2013

Security Policy

Tahir Pak Crypto Library

This document provides a non-proprietary ISO/IEC 19790 Security Policy for the Tahir Pak Crypto Library (TPCL). (Software Version 2.1.1)

M/S ACES Pvt. Ltd. Pakistan
6/18/2013

Revision History	
Version	1.0
Author	Indigenous Development Team
Date	25-07-2012

Version	1.1
Date	03-09-2012

Version	1.2
Date	14-09-2012

Version	1.3
Date	24-10-2012

Version	1.4
Date	23-11-2012

Version	1.5
Date	01-01-2013

Version	1.6
Date	06-02-2013

Version	1.7
Date	13-03-2013

Version	1.8
Date	18-04-2013

Version	1.9
Date	30-04-2013

Version	2.0
Date	14-05-2013

Version	2.1
Date	27-05-2013

Version	2.2
Date	18-06-2013

Table of Contents

1. Introduction 6
1.1 Purpose and Scope 6
1.2 Audience 6
1.3 References 7
2. Cryptographic Module Specification 8
2.1 Overview 8
2.2 Module Specification 9
2.2.1 Cryptographic Boundary 9
2.2.2 Tested Platforms 11
2.2.3 Approved Cryptographic Algorithms 11
2.2.4 Non-Approved Cryptographic Algorithms 12
2.2.5 List of Critical Security Parameters and Other Security Related Information 12
3. Cryptographic Module Ports and Interfaces 16
4. Access Control Policy 17
4.1. Roles 17
4.2. Services 19
4.2.1 Show Status 19
4.2.2 Perform Self-Tests 19
4.2.3 Bypass capability 19
4.3. Operator Authentication Mechanism 25
4.4. Strength of Authentication Mechanism 25
5. Physical Security 26
6. Operational Environment 26
6.1 Operational Rules 26
7. Cryptographic Key Management 27
7.1. Random Number Generation 28
7.2. Entropy Input 28
7.3. Key Generation 30
7.4. Key Entry and Output 31
7.5. Key Storage 31

7.6. Zeroization Procedure 31
8. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC) 31
9. Self -Tests 31
9.1. Power-Up Tests 32
9.1.1 Integrity Check 33
9.3. Conditional Tests 34
9.3.1 Pair-wise Consistency Test 34
9.3.2 Continuous Random Number Generator Test for DRBG 34
9.3.3 Health Test for DRBG 35
10. Mitigation of Other Attacks 35

List of Figures

Figure 1: Software Block Diagram of TPCL 10
Figure 2:Structure of Linux Random Number Generator 29
Figure 3: Entropy of /Dev/random device 29

List of Tables

Table 1: Security Levels 9
Table 2: Tested Platforms 11
Table 3: Approved Security Functions 12
Table 4: Critical Security Information 15
Table 5: Ports and Interfaces 17
Table 6: Roles of TPCL 18
Table 7: Services of TPCL 24
Table 8: Authentication 25
Table 9: Symmetric key generation 30
Table 10: EMI/EMC of Machines 31
Table 11: Known Answer Tests 33

1. Introduction

1.1 Purpose and Scope

This document is the ISO/IEC 19790 non-proprietary security policy for the **Tahir Pak Crypto Library (TPCL)** to meet ISO/IEC 19790 for Security level 2 requirements software module.

This Security Policy details the secure operation of the TPCL version 2.1.1 developed by **ACES Pvt. Ltd.** as required in ISO/IEC 19790.

1.2 Audience

This document is required as a part of the ISO/IEC 19790 validation process. It describes the TPCL version 2.1.1 module in relation to ISO/IEC 19790 requirements. The companion document **"TPCL Documentation"** and **"TPCL User Guidance"** describes how to configure and operate with the cryptographic module (CM) in the approved mode. It is a technical reference for developers using and installing the cryptographic module.

1.3 References

[1] FIPS 197 Advanced Encryption Standard,

http://csrc.nist.gov/publications/PubsFIPS.html

[2] SP 800-90A Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)
http://csrc.nist.gov/publications/PubsFIPS.html

[3] FIPS 198 the Keyed-Hash Message Authentication Code (HMAC)
http://csrc.nist.gov/publications/PubsFIPS.html

[4] FIPS 180-4 Secure Hash Standard (SHS)

http://csrc.nist.gov/publications/PubsFIPS.html

[5] FIPS 186-3 Digital Signature Standard (DSS)

http://csrc.nist.gov/publications/PubsFIPS.html

2. Cryptographic Module Specification

2.1 Overview

The Tahir Pak Crypto Library (TPCL) is software based cryptographic module, designed to achieve conformance as per ISO/IEC 19790 standard (security level 2). Purpose of this module is to provide FIPS Approved cryptographic functions to consuming applications via an Application Programming Interface (API).

TPCL is installed on a General Purpose computer (multi chip standalone device) with Red Hat Enterprise Linux (RHEL) 5.3 operating system which is evaluated on EAL4 of CC and PP. TPCL library is programmed in C language and the source code is compiled using “GCC 4.1.2”. TPCL is a shared library. The name of the executable files associated with the module are: **libtpcl.so** and **libgmp.so**.

Security Component	Security level
Cryptographic Module Specification	02
Cryptographic Module Ports and Interfaces	02
Roles, Services and Authentication	02
Finite State Model	02
Physical Security	N/A
Operational Environment	02
Cryptographic Key Management	02
EMI/EMC	03
Self Tests	02

Design Assurance	02
Mitigation of Other Attacks	N/A

Table 1: Security Levels

2.2 Module Specification

2.2.1 Cryptographic Boundary

The cryptographic functions are the software part of TPCL which are as follows:

i. Advanced Encryption Standard, AES (128, 192, 256) use electronic codebook (ECB) and Cipher Block Chaining (CBC) modes
ii. Secure Hash Algorithm, SHA (224, 256, 384, 512)
iii. Hash-based Message Authentication Code, HMAC (224,256,384,512)
iv. Counter-Deterministic Random Bit Generator, CTR-DRBG with AES using derivation function
v. Digital signature algorithm, DSA (Generate public & private keys)
vi. Key Generation Module
vii. Authentication Module
viii. User Management module

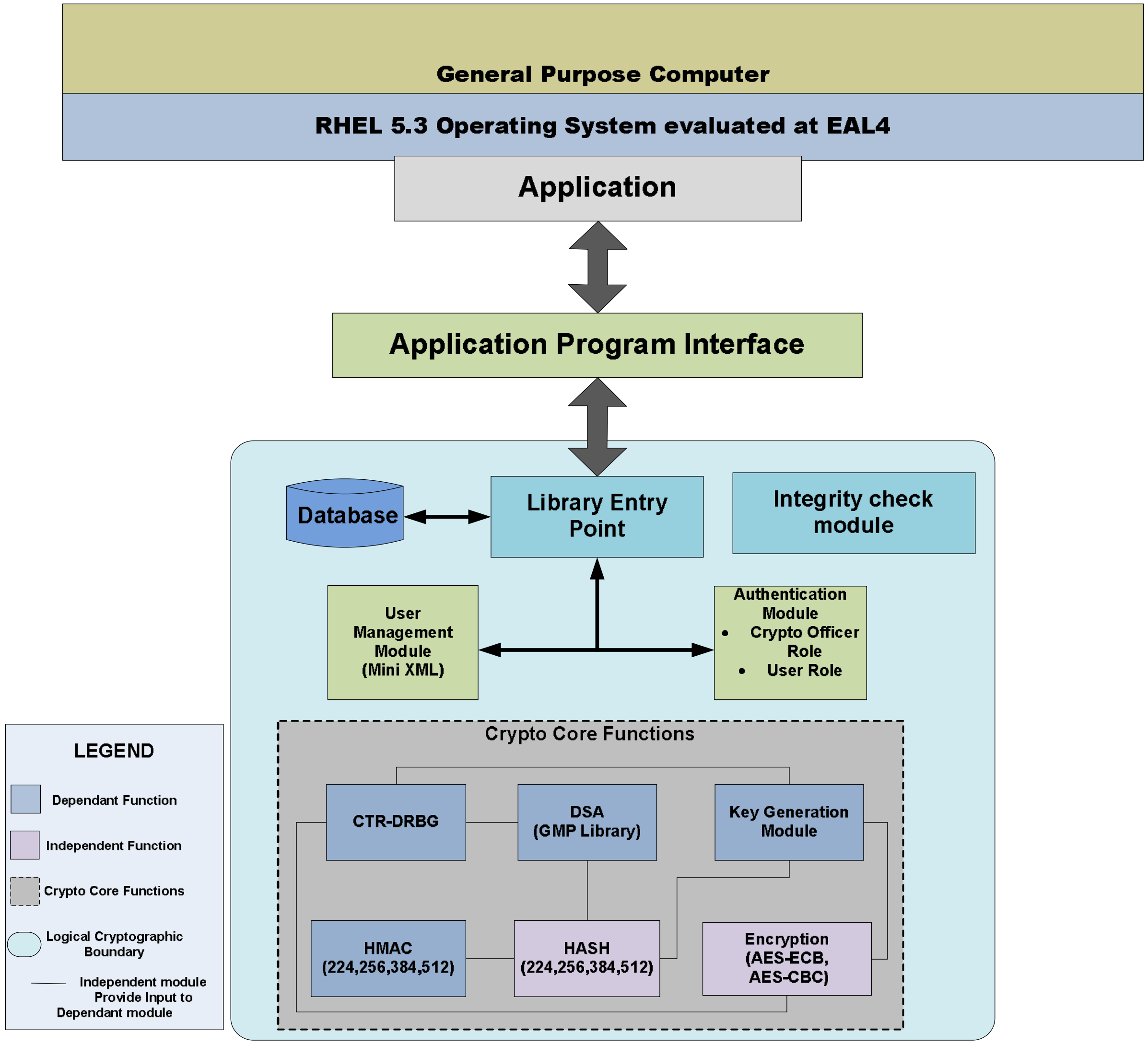

Figure 1: Software Block Diagram of TPCL

2.2.2 Tested Platforms

The TPCL has been tested on the following platforms:

Module/ Implementation	Manufacturer	Model	O/S & Version
DELL power edge 11 generation server	DELL systems	T110 ii	RHEL 5.3

Table 2: Tested Platforms

2.2.3 Approved Cryptographic Algorithms

After passing power-up self tests upon initializing the TPCL module, by default, module operates only in FIPS approved mode of operation. In Approved mode the module supports the following Approved functions:

Algorithm Type	Algorithm	Standard	Use	Certificate #
Symmetric Key	AES -ECB (128,192,256) AES -CBC (128,192,256)	FIPS-197	Data encryption and decryption	2341
Hashing	SHA (224, 256, 384, 512)	FIPS 180-4	Message Digest	2018
Keyed- Hash	HMAC (224,256,384,512)	FIPS 198-1	MAC Calculations	1450
Asymmetric key	DSA (mod 2048 and 3072)	FIPS 186-3	P,Q, G generation/Validation, Key pair generation and Digital signatures calculations /verification	733
Deterministic Random Bit	CTR-DRBG with AES (256) using derivation function	SP 800-90A	Random bit generation	291

Generator (DRBG)				
Key Generation	CTR-DRBG with AES (256) and Hash (512)	SP 800-133	To generate symmetric keys	N/A

Table 3: Approved Security Functions

Note: Validation of each cryptographic algorithm must be obtained from SP 800-131A.

2.2.4 Non-Approved Cryptographic Algorithms

NDRNG: /dev/random. It is used to seed the CTR-DRBG

2.2.5 List of Critical Security Parameters and Other Security Related Information

The following table summarizes the critical information whose disclosure or modification will compromise the security of TPCL.

S. No	CSP/PSP/Keys	Type	Algorithm/ Modes	Generation / input	Output	Storage	Usage	Zeroization
1.	Symmetric Keys (128,192,256)	Symmetric Keys: CSP	AES (ECB,CBC) Key generation method.	Generated externally or internally/ input is in plaintext	Output to the consuming application during key generation	Stored in RAM in plaintext	Encryption/decryption	Erasing from RAM soon after **tpcl_key_zeroize()** API call from consuming application in case of AES and after generation and output to consuming

								application in case of key generation.
2.	**a).** Domain parameters(p, q, g{,first seed, pseed, qseed}) modulus L = 2048, N = 224 L = 3072, N = 256 **b).** 2048 and 3072 bits modulus size key pair	Domain parameters : PSPs, public keys: PSP and private keys: CSP	DSA domain parameter generation: SHA-256 Generation of P and Q through (Provable Primes P and Q), Generation of G through (Canonical generation of G), Key pair(generation), sign(generation /verification)	Generated itself by the module/ input in plaintext	Domain parameters and Private/ Public key pair are output to the consuming application	p, q and g, Public and private key in RAM in plaintext	**a)** Key pair generation **b)** Signature generation and verification **c)** P,Q,G generation/ Validation.	p, q and g, Public / private keys erasing from the RAM after using them in required functions.
3.	Per- message secret number (224, 256) for each (2048,3072) mod respectively.	Per-message secret number: CSPs	DSA	Generated itself by the module using extra random bits method	Never outputs from the module	Stored in RAM in plaintext	Used for digital signing process.	Erasing from the RAM after using for generation of signatures.
4.	Entropy input (3 * requested security strength bits) (security strengths supported are 112, 128 and 256)	Entropy Input: CSP	SP800-90 CTR-DRBG (AES-256)	Generated from the Linux OS through entropy input call	Never outputs from the module	Stored in RAM in plaintext	Random bit generation	Erasing from the RAM after using for instantiation of DRBG.

5.	Seed (384 bits)	Seed: CSP	SP800-90 CTR-DRBG (AES-256)	Generated internally by the module.	Never outputs from the module	Stored in RAM in plaintext	Random bit generation	Erasing from the RAM after updating the internal state of DRBG.
6.	Internal state. key (256 bits)	Internal sate. Key: CSP	SP800-90 CTR-DRBG (AES-256)	Generated itself by the module	Never outputs from the module	Stored in RAM in plaintext	Random bit generation	Erasing from the RAM by calling the un instantiate function.
7.	Internal state. V (128 bits)	Internal state. V: CSP	SP800-90 CTR-DRBG (AES-256)	Generated itself by the module	Never outputs from the module	Stored in RAM in plaintext	Random bit generation	Erasing from the RAM by calling the un instantiate function.
8.	HMAC-224 key (112~448 bits) HMAC~256 key (128~512 bits) HMAC~384 key (192~768 bits) HMAC-512 key (256~1024 bits)	HMAC keys: CSP	HMAC -224 HMAC -256 HMAC -384 HMAC -512 Key generation method.	Generated externally or internally/ input in plaintext	Output to the consuming application during key generation	Stored in RAM in plaintext	Message authentication	Erasing from RAM after using them in case of all HMACs and after generation and output to consuming application in case of key generation.
9.	Global key(symmetric) (64~1024 bits)	Global key: CSP	Key generation method	Generated internally/ never inputs to the module.	Output to the consuming application during key generation	Stored in RAM in plaintext	Used by consuming application as per need.	Erasing from the RAM after generation and output to the consuming application.
10.	Software Integrity key (256 bits)	Software Integrity key: PSP	HMAC -512	Hardcoded into the module	Never outputs from the module	Hard Drive in plaintext	Software integrity testing in Self-test	Uninstalling the module
11.	Operator password (8-10 characters)	Operator password: CSP	Authentication module	Input in plaintext	Never outputs from the module	A hash of each user passwor	User authentication	-When the user is deleted from database. - Uninstalling the

						d is stored in the databas e.		module

Table 4: Critical Security Information

Note: The strength provided by the entropy source is up to 160 bits. SHA-1 is the limiting factor in /dev/random. Therefore, the maximum strength of any key generated by TPCL is 160 bits.

Note: The term "Operator Password" refers to both the user and the Crypto-Officer password.

3. Cryptographic Module Ports and Interfaces

The TPCL has four logical interfaces which can be categorized into the following ISO/IEC 19790 defined interfaces:

i. **Data Input Interface:** All data (except control data entered via the control input interface) enters via the defined "data input" interface, which is processed by the cryptographic module. plaintext, cipher text, keys and CSP's, message length, authentication data, initialization vector, personalization string, additional input, requested random data, key size, key type parameters enter through the data input interface.

ii. **Data Output Interface:** All data (except status data output via the status output interface) exits via the "data output" interface, which is output from the cryptographic module. All data output via the data output interface is inhibited when an error state exists and during self-tests. Upon successful execution of TPCL, plaintext, cipher text, public/ private key pair, message digest, MAC, Random bits and random key parameters is displayed through the data output interface.

iii. **Control Input Interface:** All input commands and control data is entered via the "control input" interface. Mechanism of digest (SHA 224, 256, 384, 512), Mechanism of MAC (HMAC 224,256,384,512), Encryption/ Decryption modes (ECB, CBC) and Prediction Resistance

iv. **Status Output Interface:** All output signals including error indicators or status messages displayed via the "standard output" interface are used to indicate the status of a cryptographic module. For example: BOOL_TRUE showing successful return TPCL_ERR_LIB_INTEG_FAILED, TPCL_ERR_AES_KAT_FAILED, etc.

FIPS Interface	Ports
Data Input	Function input arguments
Data Output	Output parameter of API Function call
Control Input	All API function calls, API Parameters
Status Output	Status information used to indicate the status of the TPCL module i.e. error messages and return codes
Power Input	Through power supply of GPC.

Table 5: Ports and Interfaces

4. Access Control Policy

4.1. Roles

TPCL supports three types of authorized roles:

- **Crypto Officer Role:** There is only one crypto officer in the module that is created by default. Therefore it is not possible to create more users with this role.
- **User Role:** There can be multiple users against this role.
- **Other Role:** Role that do not require authentication.

Following table elaborates the capabilities of each role in detail.

Role	Services (See list below)
User	Performs: • Module initialization • Run Self test • Change his/her password • Logout mechanism • Encryption/ Decryption • Hashing • Signing and MAC function • Generate keys

	• Generate requested number of random bits • Generate domain parameters, public/ private keys and DSA signature • Module finalization • Show status • DRBG generate function health test interval set • Error Display
Crypto Officer	Performs: • Module initialization • Run Self test • Account management • Logout mechanism • Encryption/ Decryption • Hashing • Signing and MAC function • Generate keys • Generate requested number of random bits • Generate domain parameters, public/ private keys and DSA signature • Module finalization • Show status • Error Display • DRBG generate function health test interval set
Other	Performs public services: • Module initialization • Show status • Login authentication • Hashing • Module finalization • Error Display

Table 6: Roles of TPCL

- **Maintenance Role:** There is no provision for a maintenance role in the TPCL.

The crypto officer is in charge of the installation of the module and is also in charge of reviewing the audit logs in order to detect problems in the operation of the module.

4.2. Services

Services refer to all the operations or functions that are performed by the cryptographic module. Service inputs consist of data and control inputs to the cryptographic module. Similarly service outputs consist of data and status output from the cryptographic module. Each service input results in a service output.

The table: 7 describe all services in terms of their respective API functions, associated role and its description.

4.2.1 Show Status

tpcl_show_status function shows the current status of TPCL whether it is in error state or in operational state.

4.2.2 Perform Self-Tests

Power-up self-tests are automatically carried out whenever TPCL library is loaded by consuming application. However, on demand self tests can be initiated by calling ***tpcl_selftest*** utility as described in section 9.

4.2.3 Bypass capability

The TPCL has no bypass capability.

S.No	Service Name	Authorized Role			Access to CSPs	API Function Name	Function Description
		User	Crypto-officer	Other			
1.	Module Initialization	Yes	Yes	Yes	No	tpcl_initialize	Initializes the TPCL module (tpcl_initialize is called directly when the module is loaded into memory)
2.	Run Self Test	Yes	Yes	No	No	tpcl_selftest	Runs the on-demand power-up self-tests functions related to each sub-module (with authentication)
3.	Login Authentication	Yes	Yes	No	Read access to operator password	tpcl_login	Performs authentication of users & updating their status from only accessing public services to now accessing private services as well
4.	Account Management	No	Yes	No	-Read/write access to operator password	tpcl_useradd	Creates users of the TPCL library
		No	Yes	No	-Read/write access to operator password	tpcl_userdel	Deletes users of the TPCL library
		Yes	Yes	No	-Read/write access to operator password	tpcl_changepwd	Changes password of users or crypto-officer

		No	Yes	No	Read access to operator password	tpcl_list_users	List down all available users.
5.	Logout Authentication	Yes	Yes	No	No	tpcl_logout	Logs out user from being an authorized one who could access private services to the one who can only access public services now. Whenever the authenticated user logs out, the credentials of user are zeroized from the memory.
6.	Database Reset	No	Yes	No	Read/write access to operator password	tpcl_reset_db	Reset the database to factory settings
7.	Encryption Service	Yes	Yes	No	Read/write access to AES symmetric key	-tpcl_key_init -tpcl_encrypt -tpcl_key_zeroize	Provides AES encryption service to the user
8.	Decryption Service	Yes	Yes	No	Read/write access to AES symmetric key	-tpcl_key_init -tpcl_decrypt -tpcl_key_zeroize	Provides AES decryption service to the user.
9.	Hashing Service	Yes	Yes	Yes	No	-tpcl_sha -tpcl_sha_init -tpcl_sha_update -tpcl_sha_finalize	Calculates message digest by calling hashing algorithm SHA with the following combinations: ***TPCL_SHA512***, ***TPCL_SHA384***, ***TPCL_SHA256***, ***TPCL_SHA224***
10	Signing & MAC Function (HMAC 224,256,384,512)	Yes	Yes	No	Read/write access to HMAC key	tpcl_hmac	Facilitates user by calculating the MAC of a message using keyed-hash algorithm SHA with the following combinations: ***TPCL_HMAC512***, ***TPCL_HMAC384***,

							TPCL_HMAC256, ***TPCL_HMAC224***
11	Key Generation	Yes	Yes	No	-Read/Write access to CTR-DRBG entropy input, seed, internal state.key and internal state.V -Write access to global/AES/HMAC key	tpcl_keygen	This function generates key, either to be used in any of the internal modules i.e. HMAC and AES or for global use, but the maximum length of global key can be 1024
12	Random Number Generation	Yes	Yes	No	-Read/Write access to CTR-DRBG entropy input, seed, internal state.key and internal state.V	tpcl_instantiate_rand	Creates an instance of a state handle
		Yes	Yes	No		tpcl_uninstantiate_rand	Clears the state handle
		Yes	Yes	No		tpcl_generate_rand	Generates requested number of random bits
13	DSA domain parameters generation	Yes	Yes	No	-Read/Write access to CTR-DRBG entropy input, seed, internal state.key and internal state.V -Read/write access to DSA domain parameters	tpcl_dsa_generateDP	Generates Domain Parameters to be used by DSA module for generation of unique key-pair
14	DSA key pair generation	Yes	Yes	No	-Read/Write access to CTR-DRBG entropy input, seed, internal state.key and	tpcl_dsa_generate_keypa ir	Generates public-private key-pair

					internal state.V -Read access to DSA domain parameters and write access to DSA key pairs		
15	DSA signature generation	Yes	Yes	No	- Read access to domain parameters, per message secret number and DSA private key -Read/Write access to CTR-DRBG entropy input, seed, internal state.key and internal state.V	tpcl_dsa_generate_signature	Generates the DSA signature
16	DSA signature verification	Yes	Yes	No	Read access to DSA domain parameters and DSA public key	tpcl_dsa_verify_signature	Verifies the signature generated by DSA
17	Module Finalization	Yes	Yes	Yes	Write access to CTR-DRBG internal state.key and internal state.V	tpcl_finalize	Performs zeroization of the CSPs used the module
18	Error display	Yes	Yes	Yes	No	tpcl_error_string	Display the error message associated with error code.
19	DRBG generate function health	Yes	Yes	No	No	tpcl_set_rand_interval	Change the default interval to user provided value.

	test interval set						
20	Show Status	Yes	Yes	Yes	No	tpcl_show_status	Shows the status of library at any instant

Table 7: Services of TPCL

4.3. Operator Authentication Mechanism

The role-based authentication mechanism is supported for CO instead of ID-based authentication mechanism, given that it is not possible to change the name of the CO. ID-based authentication mechanism is supported for User. In identity base authentication an operator must explicitly request to assume a role and has to prove his identity (username, password) against this role to gain access to the private services. TPCL supports a password based authentication mechanism to access the above mentioned TPCL User services.

Role	Type of Authentication	Authentication Data	Strength
Crypto Officer	Password based authentication	Username and Password	Password are required to be at least of 8 characters
User	Password based authentication	Username and password	Password are required to be at least of 8 characters

Table 8: Authentication

4.4. Strength of Authentication Mechanism

The strength of authentication mechanism depends upon the length and complexity of password. Password length must be 8 characters long chosen from 96 human readable ASCII characters (ASCII lower case, upper case, digits and non-alphanumeric characters). This makes the probability of successful random attempts (false acceptance) equal to $\left({}^{1}/_{96}\right)^{8}$ which is less than one in 1,000,000.

The enforcement of time provided by the access mechanism in TPCL is as follows:

For the first failed attempt, the module pauses for two seconds and after that for each failed attempt, pause increases by one second. This leads to maximum of 9 possible attempts in one minute that makes the probability of false acceptance equals to $\left({}^{9}/_{96}\right)^{8}$ in one minute, which is less than one in 100,000.

5. Physical Security

The TPCL is software based cryptographic module and thus does not claim any physical security.

6. Operational Environment

The TPCL cryptographic module is capable of running and tested on the following Common Criteria-evaluated platforms:

- **Red Hat Enterprise Linux 5.3 on Dell power edge T110 ii 11th generation server. (http://www.niap-ccevs.org/st/vid10338/)**

The cryptographic module runs in its own operating system threads. This provides it with protection from all other processes, preventing access to all keys, and other CSPs.

6.1 Operational Rules

The TPCL will operate in a modifiable operational environment as per FIPS 140-2 definition. The following rules must be adhered to for operating the TPCL in a FIPS 140-2 compliant manner:

- The Operating System authentication mechanism must be enabled in order to prevent unauthorized users from being able to access system services.
- The crypto officer shall install the cryptographic module correctly following the user guidance.
- All host system components that can contain sensitive cryptographic data (main memory, system bus, disk storage) must be located within a secure environment.
- When the operator finishes using the module, he must logout, he must call the service module finalization and he must call **tpcl_key_zeroize** function if the AES key was set.
- The module checks that it has been initialized before reaching a fully operational state (where all the API functions are callable).
- The operators of the module must protect their credentials (user and passwords).
- Administrative privileges must not be extended to normal users.
- The applications using this library will be single-threaded.

- The module enters in FIPS approved mode of operation after passing successfully the power-up self-tests.
- The Crypto Officer shall be well-trained and non-hostile.
- The Crypto Officer should install the generated files in a location protected by the host operating system security features.
- The operating system is responsible for multitasking operations so that other processes cannot access the address space of the process containing the Module. The Operating system is responsible avoiding the unauthorized reading, writing, or modification of the address space of the Module.
- The writable memory areas of the Module (data and stack segments) are accessible only by a single application, i.e. only one application has access to that instance of the module. The user application accesses the module services in a separate virtual address space with a separate copy of the executable code.
- The application designer must be sure that the client application is designed correctly and does not corrupt the address space of the Module.
- All Critical Security Parameters are verified as correct and are securely generated, stored, and destroyed.
- Secret or private keys that are input to or output from an application must be input or output in encrypted form using a FIPS Approved algorithm. Note that keys exchanged between the application and the FIPS Module may not be encrypted
- Default password of CO, that comes with the library must be changed after installation immediately.
- Password must be at least 8 characters long.
- Maximum allowable password length is 10 characters.
- Password must contain all human readable ASCII characters (ASCII lower case, upper case, digits and non-alphanumeric characters).
- Passwords must be changed every 3 months.

7. Cryptographic Key Management

7.1. Random Number Generation

The TPCL module incorporates "CTR-DRBG" using AES-256 with derivation function as specified in SP800-90A, for generation of cryptographic keys. CTR-DRBG implementation in TPCL supports five internal states. Whenever a user application call API function **tpcl_ instantiate_rand**, a new instance of CTR-DRBG will be created. If, in case all internal states are occupied, then upon receiving **tpcl_ instantiate_rand** request, the CTR-DRBG will return an error TPCL_ERR_DRBG_STATE showing that no internal state is available for this instantiation.

Another Nondeterministic RNG **/dev/random** is used to seed the CTR-DRBG.

7.2. Entropy Input

The "CTR-DRBG AES-256" with derivation function takes entropy input from /dev/random utility of Linux Kernel. First of all entropy function ensure whether there is minimum entropy is available in primary entropy pool through ***cat /proc/sys/kernel/random/entropy_avail*** command. If minimum entropy bits are not available in primary entropy pool, it will wait until there are minimum entropy bits available. The minimum entropy bits are equal to requested instantiation security strength. After that entropy method primarily takes min_length bits from **/dev/random** if available.

The instantiation of DRBG requires the entropy input and the nonce in a single request call as specified in SP800-90. This necessitates that (**min_length** = 3*security strength) random bits are taken against each CTR-DRBG request from entropy source.

Reseeding requires only the entropy input and not the nonce. In this case, (requested security_strength*2) random bits are taken from entropy source

The random number generator gathers environmental noise from device drivers and other sources into an entropy pool. The generator also keeps an estimate of the number of bits of noise in the entropy pool. From this entropy pool random numbers are created.

Following figure depicts the structure of Linux Kernel random number generator.

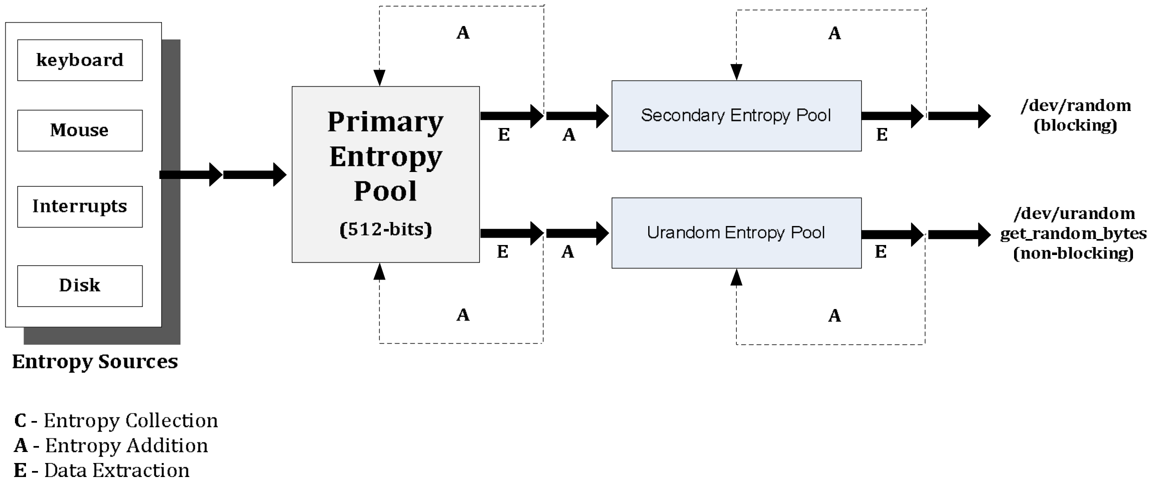

Figure 2:Structure of Linux Random Number Generator

The operating system provider claim about quality of entropy provided by the ***\dev\random*** can be found in Linux programmer's manual pages through **man urandom** command as shown below.

```
RANDOM(4)                   Linux Programmer's Manual                   RANDOM(4)

NAME
       random, urandom - kernel random number source devices

DESCRIPTION
       The character special files /dev/random and /dev/urandom (present since
       Linux 1.3.30) provide an interface to the kernel's random number gener-
       ator.  File /dev/random has major device number 1 and minor device num-
       ber 8.  File /dev/urandom has major device number 1  and  minor  device
       number 9.

       The  random  number  generator  gathers environmental noise from device
       drivers and other sources into an entropy  pool.   The  generator  also
       keeps  an  estimate of the number of bits of noise in the entropy pool.
       From this entropy pool random numbers are created.

       When read, the /dev/random device will only return random bytes  within
       the estimated number of bits of noise in the entropy pool.  /dev/random
       should be suitable for uses that need very high quality randomness such
       as  one-time  pad  or  key generation.  When the entropy pool is empty,
       reads from /dev/random will block until additional environmental  noise
       is gathered.

       A  read  from  the  /dev/urandom device will not block waiting for more
       entropy.  As a result, if  there  is  not  sufficient  entropy  in  the
       entropy  pool,  the  returned  values are theoretically vulnerable to a
       cryptographic attack on the algorithms used by the  driver.   Knowledge
       of how to do this is not available in the current non-classified liter-
       ature, but it is theoretically possible that such an attack may  exist.
       If this is a concern in your application, use /dev/random instead.
```

Figure 3: Entropy of /Dev/random device

Note:

For the key generation process, the security strength requested is always 256 bits. Therefore, it is necessary to ensure, checking the /proc/sys/kernel/random/entropy_avail command, that the entropy source has 256 bits of strength before taking the entropy bits. However, given that the strength provided by the entropy source is up to 160 bits. (SHA-1 is the limiting factor in /dev/random). Therefore, the maximum strength of any key generated by TPCL is 160 bits"

7.3. Key Generation

The TPCL incorporates an internal key generation module. The "Key Generation Module" follows the specifications stated in "sp800-133 Recommendations for Cryptographic key generation, section 5.1 (bullet 5) and section 5.2 (section 5.2.2 bullet 2)". The key sizes supported by the internal key generation module are as listed below:

Algorithm/Key type	Key Size (in bits)
AES	128, 192, 256
HMAC (224,256,384,512)	(112~448), (128~512), (192~768), (256~1024) in multiple of 8.
Global Key	$64 \leq \text{Key size} \leq 1024$ in multiple of 8.

Table 9: Symmetric key generation

The purpose of Global key is to facilitate application developer, if they want to use implementations other than TPCL, but they can acquire key from TPCL key generation module.

Furthermore, the DSA (FIPS 186-3) module facilitates the generation of domain parameters (p, q & g) and the respective public/private key pair. The module returns, on request, either only the domain parameters or the public/private key pair generated from the domain parameters. The key sizes (modulus) supported by the DSA module are as listed below:

i. 2048 bits
ii. 3072 bits

The module supports the use of both internally and externally generated keys. The generated keys (internal/external) are input in or output from the module in plain-text form for further processing.

No intermediate key generation values are output from the module.

7.4. Key Entry and Output

Keys are entered into and output from the Module in plaintext form through the C language APIs.

7.5. Key Storage

The TPCL module, as a software library, does not provide any key storage.

7.6. Zeroization Procedure

The variables holding the secret keys or CSPs are zeroized soon after using them.

8. Electromagnetic Interference/Electromagnetic Compatibility (EMI/EMC)

The Tahir Pak cryptographic Library (TPCL) is installed on a DELL power edge T110 ii 11th generation Server. The make and model of the machine along with the EMI/EMC specifications is as given below:

EMI/EMC of Machines	FCC Rating
DELL power edge T110 ii	Class B

Table 10: EMI/EMC of Machines

9. Self -Tests

The power-up self-tests are carried out automatically on module initialization. The module initialization function i.e. **tpcl_initialize** internally calls the **tpcl_selftest** function to perform the integrity and known answer tests. Once called, the initialization function does not allow any user intervention.

However when an Operator (Crypto officer or User) performs the power up self test on demand through the API function "**tpcl_selftest ()**", if the power up self-tests are successful, the module returns to the "private service" state of the Operator, who carried out the call.

Upon successful completion, the following success indicators are showed in the log file (/var/log/message):

- tpcl_drbg_health test service run successful.
- tpcl_initialize: () selftest service run successful.
- tpcl_initialize: () initialize service run successful.
- tpcl_initialize: () power-on-self-test service run successful.

Failure is indicated in log file (/var/log/message), as follows:

- when a health-test or KAT error occurs:
  - tpcl_initialize: () self test failed
  - tpcl_initialize: () error 1 occurred while running initialize service
- when a integrity error occurs:
  - tpcl_initialize: () error 1 occurred while running initialize service

When the self-tests (on demand or power-up) are executed. If a failure occurs, the module enters in error state and no API function except **tpcl_finalize** may be executed.

Additionally, when the on demand self-tests are performed, **tpcl_selftest** function returns a value (TPCL_SUCCESS in case the self-tests are successfully passed or error in cases of failure). For the power-up self-tests; no other output is performed given that this API function is internally called automatically.

9.1. Power-Up Tests

The TPCL module performs self-tests automatically when the API function tpcl_initialize () is called or on demand when the API function tpcl_selftest () is called.

Whenever the power-up tests are initiated, the module performs the integrity test and the cryptographic algorithm Known Answer Test (KAT). If any of these tests fails, the module enters the error state. The known answer tests (KATs) are performed for the following listed Algorithms:

Algorithm	Tests
AES (128, 192, 256)	AES-ECB KA Encryption AES-ECB KA Decryption AES-CBC KA Encryption AES-CBC KA Decryption
SHA (224, 256, 384, 512)	Short Message Known Answer Hashing Long Message Known Answer Hashing
HMAC (224,256,384,512)	Double Round Known Answer HMAC(224,256,384,512)
DSA signature and verification (2048, 3072 modulus size)	Known Answer Signature for each modulus Known Answer Verification for each modulus
DRBG	2 DRBG KAT for Prediction Resistance true for each security strength(16, 24 and 32 bytes) 2 DRBG KAT for Prediction Resistance false for each security strength(16, 24 and 32 bytes) DRBG health Testing.

Table 11: Known Answer Tests

9.1.1 Integrity Check

The integrity test of TPCL is performed using HMAC-512. The HMAC-512 value of library is pre-computed using **tpcl_hmac** function and stored in a finger print file **libtpclfingerprint.so,** which is delivered with the module. The HMAC-512 value of following items is calculated for integrity test:

i. libtpcl.so
ii. tpcl.h
iii. tpcl_err.h
iv. libgmp.so

The integrity test is initiated by calling the **tpcl_initialize** function. The initialization function in turn calls the non-API function, i.e. **tpcl_integrity_check**, to check the integrity of the crypto library. Fundamentally, the **tpcl_integrity_check** function internally requests the **libtpclfingerprint.so** through the module entry point to get the pre stored HMAC values of library items.

The module compares the HMAC-512 value, generated at runtime (using the **tpcl_hmac** function), of the library components with the pre-computed value to verify the module integrity. Upon failure, the module enters an error state.

The standard C **strncmp, memcmp** and **strcmp** functions are used for comparison.

9.3. Conditional Tests

9.3.1 Pair-wise Consistency Test

The TPCL performs pair-wise consistency test on each DSA public/private key pair it generates. Once generated, a message is constructed by requesting random data (256-bits) from CTR_DRBG (residing inside the cryptographic module). The constructed message (256-bits) is signed and verified by the generated private and public keys respectively. The key pair is returned on successful verification, whereas the module enters an error state upon failure.

9.3.2 Continuous Random Number Generator Test for DRBG

The CTR_DRBG implemented in the TPCL module and its entropy source are both tested for randomness using the continuous random number generator test. Upon failure, the module enters an error state.

The DRBG generates a minimum of 16 bytes per request. Upon request of random data the DRBG does not output first 16 bytes but these are stored for comparison. Each successive

16 byte generated block is compared with previous one and stored in output buffer, if both block are not identical.

9.3.3 Health Test for DRBG

Health testing is performed on CTR-DRBG in power up self test and whenever CTR-DRBG is invoked by the consuming application for random data. Health tests are performed on each function of CTR-DRBG. These functions include instantiate, reseed, generate and un instantiate.

10. Mitigation of Other Attacks

Not applicable