TCG FIPS 140-2 Guidance for TPM 2.0

Version 1.0 Revision 1.0
February 2, 2017
Published

Contact: admin@trustedcomputinggroup.org

TCG Published

TCG

Disclaimers, Notices, and License Terms

Acknowledgements

James Hallman, Atmel

Amy C Nelson, Dell, Inc.

Ga-Wai Chin, Infineon Technologies

Juergen Noller, Infineon Technologies

Dr. David Challener, Johns Hopkins University Applied Physics Lab

Fabien Arrivé, STMicroelectronics

Olivier Collart, STMicroelectronics

Andrew Regenscheid, United States Government

Apostol Vassilev, United States Government

Table of Contents

Table of Tables

1. Scope

The TPM 2.0 FIPS guidance is provided as a supporting document for FIPS 140-2 evaluation of a TPM 2.0 product compliant with TPM 2.0 library level 0 version 1.16. The intended audience for this document includes TPM manufacturers, FIPS Cryptographic Module Validation Program Laboratories and FIPS Evaluators.

This document describes additional development constraints or library interpretation necessary for a successful FIPS evaluation. The intent is to highlight areas of the specification that may require specific attention when the device is in a FIPS approved mode. If the general specification is in compliance with FIPS no guidance is provided on the topic. The organization of the document starting with Section 5 follows the FIPS 140-2 convention and language. Each section contains a FIPS 140-2 Summary section which contains the text from the FIPS 140-2 Security Requirements Summary table for the specific requirement and security level.

This specification targets FIPS 140-2 level 1 or level 2.

2. Terms and Definitions

For the purposes of this document, the acronyms given in Parts 2 and 3 of the TPM 2.0 Library Specification apply.

3. References

1. TCG Trusted Platform Module Library Specification Family 2.0 Revision 1.16 or later https://www.trustedcomputinggroup.org/tpm-library-specification/
2. TCG PC Client Specific Platform TPM Profile for TPM Family 2.0 Revision .43 or later https://www.trustedcomputinggroup.org/pc-client-platform-tpm-profile-ptp-specification/
3. FIPS 140-2 http://csrc.nist.gov/groups/STM/cmvp/standards.html#02
4. Implementation Guidance for FIPS Pub 140-2 and the Cryptographic Module Validation Program

   http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf
5. FIPS 140-2 Derived Test Requirements
   http://csrc.nist.gov/groups/STM/cmvp/documents/fips140-2/FIPS1402IG.pdf

4. Library Addendum

4.1 FIPS bit

In revision 1.16 of [1] the following table was added to Part 2, section 8.10 TPMA_MODES, Table 38:

Table 1 Definition of (UINT32) TPMA_MODES Bits <Out>

Bit	Name	Definition
0	FIPS_140_2	SET (1): indicates that the TPM is designed to comply with all of the FIPS 140-2 requirements at Level 1 or higher.
31:1	Reserved	shall be zero

The FIPS bit is a static flag set by the TPM manufacturer to indicate whether the TPM is designed to comply with all of the FIPS 140-2 requirements at Level 1 or higher. This structure may be read using TPM2_GetCapability as described in [1].

This FIPS bit does not indicate whether a TPM operates in an Approved mode of operation (a mode of the cryptographic module that employs only Approved security functions). It also provides no information whether the TPM is certified according to FIPS 140-2 or in the process of being certified.

5. Cryptographic Module Specification

5.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for the Cryptographic Module Specification is restated below.

Level 1, Level 2:

Specification of cryptographic module, cryptographic boundary, Approved algorithms, and Approved modes of operation. Description of cryptographic module, including all hardware, software, and firmware components. Statement of module security policy.

5.2 Implementation

5.2.1 Cryptographic Algorithms

Whereas the FIPS requirement related to approved and non-approved algorithms can be managed by security policy statements, there are potential hardware or firmware implementations that may require alteration in a FIPS mode of operation to allow the end user to comply with the requirement.

5.2.1.1 Non-approved Algorithms

The below table summarizes TPM 2.0 specified algorithm functions that do not meet FIPS 140-2 cryptographic requirements. Usage of these algorithms in a TPM application is limited to non-cryptographic functions.

Table 2 Non-Approved Algorithms

Algorithm	Use
SHA-1	Used for digital signature verification (legacy) and any non-digital signature application. Not used for digital signature generation.
RSA	Not permitted for digital signature generation, key agreement and key transport schemes with key size = 1024. Usage of 1024 bit or smaller keys is considered equivalent to plaintext or obfuscation versus cryptography.
ECDAA	Used for object creation and approved actions on keys that are non-FIPS compliant. Not used for cryptography. Usage considered plaintext or obfuscation.
XOR	XOR obfuscation used as a hash-based stream cipher.
MGF1	RSAES_OAEP mask generation function equivalent to plaintext or obfuscation versus cryptography.
EC Schnorr	Used for signing and verifying signatures that are non-FIPS compliant. Not used for cryptography. Usage considered plaintext or obfuscation.

5.2.1.2 RSA

RSA key size must be 2048 or above for digital signature generation.

Impacted commands:

- TPM2_Create

5.2.1.3 RSASSA_PSS

Language in [1] Part 1 Appendix B.7 RSASSA_PSS indicates:

"For both restricted and unrestricted signing keys, the random salt length will be the largest size allowed by the key size and message digest size.

NOTE If the TPM implementation is required to be compliant with FIPS 186-4, then the random salt length will be the largest size allowed by that specification."

Impacted commands:

- TPM2_Create
- TPM2_CreatePrimary

5.2.1.4 Primary Key Generation

In TPM 2.0, multiple Primary Keys are derived from a single Primary Seed. The attribute of Primary Keys is that they are reproducible and therefore do not need to be stored persistently in the TPM's memory. (Every time the same public key template is input to the TPM, the same key is output.) This reproducibility is achieved by deriving the Primary Keys using a Pseudo Random Function (PRF) from the Primary Seed. For the PRF, [1] has two example implementations. An SP800-108 KDF using counter mode and HMAC, and a SP800-90A DRBG where the Primary Seed is used as entropy to instantiate the same DRBG instance.

This section summarizes the guidance that NIST provided to TCG for the FIPS compliant usage of a DRBG and a KDF for Primary Key generation.

The RSA key generation routines are specified in appendix B.3 of FIPS 186-4, and the ECC key generation routines in appendix B.4. Generally, the key generation routines can be differentiated in (1) key generation algorithms that start from a seed value and (2) key generation algorithms that repeatedly call an RBG (only applies to RSA). A DRBG can be used for any of the key generation routines. A KDF can be used only for the key generation routines (symmetric or asymmetric) that start with a seed.

In addition, the same secret should not be used with two different crypto algorithms as this is not a good practice. Therefore, all Primary Keys (RSA, ECC, AES), as well as the seed value, that are derived from the same Primary Seed must use the same key generation routine (KDF or DRBG).

In summary:

1) Any key generation routine can be used with a DRBG.
2) Only key generation routines (symmetric or asymmetric) that use a single seed value can be used with a KDF. If the routine ever needs to call a RBG to obtain a new seed, then only a DRBG should be used.

3) For a given primary seed, all primary keys and the seed Value should be generated using either a DRBG or a KDF, not a mix between the two.

Impacted commands:

- TPM2_CreatePrimary

5.2.1.5 RSA Key generation

The TPM Library specification permits multiple methods for RSA key generation, with the methods defined by FIPS 186-4 being recommended. However, as of January 2014, FIPS 140-2 requires use of the methods specified in FIPS 186-4. For FIPS compliance, this correlates to selection of an IFC key pair generation procedure in FIPS 186-4 Appendix B.3. The implementer should be aware that these procedures call for the usage of an approved DRBG or random number generator versus [1] Part 1 defined KDFa function (Section 11.4.9.1). However, NIST has approved the usage of a KDF in those key generation routines (symmetric or asymmetric) that only require a single seed value (see Section 5.2.1.4 on this topic).

The TCG reference implementation uses the procedure of FIPS 186-4 B.3.3 "Generation of Random Primes that are Probably Prime." This method can be used for primes of 1024 bits or larger. Alternate methods may be selected which include smaller prime ranges. The methods described in B.3.5 "Generation of Probable Primes with Conditions Based on Auxiliary Provable Primes" or B.3.6 "Generation of Probable Primes with Conditions Based on Auxiliary Probable Primes" support primes of 512, 1024 or 1536 bits.

Note that the primes for “p” and “q” are of length modulus/2. Hence RSA 2048 would have “p” and “q” prime lengths of 1024-bits.

Impacted commands:

- TPM2_Create
- TPM2_CreatePrimary

5.2.1.6 KDFa

Part 1 of [1] section 11.4.9.1 defines KDFa as using counter mode from SP800-108, with HMAC as the PRF. However, SP800-133 says that a KDF according to this standard is only allowed to derive **symmetric** keys. The implication is that usage of KDFa for key generation of any asymmetric key (RSA or ECC) is not FIPS-compliant. FIPS requires prime generation functions to obtain strings of bits of sufficient security strength from a DRBG or other random number generator mechanism.

NIST has indicated that it intends to update its key generation guidance to allow KDF usage for asymmetric keys. A KDF is permitted as part of the key generation routine if only a single seed value is required by the routine. If multiple seed values are required, then the implementation must use the DRBG method. See the above “Primary Key generation” section on this topic.

The TCG reference code and descriptions in section B.8.2 of [1] part 1 are a single implementation method and are provided as an example reference only. Alternate implementations using the methods permitted in section B.8.1 are permitted.

Note that the TPM may choose to use KDFa for Primary Keys because Primary Keys need to be reproducible (every time you input the same public template, the same key is derived).

Ordinary keys do not need to be reproducible. So, they do not need to use a key derivation function.

Alternatively, an SP800-90A DRBG may be used where the Primary Seed is used as entropy and the name (hash of public template) is used as a personalization string to instantiate the DRBG. Since the seed value is a random number from a real random number generator, it has adequate entropy for this usage. From a FIPS perspective, it is unconventional to use the same seed to instantiate a DRBG several times. Normally only one DRBG output is generated per input seed. However, NIST has agreed to this method and may write an IG on the topic (it may take a while for an IG to appear). It is not an issue that the same random number generator is instantiated each time the same Primary is generated.

Symmetric key generation algorithms may use KDFa as defined in [1] Part 1.

Note there are other sequences where KDFa usage is permitted. Appropriate uses may include generating the HMAC and symmetric keys used for protection of key blobs. So, TPM implementers still need to include the KDFa function in their device.

Impacted commands:

- TPM2_Create
- TPM2_CreatePrimary

5.2.1.7 ECDAA

NIST does not consider the ECDAA algorithm to be cryptography. In FIPS mode of operation, the TPM should not use the ECDAA command related to any CSP. An ECDAA key is never used to approve an action on a key that is compliant with FIPS (i.e., an RSA key or an ECC key).

Users can create objects with ECDAA but they cannot be considered keys (TPM2_Create).

Impacted Commands:

- TPM2_Create
- TPM2_CreatePrimary
- TPM2_Certify
- TPM2_Sign
- TPM2_Commit

5.2.1.8 TPM2_Duplicate

The TPM2_Duplicate command allows sending objects to the NULL hierarchy which sends it off chip unprotected. This is not allowed for FIPS 140-2.

The command has an attribute, “encryptedDuplication”, which should always be SET for FIPS devices. This requires an inner symmetric wrapping prior to the object receiving asymmetric encryption to go off chip. This also prevents the new parent from being TPM_RH_NULL.

5.2.1.9 Input/Output Parameter Encryption

Enforce parameter encryption for commands that have CSPs as input/output parameters. Note 1: commands may be used for non-authorized subjects without an authorization value.

Impacted Commands:

- TPM2_ActivateCredential
- TPM2_Certify
- TPM2_CertifyCreation
- TPM2_Commit
- TPM2_Create
- TPM2_CreatePrimary
- TPM2_Duplicate
- TPM2_ECDH_KeyGen
- TPM2_ECDH_ZGen
- TPM2_EncryptDecrypt
- TPM2_EventSequenceComplete[1]
- TPM2_FieldUpgradeData
- TPM2_FieldUpgradeStart
- TPM2_GetCommandAuditDigest
- TPM2_GetRandom
- TPM2_GetSessionAuditDigest
- TPM2_GetTime
- TPM2_Hash[1]
- TPM2_HashSequenceStart[1]
- TPM2_HierarchyChangeAuth
- TPM2_HMAC
- TPM2_HMAC_start
- TPM2_Import
- TPM2_Load
- TPM2_LoadExternal
- TPM2_MakeCredential
- TPM2_NV_Certify
- TPM2_NV_ChangeAuth
- TPM2_NV_DefineSpace
- TPM2_NV_Extend
- TPM2_NV_Read[1]
- TPM2_NV_ReadPublic
- TPM2_NV_Write
- TPM2_ObjectChangeAuth
- TPM2_PCR_Event[1]
- TPM2_PCR_SetAuthPolicy
- TPM2_PCR_SetAuthValue
- TPM2_PolicyAuthorize
- TPM2_PolicyCounterTimer
- TPM2_PolicyCpHash
- TPM2_PolicyDuplicationSelect
- TPM2_PolicyGetDigest
- TPM2_PolicyNameHash
- TPM2_PolicyNV
- TPM2_PolicyPCR
- TPM2_PolicySecret
- TPM2_PolicySigned
- TPM2_PolicyTicket
- TPM2_Quote
- TPM2_ReadPublic
- TPM2_RSA_Decrypt
- TPM2_RSA_Encrypt
- TPM2_SequenceComplete[1]
- TPM2_SequenceUpdate[1]
- TPM2_SetPrimaryPolicy
- TPM2_Sign
- TPM2_StartAuthSession
- TPM2_StirRandom
- TPM2_Unseal
- TPM2_VerifySignature
- TPM2_ZGen_2Phase

Acc (Access Type): (Z) Zeroize, (W) Write, (E) Execute

Table 3 Table of CSPs and Commands

CSPs	Acc	Commands	Details	References from the spec (1.16)
Hierarchical CSPs				
Primary Seeds (PPS, SPS, EPS, nullSeed)	Z,W	Clear, ChangePPS/EPS		
	E	CreatePrimary	entropy for Primary Keys generation, KDFa key to derive seedValue	seedValue := KDFa (hashAlg, seed, "seedValue", tName, proof, bits) (52), P1, 27.6.4 seedValue
Hierarchy Proofs (phProof, shProof, ehProof, nullProof)	Z,W	Clear(s,e), ChangePPS/EPS, Reset(n)	s=shProof, e=ehProof, n=nullProof	
	E	(1) ContextSave/Load	KDFa key to derive context protection keys, HMAC key to compute context integrity	(symKey, symIv) := KDFa (hashAlg, hProof, vendorString, sequence, handle, bits) (54), P1, 30.3.1 Context Confidentiality Protection contextHMAC := $HMAC_{vendorAlg}$ (hProof, resetValue { || clearCount} || sequence || handle || encContext) (56), P1, 30.3.2 Context Integrity Protection
		(2) cmds that create/verify a ticket	HMAC key to compute ticket	computation dependent on type of ticket
		(3) attestation cmds	only shProof, KDFa key to derive obfuscation value	obfuscation := KDFa (signHandle→nameAlg, shProof, "OBFUSCATE", signHandle→QN, 0, 128) (60), P1, 36.7 Privacy Aspects of Clock to obfuscate for reset/restartCount, FWVersion
		(4) CreatePrimary	only ehProof, KDFa input to derive seedValue	to void child objects in the endorsement hierarchy on TPM2_Clear() or TPM2_ChangeEPS()

CSPs	Acc	Commands	Details	References from the spec (1.16)
		(5) cmds with a wrapped secret	only nullProof, if parent/encryption key = NULL	e.g. TPM2B_PRIVATE, TPM2B_ENCRYPTED_SECRET
Hierarchy Authorization/ Policy (platformAuth/Policy, ownerAuth/Policy, endorsementAuth/Policy, lockoutAuth/Policy)	Z	Reset, Restart(pA,pP), ChangePPS (pP), Clear(o,e,l), ChangeEPS(e)	pA=platformAuth, pP=platformPolicy, o,e,l=owner,endorsement,lockout Auth/Policy	
	W	HierarchyChangeAuth, SetPrimaryPolicy		
	E	cmds that require hierarchy authorization	Password, HMAC authorization, or policy authorization	
Object				
	W	CreatePrimary, Create	applies to all Object CSPs	
	Z	Clear(s,e), ChangePPS(p)/EPS(e)	applies to all Object CSPs p=platform, s=storage, e=endorsement,	
authValue	Z,W	ObjectChangeAuth	only ordinary keys	
	E	(1) cmds that require authorization from this object	handle authentication (pwd or HMAC)	
		(2) StartAuthSession	KDFa key to derive sessionKey	sessionKey := KDFa(sessionAlg, (authValue || salt), “ATH”, nonceTPM, nonceCaller, bits) (19), P1, 19.6.8 sessionKey Creation
		(3) PolicySecret	extended into policyDigest	PolicyUpdate(TPM_CC_PolicySecret, authObject→Name, policyRef) (15), P3, 23.4 TPM2_PolicySecret
	E	cmds that require policy authorization from this object	handle authentication (policy)	
seedValue - derived from PS or random	E	(1) CreatePrimary, Create, Load	only sym keys, compute Hash (unique)	unique := $H_{nameAlg}$ (obfuscate || key) (51), with obfuscate = seedValue.buffer, key =

CSPs	Acc	Commands	Details	References from the spec (1.16)
				sensitive.bits.buffer, P1, 27.5.3.2 TPM_ALG_KEYEDHASH
		(2) Create, Load, ObjectChangeAuth	only asym parent keys, KDFa key to derive child protection keys (symKey, HMACkey)	symKey := KDFa (pNameAlg, seedValue, “STORAGE”, name, NULL , bits) (35), P1, 22.4 Symmetric Encryption HMACkey := KDFa (pNameAlg, seedValue, “INTEGRITY”, NULL, NULL, bits) (37), P1, 22.5 Integrity
symKey - derived from seedValue	E	Create, Load, ObjectChangeAuth	only asym parent keys, encryption key to encrypt sensitive	encSensitive := $CFB_{pSymAlg}$ (symKey, symIv, sensitive) (36), P1, 22.4 Symmetric Encryption
HMACkey - derived from seedValue	E	Create, Load, ObjectChangeAuth	only asym parent keys, HMAC key to compute integrity	outerHMAC := $HMAC_{pNameAlg}$ (HMACkey, symIv || encSensitive || name.buffer) (38), P1, 22.5 Integrity
sensitive - key material, derived from PS or random	E	cmds that use a key for encryption, decryption, signing or ECDH	depending on key attributes, encrypt, decrypt, sign or ECDH	
NV Index				
	W	DefineSpace	applies to all NV Index CSPs	
	Z	UndefineSpace/Special	applies to all NV Index CSPs	
authValue	Z,W	NV_ChangeAuth		
	E	(1) cmds that require authorization from this nv index	handle authentication (pwd or HMAC)	
		(2) StartAuthSession	KDFa key to derive sessionKey	see object
		(3) PolicySecret	extended into policyDigest	see object
authPolicy	E	cmds that require policy authorization from this nv index	handle authentication (policy)	
Session				

CSPs	Acc	Commands	Details	References from the spec (1.16)
	W	StartAuthSession	applies to all Session CSPs	
	Z	FlushContext	applies to all Session CSPs	
salt	E	StartAuthSession	KDFa key to derive sessionKey	sessionKey := KDFa(sessionAlg, (authValue || salt), "ATH", nonceTPM, nonceCaller, bits) (19), P1, 19.6.8 sessionKey Creation
sessionKey	E	cmds that use an authorization session	HMAC key to compute session HMAC	authHMAC := $HMAC_{sessionAlg}$ ((sessionKey || authValueentity), (pHash || nonceNewer || nonceOlder || sessionAttributes)) (27), P1, 19.6.12 Salted and Bound Session Key Generation, computation depends on presence of bind and salt
symKey, IV - derived from session key	E	cmds that have a TPM2B as first parameter	parameter encryption	KDFa (hashAlg, sessionValue, "CFB", nonceNewer, nonceOlder, bits) (34), P1, 21.3 CFB Mode Parameter Encryption
Context				
symKey , IV - derived from proof	Z	FlushContext	flushes the context	
	W,E	ContextLoad/Save	context encryption	(symKey, symIv) := KDFa (hashAlg, hProof, vendorString, sequence, handle, bits) (54), P1, 30.3.1 Context Confidentiality Protection encContext := CFB_{symAlg} (symKey, symIv, context) (55), P1, 30.3.1 Context Confidentiality Protection
Duplication				
inner symKey - provided by caller or TPM	Z,W,E	Duplicate, Rewrap, Import	encryption for inner wrapper	encSensitive := $CFB_{pSymAlg}$ (symKey, 0, innerIntegrity || sensitive) (40), innerIntegrity := $H_{nameAlg}$ (sensitive || name) (39), P1, 23.3.2.2 Inner Duplication Wrapper

CSPs	Acc	Commands	Details	References from the spec (1.16)
Seed - provided by the TPM			KDFa key to derive outer symKey, HMACkey	symKey := KDFa ($np_{NameAlg}$, seed, “STORAGE”, Name, NULL , bits) (42), HMACkey := KDFa ($np_{NameAlg}$, seed, “INTEGRITY”, NULL, NULL, bits) (44), P1, 23.3.2.3 Outer Duplication Wrapper
outer symKey - derived from seed			encryption (outer wrapper)	dupSensitive := $CFB_{npSymAlg}$ (symKey, 0, encSensitive) (43), P1, 23.3.2.3 Outer Duplication Wrapper
outer HMACkey - derived from seed			HMAC key to compute outer wrapper HMAC	outerHMAC := $HMAC_{npNameAlg}$ (HMACkey, dupSensitive || Name) (45), P1, 23.3.2.3 Outer Duplication Wrapper
Credential				
seed - provided by caller	Z,W,E	Activate/MakeCredential	derive symKey, HMACkey	symKey := KDFa ($ek_{NameAlg}$, seed, “STORAGE”, name, NULL , bits) (46), P1, 24.4 Symmetric Encrypt, HMACkey := KDFa ($ek_{NameAlg}$, seed, “INTEGRITY”, NULL, NULL, bits) (48), P1, 24.5 HMAC
symKey - derived from seed			encryption	encIdentity := $CFB_{ekSymAlg}$ (symKey, 0, CV) (47), P1, 24.4 Symmetric Encrypt
HMACkey - derived from seed			HMAC key to compute integrity	identityHMAC := $HMAC_{ekNameAlg}$ (HMACkey, encIdentity || Name) (49), P1, 24.5 HMAC
DRBG				
Entropy, State (Key, V)	Z,W,E	GetRandom, StirRandom, key generation		
Z for ECDH				
Z=x coordinate (xP) of the product (P)	Z,W,E	cmds with an encrypted secret	KDFe key to derive seed	P := h [de,U]Qs,V, Z := xP seed := KDFe(hashAlg, Z, label,

CSPs	Acc	Commands	Details	References from the spec (1.16)
		(salt, duplication, credential)		PartyUInfo, PartyVInfo, bits) (70), P1, C.7.1 ECDH

6. Cryptographic Module Ports and Interfaces

6.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for the Cryptographic Module Ports and Interfaces is restated below.

Level 1, Level 2:

Required and optional interfaces. Specification of all interfaces and of all input and output data paths.

6.2 Implementation

No clarification required

7. Roles, Services and Authentication

7.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for the Roles, Services and Authentication is restated below.

Level 1

Logical separation of required and optional roles and services.

Level 2

Role-based or identity-based operator authentication.

7.2 Implementation

7.2.1 Roles and Services

Level 1

The Security Policy Document (SPD) should describe the roles of the TPM, including TPM Key Hierarchy administrators, Object administrator and Object User, and how those roles relate to TPM services. The SPD should provide guidance on configuring the authorization values and authorization policies for each of these roles.

Level 2

The TPM defines roles are User, Admin, and Dup. These roles are defined in [1] Part 1 Section 19.2. The Security Policy Document should list the supported TPM policy commands, which at a minimum should include the list of mandatory commands as defined in the Platform Specific TPM Profiles and describe in general how policy permits an administrator to manipulate the TPM.

7.2.2 Authentication mechanisms

Level 1

The TPM permits creation of objects with password, HMAC or policy based authorization. The TPM allows the use of a NULL password (Empty Buffer). The SPD should describe permissible uses of Null passwords (shared resources such as SRK may require an update to [4] from NIST), such as PCR and NV Indices.

Level 2

In addition to the requirements from Level 1, the SPD should advise users to avoid use of password authorization with a NULL authorization value for objects such as keys and NV Indices. The SPD should advise users to configure the authorization values of the Storage and Endorsement hierarchies if the user is not using an OS which will manage those authorization values for the user.

The following table provides a listing of the TPM2 commands and indicates whether authorization is required (Y), allowed (O), or not permitted (N). If a command allows

authorization, that indicates there are objects or operations which can be set up with or without authorization. The presence of authorization data would depend on the attributes of the object.

Table 4 Command Requiring Authorization

Name	Command Code	Auth	Comments
TPM_CC_FIRST	0x0000011F		Compile variable. May decrease based on implementation.
TPM_CC_PP_FIRST	0x0000011F		Compile variable. Would decrease if new PP commands are added
TPM_CC_NV_UndefineSpaceSpecial	0x0000011F	Y	
TPM_CC_EvictControl	0x00000120	Y	
TPM_CC_HierarchyControl	0x00000121	Y	
TPM_CC_NV_UndefineSpace	0x00000122	Y	
TPM_CC_ChangeEPS	0x00000124	Y	
TPM_CC_ChangePPS	0x00000125	Y	
TPM_CC_Clear	0x00000126	Y	
TPM_CC_ClearControl	0x00000127	Y	
TPM_CC_ClockSet	0x00000128	Y	
TPM_CC_HierarchyChangeAuth	0x00000129	Y	
TPM_CC_NV_DefineSpace	0x0000012A	Y	
TPM_CC_PCR_Allocate	0x0000012B	Y	
TPM_CC_PCR_SetAuthPolicy	0x0000012C	Y	
TPM_CC_PP_Commands	0x0000012D	Y	
TPM_CC_SetPrimaryPolicy	0x0000012E	Y	
TPM_CC_FieldUpgradeStart	0x0000012F	Y	
TPM_CC_ClockRateAdjust	0x00000130	Y	
TPM_CC_CreatePrimary	0x00000131	Y	
TPM_CC_NV_GlobalWriteLock	0x00000132	Y	
TPM_CC_GetCommandAuditDigest	0x00000133	Y	
TPM_CC_NV_Increment	0x00000134	Y	
TPM_CC_NV_SetBits	0x00000135	Y	

Name	Command Code	Auth	Comments
TPM_CC_NV_Extend	0x00000136	Y	
TPM_CC_NV_Write	0x00000137	Y	
TPM_CC_NV_WriteLock	0x00000138	Y	
TPM_CC_DictionaryAttackLockReset	0x00000139	Y	
TPM_CC_DictionaryAttackParameters	0x0000013A	Y	
TPM_CC_NV_ChangeAuth	0x0000013B	Y	
TPM_CC_PCR_Event	0x0000013C	O	
TPM_CC_PCR_Reset	0x0000013D	O	
TPM_CC_SequenceComplete	0x0000013E	O	
TPM_CC_SetAlgorithmSet	0x0000013F	Y	
TPM_CC_SetCommandCodeAuditStatus	0x00000140	Y	
TPM_CC_FieldUpgradeData	0x00000141	Y	
TPM_CC_IncrementalSelfTest	0x00000142	N	
TPM_CC_SelfTest	0x00000143	N	
TPM_CC_Startup	0x00000144	N	
TPM_CC_Shutdown	0x00000145	N	
TPM_CC_StirRandom	0x00000146	N	
TPM_CC_ActivateCredential	0x00000147	Y	
TPM_CC_Certify	0x00000148	Y	
TPM_CC_PolicyNV	0x00000149	Y	
TPM_CC_CertifyCreation	0x0000014A	Y	
TPM_CC_Duplicate	0x0000014B	Y	
TPM_CC_GetTime	0x0000014C	Y	
TPM_CC_GetSessionAuditDigest	0x0000014D	Y	
TPM_CC_NV_Read	0x0000014E	O	
TPM_CC_NV_ReadLock	0x0000014F	O	
TPM_CC_ObjectChangeAuth	0x00000150	Y	
TPM_CC_PolicySecret	0x00000151	Y	
TPM_CC_Rewrap	0x00000152	Y	

Name	Command Code	Auth	Comments
TPM_CC_Create	0x00000153	Y	
TPM_CC_ECDH_ZGen	0x00000154	Y	
TPM_CC_HMAC	0x00000155	Y	
TPM_CC_Import	0x00000156	Y	
TPM_CC_Load	0x00000157	Y	
TPM_CC_Quote	0x00000158	Y	
TPM_CC_RSA_Decrypt	0x00000159	Y	
TPM_CC_HMAC_Start	0x0000015B	Y	
TPM_CC_SequenceUpdate	0x0000015C	O	
TPM_CC_Sign	0x0000015D	Y	
TPM_CC_Unseal	0x0000015E	Y	
TPM_CC_PolicySigned	0x00000160	Y	
TPM_CC_ContextLoad	0x00000161	Y	
TPM_CC_ContextSave	0x00000162	Y	
TPM_CC_ECDH_KeyGen	0x00000163	Y	
TPM_CC_EncryptDecrypt	0x00000164	Y	
TPM_CC_FlushContext	0x00000165	N	
TPM_CC_LoadExternal	0x00000167	Y	
TPM_CC_MakeCredential	0x00000168	Y	
TPM_CC_NV_ReadPublic	0x00000169	N	
TPM_CC_PolicyAuthorize	0x0000016A	Y	
TPM_CC_PolicyAuthValue	0x0000016B	Y	
TPM_CC_PolicyCommandCode	0x0000016C	Y	
TPM_CC_PolicyCounterTimer	0x0000016D	Y	
TPM_CC_PolicyCpHash	0x0000016E	Y	
TPM_CC_PolicyLocality	0x0000016F	Y	
TPM_CC_PolicyNameHash	0x00000170	Y	
TPM_CC_PolicyOR	0x00000171	Y	
TPM_CC_PolicyTicket	0x00000172	Y	
TPM_CC_ReadPublic	0x00000173	Y	

Name	Command Code	Auth	Comments
TPM_CC_RSA_Encrypt	0x00000174	Y	
TPM_CC_StartAuthSession	0x00000176	Y	
TPM_CC_VerifySignature	0x00000177	Y	
TPM_CC_ECC_Parameters	0x00000178	N	
TPM_CC_FirmwareRead	0x00000179	Y	
TPM_CC_GetCapability	0x0000017A	N	
TPM_CC_GetRandom	0x0000017B	N	
TPM_CC_GetTestResult	0x0000017C	N	
TPM_CC_Hash	0x0000017D	N	
TPM_CC_PCR_Read	0x0000017E	O	
TPM_CC_PolicyPCR	0x0000017F	Y	
TPM_CC_PolicyRestart	0x00000180	Y	
TPM_CC_ReadClock	0x00000181	Y	
TPM_CC_PCR_Extend	0x00000182	O	
TPM_CC_PCR_SetAuthValue	0x00000183	Y	
TPM_CC_NV_Certify	0x00000184	Y	
TPM_CC_EventSequenceComplete	0x00000185	O	
TPM_CC_HashSequenceStart	0x00000186	O	
TPM_CC_PolicyPhysicalPresence	0x00000187	Y	
TPM_CC_PolicyDuplicationSelect	0x00000188	Y	
TPM_CC_PolicyGetDigest	0x00000189	Y	
TPM_CC_TestParms	0x0000018A	N	
TPM_CC_Commit	0x0000018B	Y	
TPM_CC_PolicyPassword	0x0000018C	Y	
TPM_CC_ZGen_2Phase	0x0000018D	Y	
TPM_CC_EC_Ephemeral	0x0000018E	Y	
TPM_CC_PolicyNvWritten	0x0000018F	Y	

8. Finite State Model

8.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for the Finite State Model is restated below.

Level 1, Level 2, Level 3, Level 4: (All requirements the same.)

Specification of finite state model. Required states and optional states. State transition diagram and specification of state transitions.

8.2 Implementation

No clarification required

9. Physical Security

9.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for Physical Security is restated below.

Level 1:

Production grade equipment.

Level 2:

Locks or tamper evidence.

9.2 Implementation

No clarification required

10. Operational Environment

10.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for the Operational Environment is restated below.

Level 1:

Single operator. Executable code. Approved integrity technique.

Level 2

Referenced PP's evaluated at EAL2 with specified discretionary access control mechanisms and auditing.

10.2 Implementation

No clarification required

11. Cryptographic Key Management

11.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for Cryptographic Key Management is restated below.

Level 1, Level 2, Level 3, Level 4: (All requirements the same.)

Key management mechanisms: random number and key generation, key establishment, key distribution, key entry/output, key storage, and key zeroization.

Level 1, Level 2:

Secret and private keys established using manual methods may be entered or output in plaintext form.

11.2 Implementation

For TPM, manual methods of establishing keys are not applicable.

There are multiple ways to create and manage keys in the TPM. The following commands are used to create, interact with, or zeroize keys in the TPM. To determine which of the commands below zeroize which CSPs, refer to Table 3.

Impacted Commands:

- TPM2_ChangePPS
- TPM2_ChangeEPS
- TPM2_UndefineSpaceSpecial
- TPM2_Clear
- TPM2_Create
- TPM2_CreatePrimary
- TPM2_Load
- TPM2_FlushContext
- TPM2_ContextSave
- TPM2_ContextLoad
- TPM2_Unseal
- TPM2_ActivateCredential
- TPM2_MakeCredential
- TPM2_Duplicate
- TPM2_Rewrap
- TPM2_Import
- TPM2_GetRandom

12. EMI/EMC

12.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for EMI/EMC is restated below.

Level 1, Level 2:

47 CFR FCC Part 15. Subpart B, Class A (Business use). Applicable FCC requirements (for radio).

12.2 Implementation

No clarification required (Individual IC's do not require EMI/EMC testing for FIPS)

13. Self-Tests

13.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for Self-Tests is restated below.

Level 1, Level 2, Level 3, Level 4: (All requirements the same.)

FIPS 140-2 requires that ALL cryptographic algorithms be self-tested at "Power-up". FIPS 140-2 allows non-security relevant services to operate after the integrity test passes. According to NIST IG 1.7, it is possible to perform a serialized self-test, however different TPM vendors may implement this function differently. As such, this document provides guidance on how a vendor may choose to perform a serialized self-test. The actual implementation is dependent upon the TPM Vendor and their lab.

The below table identifies the cryptographic algorithms required in a PC Client specific TPM 2.0 module with guidance on the type of self-test or on-demand test required in accordance to the referenced NIST standard. A TPM vendor may augment this table with additional algorithms.

Table 5 Self-Test Requirements

Self-Test	Algorithm details	FIPS 140-2		Notes
		Power-up	Condi-tional	
Firmware integrity	EDC, MAC or digital signature	X		FIPS 140-2 4.9.1: EDC (16-bit min) or approved authentication technique applied to all validated software/firmware (including NVM and ROM).
HMAC	KAT	X		FIPS 198-1 HMAC KAT requested on only one of the underlying supported SHA algorithm. IG9.1 Underlying SHA algorithm KAT not required if HMAC KAT done.
SHA-1	KAT	X		FIPS 180-4 (separate KAT from SHA-256)
SHA-256	KAT	X		FIPS 180-4 (separate KAT from SHA-1)
AES encryption	KAT	X		FIPS 197, SP800-38A. KAT with one of the key sizes (128, 192 or 256 bits) in CFB mode.
AES decryption	KAT	X		
ECDSA sign	KAT ECC_NIST_P256	X		FIPS 186-4. KAT on signature generation then signature verification if randomization parameter is fixed or pair-wise consistency test (i.e. no intermediate comparison on generated signature).
ECDSA verify	KAT ECC_NIST_P256	X		
ECDH	KAT ECC_NIST_P256	X		SP800-56Ar2 6.2.2 IG 9.6: KAT on primitive "Z" calculation. Point multiplication on an elliptic curve

Self-Test	Algorithm details	FIPS 140-2		Notes
		Power-up	Condi-tional	
				(comparison of x-coordinate sufficient). Note: This also covers testing of ECDH_ZGen
RSA sign	KAT RSASSA-PSS or RSASSA-PKCS1-v1_5 (signature generation)	X[1]		PKCS#1v2.1, FIPS 186-4. IG 9.4: RSA sign KAT validates RSA decryption. Use 2048 key, either scheme. See Note 1 below concerning RSASSA_PSS.
RSA verify	KAT RSASSA-PSS or RSASSA-PKCS1-v1_5 (signature verification)	X		PKCS#1v2.1, FIPS 186-4 IG 9.4: RSA verify KAT validates RSA encryption. Use 2048 key, either scheme. IG 9.4. Output of RSA signature KAT might be used as input to RSA signature verification.
ECC key generation	ECC_NIST_P256		X[2]	Whenever an ECC key is generated (any ECC algorithm) a pairwise consistency test is required. Test is dependent upon intended usage. See Note 2 below.
RSA key generation	Key length 2048 bits		X[2]	Whenever a key is generated (any RSA algorithm) a pairwise consistency test is required. Test is dependent upon intended usage. See Note 2 below.
KDFa	KAT	-	-	TPM 2.0 Part 1: 11.4.9.1. SP800-108. Previous KAT of underlying HMAC function sufficient (IG 9.1).
KDFe	KAT	-	-	TPM 2.0 Part 1: 11.4.9.3. SP800-56A 5.8.1.2.1 Previous KAT of underlying SHS function sufficient (IG 9.6).
DRBG Instantiate	KAT (might be grouped in one single KAT)	X[3]		SP800-90A Jun2015 11.3.2. For On-Demand testing, see Note 3.
DRBG Generate		X[3]		SP800-90A Jun2015 11.3.3. For On-Demand testing, see Note 3. Per IG9.8, for SP800-90A RBGs, a continuous random number test is not required.
DRBG Reseed		X[3]		SP800-90A Jun2015 11.3.4. For On-Demand testing, see Note 3.
DRBG Uninstantiate		-	-	SP800-90A Jun2015 11.3.5. Not required (only done during device validation)

Self-Test	Algorithm details	FIPS 140-2		Notes
		Power-up	Condi-tional	
NDRNG	RCT or CRNGT	X	X	IG9.8: The NDRNGs shall perform either the RCT (as described in IG9.8) or the CRNGT as described in AS09.42 and AS09.43.

Note 1: IG9.4 requires KAT shall be performed for RSA unless a module supports *only* RSASSA-PSS (in which case a pair-wise test may be used instead). In a module supporting RSASSA-PKCS1-v1_5 and RSASSA-PSS a KAT shall be performed, either on RSASSA-PSS if randomization parameter is fixed, or on RSASSA-PKCS1-v1_5.

Note 2: Pairwise consistency test conditions. FIPS 140-2: The pairwise consistency test is done in accordance to the intended usage. See FIPS 140-2 DTR requirements AS09.30, AS09.31 and AS09.33.

Note 3: SP800-90A guidelines are that this function "should" be testable on demand. It is not required.

13.2 Implementation

The firmware integrity test and algorithm tests must be executed before the first access to a CSP. The execution time of all self-tests listed in section 13.1 may be too long for some platforms requiring a fast boot time.

According to [4] 1.7, it is permissible to consider several approved modes of operations. For each approved mode of operation, the self-tests executed may be different. The two constraints are:

- the cryptographic module shall reinitialize and perform all power-up self-tests associated with the new Approved mode of operation,
- Only the self-tested algorithms may be accessed in the approved mode.

A TPM vendor may define two approved modes of operation in the security policy. The self-tests required to access the first mode would be limited and relatively fast. The algorithms tested must comply with the commands that must be supported in [2] Section 5.5.1.6.

All firmware and algorithm self-tests must be executed to access the second mode, including functions that were previously tested. This second mode supports all of the commands and CSPs defined in the security policy.

The TPM vendor must describe the properties of the two approved modes of operation.

The following tables define the approved modes properties. The descriptions may be vendor specific and may not be implemented by all vendors.

Approved mode 1

This mode is the default mode when the TPM powers up.

Table 6 Approved Mode 1

Properties	Description
Definition	Definition of the mode Example: Transient mode
Configuration	Definition of the event to enter mode Example: when TPM is powered up
Services available	TPM vendor must list the security services available; note example below does not list all available services. This would likely be required by a lab in the Security Policy. Example: All services that don't use asymmetric cryptography (RSA, ECDSA, ECDH)
Algorithms used	TPM vendor must list the algorithms used in this mode. Example: SHS / HMAC / AES / DRBG / KDF
CSPs used	TPM vendor must list the accessible CSPs ; note example below does not list all available services. This would likely be required by a lab in the Security Policy. Example: Only asymmetric CSPs can't be used (RSA and ECC keys)
Self-tests	TPM vendor must list the algorithms self-tested and the firmware integrity test. Example: SHS / HMAC / AES / DRBG / KDF

Approved mode 2

This mode is the approved mode of operation where all CSPs are accessible.

Table 7 Approved Mode 2

Properties	Description
Definition	Example: Full approved mode of operation
Configuration	Example: TPM2_SelfTest(full=YES) execution
Services available	Example: All services
Algorithms used	Example: All supported algorithms
CSPs used	Example: All CSPs
Self-tests	Example: SHS / HMAC / AES / DRBG / KDF / RSA / ECDH / ECDSA and firmware integrity test. See Table 5 Self-Test Requirements for specific tests.

14. Design Assurance

14.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for Design Assurance is restated below.

Level 1:

Configuration management (CM). Secure installation and generation. Design and policy correspondence. Guidance documents.

Level 2:

CM system. Secure distribution. Functional specification.

14.2 Implementation

No clarification required

15. Mitigation of Other Attacks

15.1 FIPS 140-2 Summary

From Table 1 of FIPS 140-2, the security requirements summary for Mitigation of Other Attacks is restated below.

Level 1, Level 2, Level 3, Level 4: (All requirements the same.)

Specification of mitigation of attacks for which no testable requirements are currently available.

15.2 Implementation

No clarification required