separately for CBC mode.
- Triple-DES Encryption/Decryption tested separately for CBC mode.
- HMAC-SHA1
- SHA1

If the values calculated and the known answers do not match then the module will be set in an ERROR state and no further operations are allowed.

9.3 On-Demand Tests

On-Demand tests can be invoked by calling the specific API function for self-test - PTY_FIPS_SelfTest. This will only run the Known-Answer tests. The Integrity test will be executed only during power up self-tests.

While the module is performing the on-demand self-tests no other functions are available and all output is inhibited.

If all Known-Answer tests pass, the function will return a successful code and the module will remain in the current state. If any of the test fails, the function will return an error code and the module will transition to the Error state.

9.4 Error State

The cryptographic module transitions to the ERROR state when the self-tests fail (executed either during power-up tests or requested on-demand). During the ERROR state, all output is inhibited and cryptographic operations are no longer allowed. The only services allowed in the ERROR state are ‘Get Version’, ‘Show Status’ and ‘Get current Mode’. The module needs to be reloaded in order to recover from the ERROR state.

10 Design assurance

10.1 Configuration Management

Protegrity manages and records all source code and documentation by using the Apache Subversion version control system – http://subversion.apache.org/

Access to the Subversion repository is granted or denied by the server administrator based on company and team policy.

10.2 Delivery and Operation

The Crypto module is delivered in a package for each target platform:

- PtyCrypto_Linux_x64_1.0.0.1.tgz
- PtyCrypto_Windows_x64_1.0.0.1.zip
- PtyCrypto_zOS_Mainframe_1.0.0.1.tgz

The Crypto module (pty_crypto.plm) is delivered in binary form as a pre-compiled shared library. The deliverables also include C-header files and the fingerprint calculated for the Crypto module (only for the z/OS mainframe platform).

When a Crypto Officer receives the module delivered in .tgz or .zip file, he shall follow the instructions provided in the Crypto Officer User Guidance to unzip it in an appropriate subdirectory and install the module, then follow the API reference documentation to integrate it into applications.

Version of the Crypto module is provided in the package name. The version of the Crypto module can be also obtained programmatically using the PTY_FIPS_GetVersion() function.

The Crypto module is automatically initialized when it is loaded, whereby a series of self-tests are run on the module. The tests include examining the integrity of the shared library and the correct operation of the cryptographic algorithms. If all tests are successful then the module is set to be in FIPS mode. If any of the tests fail the module will be disabled and no cryptographic services can be utilized. To recover from the error state the module has to be reloaded.

The PTY_FIPS_GetState() function can be used to return the current status of the cryptographic module. The calling application must invoke this function to verify that the Crypto module has been loaded successfully and is fully operational.

The Crypto module can be set to FIPS mode or non-FIPS mode using the PTY_FIPS_SetMode(mode) function (mode=1 for FIPS mode, mode=0 for non-FIPS mode). The PTY_FIPS_GetMode() function can be used to obtain the current mode of the module (FIPS mode, non-FIPS mode).

At any point in time the user can run a self-test by invoking the PTY_FIPS_SelfTest() function. This will test the correct operation of the cryptographic algorithms. If any of the tests fail the module will be disabled and no cryptographic services can be run. To recover from the error state the module has to be reloaded.

11 Mitigation of other attacks

No other attacks are mitigated.

12 Abbreviations

AES	Advanced Encryption Specification
CAVP	Cryptographic Algorithm Validation Program
CBC	Cipher Block Chaining
CUSP	Cryptographic Unit Service Provider mode of encryption (IBM)
CO	Crypto Officer
CSP	Critical Security Parameter
DES	Data Encryption Standard
DTP2	Data Type Preserving encryption version 2
FSM	Finite State Model
HMAC	Hash Message Authentication Code
KAT	Known Answer Test
MAC	Message Authentication Code
NIST	National Institute of Science and Technology
O/S	Operating System
PLM	Protegrity Load Module, i.e. a shared library like an .so file on a Unix environment or .dll in a windows environment
SHA	Secure Hash Algorithm
SHS	Secure Hash Standard
TDES	Triple DES