ERRATA

# ERRATA

**Errata Version 1.6**
**January 9, 2023**

# FOR

## TCG Trusted Platform Module Library

**Specification Version 2.0**
**Revision 1.16**
**October 30, 2014**

**Contact:** admin@trustedcomputinggroup.org

**TCG PUBLISHED**

## CONTENTS

# 1 Introduction

This document describes errata and clarifications for the TCG Trusted Platform Module Library Version 2.0 Revision 1.16 as published. The information in this document is likely – but not certain – to be incorporated into a future version of the specification. Suggested fixes proposed in this document may be modified before being published in a later TCG Specification. Therefore, the contents of this document are not normative and only become normative when included in an updated version of the published specification. Note that since the errata in this document are non-normative, the patent licensing rights granted by Section 16.4 of the Bylaws do not apply.

# 2 Errata

## 2.1 Policy with trial session

In Part 3, section 23.4 and 23.9, the general description of the commands TPM2_PolicySecret and TPM2_PolicyNV specifically say that the authorization value is not checked if a trial session is being used. The exemplar code actually does do this check. As a result, if the authorization value given is not correct, the result may be either a success or an authorization failure (with accompanied increase in the dictionary attack counter). In a future version of the specification, this may be reconciled. It should be noted that if policies are calculated outside the TPM, no authorization check is needed.

## 2.2 Encryption with trial session

In the current specification, Part 1 Table 7, it indicates that you can use encryption and decryption attributes with a trial session. In fact you cannot because a trial session handle can never be used in the authorization area.

## 2.3 CFB Mode Parameter Encryption

Equation (34) in Part 1, section 21.3, CFB Mode Parameter Encryption needs to have *sessionValue* instead of *sessionKey* in the equation and the following description. The exemplar code uses *sessionValue* which is correct.

## 2.4 Authorization sessions

In Part 1, section 19.6.17 Authorization Session Termination, the current specification says

- when the TPM executes TPM2_Startup(TPM_SU_CLEAR), all authorization sessions are terminated; and
- when the TPM executes TPM2_Startup(TPM_SU_STATE), authorization sessions in TPM memory will be terminated but sessions stored off the TPM will remain active.

This is incorrect. It should state

- on TPM Reset, all authorization sessions are terminated; and
- on TPM Resume or TPM Restart, authorization sessions in TPM memory will be terminated but sessions context saved off the TPM will remain active.

## 2.5 TPM2_Quote

If no key is provided in the *signHandle* parameter then this command can either return an error or (if the signing scheme is present) a TPMS_ATTEST structure using the hash algorithm provided in the signing scheme.

Currently the reference code returns an error while the text in Part 3, section 18.4 TPM2_Quote indicates that the TPMS_ATTEST structure is returned.

## 2.6 lockoutAuth behavior

In Part 1, section 19.11.5 Authorization Failures Involving *lockoutAuth*, it indicates that if *lockoutRecovery* is set to zero, then the TPM will not allow the usage of *lockoutAuth* after an authorization failure until the next TPM Reset. This disagrees with Part 3, section 25.3 TPM2_DictionaryAttackParameters, which indicates that a TPM Reset or TPM Restart is required.

In the reference implementation the use of *lockoutAuth* is enabled on any TPM2_Startup() if *lockoutRecovery* is zero. The behavior in the reference code is correct, the descriptions in Part 1 and 3 are incorrect. *lockoutAuth* is reset on TPM Reset, TPM Restart and TPM Resume if *lockoutRecovery* is zero.

## 2.7 Sign/decrypt attribute encoding

According to the current definition in Part 1, Table 24, a SymCipher object can be used to encrypt and decrypt data if the *decrypt* attribute is SET. This is incorrect. The current Table 24 is only valid for KeyedHash and asymmetric objects.

For a SymCipher object the *sign* attribute encodes encryption and the *decrypt* attribute encodes decryption. Thus, a SymCipher object can only be used for encryption if the *sign* attribute is SET. To allow encryption and decryption, both the *sign* and *decrypt* attribute need to be SET. This allows SymCipher objects to differentiate between encryption and decryption.

This issue affects the command TPM2_EncryptDecrypt() and the table in Part 2 that defines the allowed scheme values for SymCipher object .

## 2.8 TPM2_EncryptDecrypt

The scheme selection in TPM2_EncryptDecryt() is specified incorrectly in Part 3 section 15.2, since it doesn’t work like the other scheme selections. The correct behavior is specified below:

1) *keyHandle* is not allowed to reference a restricted key (TPM_RC_ATTRIBUTES).
2) If the key has a mode that is not TPM_ALG_NULL, then the caller cannot override it. The caller must either pick the same mode or TPM_ALG_NULL (TPM_RC_MODE).
3) If both the caller and the key have a mode selection of TPM_ALG_NULL, then it is an error (TPM_RC_MODE).

To allow these selections, Table 126 in Part 2 should be replaced with:

**Table 1 — Definition of TPMU_SYM_MODE Union**

| Parameter | Type | Selector | Description |
|---|---|---|---|
| !ALG.S | TPMI_ALG_SYM_MODE+ | TPM_ALG_!ALG.S | |
| sym | TPMI_ALG_SYM_MODE+ | | when selector may be any of the symmetric block ciphers |
| xor | | TPM_ALG_XOR | no mode selector |
| null | | TPM_ALG_NULL | no mode selector |

## 2.9 XOR Obfuscation

When a KeyedHash Object is used for encryption, the creator of the key has the option of limiting the use of the key to specific schemes or of deferring the choice until the object is used. The current scheme definitions do not currently allow this selection.

To allow the intended functionality, Table 139 in Part 2 should be replaced with:

**Table 2 — Definition of TPMS_SCHEME_XOR Structure**

| Parameter | Type | Description |
|---|---|---|
| hashAlg | TPMI_ALG_HASH+ | the hash algorithm used to digest the message |
| kdf | TPMI_ALG_KDF+ | the key derivation function |

Currently, no command supports direct XOR encryption/decryption.

## 2.10 TPM2_NV_UndefineSpaceSpecial

In the reference implementation the TPM enters Failure Mode if the command TPM2_NV_UndefineSpaceSpecial() is executed with a policy that contains TPM2_PolicyAuthValue(). When the response HMAC is computed the code tries to access the authorization value for an NV index that was deleted during the command. This causes the TPM simulator to assert.

In a future version of the Library specification the TPM will use EmptyAuth for the computation of the response HMAC if the *authValue* for an entity was deleted during the command.

Only platform manufacturers are affected by this issue. It is recommended that platform manufacturer do not include TPM2_PolicyAuthValue() in the authorization policy used to authorize TPM2_NV_UndefineSpaceSpecial().

## 2.11 TPM_PT_NV_BUFFER_MAX

In the reference code, Part 4 section 9.14.3.1, function TPMPropertyIsDefined() the property tag TPM_PT_NV_BUFFER_MAX is missing in the list of properties. Therefore the command TPM2_Get-Capability (capability = TPM_CAP_TPM_PROPERTIES, property = TPM_PT_NV_BUFFER_MAX) does not return the value for MAX_NV_BUFFER_SIZE.

## 2.12 Session-based Encryption

According to Part 1, section 21.2 XOR Parameter Obfuscation and 21.3 CFB Mode Parameter Encryption, *sessionValue* in the equations (33) and (34) is the session-specific HMAC key.

This definition of *sessionValue* is not correct, in alignment with Part 4 it should say:

- When the encryption session is not used for authorization, *sessionValue* is the *sessionKey*.
- When the encryption session is also an authorization session, *sessionValue* is the concatenation of *sessionKey* and *authValue*. The binding of the session is ignored.

In the reference implementation the check whether *authValue* should be included in the latter case is not performed correctly for an NV Index in the functions ParseSessionBuffer() and BuildResponseSession() (Part 4, 6.4.4.10 and 6.4.5.11). As a result the TPM might not encrypt/ decrypt the parameter with the proper *sessionValue*. Affected by this issue are commands that use the same session to authorize an NV Index and to encrypt/ decrypt a parameter.

## 2.13 TPM2_PCR_Allocate

In the reference code, the function PCRAllocate in PCR.c (Part 3, 22.5) does not correctly determine if the DRTM or HCRTM PCR is properly allocated. As a result, the command TPM2_PCR_Allocate might fail even if the PCR are selected correctly.

## 2.14 TPM_SPEC Date Constants

The spec date fields TPM_SPEC_YEAR and TPM_SPEC_DAY_OF_YEAR defined in Part 2, Table 6 currently indicate the date of the Library specification according to which a TPM is implemented. Unfortunately this does not give information about the errata a TPM implemented and therefore causes problems with regard to testing.

In order to identify the implemented errata in a TPM, the spec date fields TPM_SPEC_YEAR and TPM_SPEC_DAY_OF_YEAR shall indicate the date of the applicable errata.

When implementing the first version of the Library specification (no errata), the date fields can be set to values of zero or to the date of the Library specification. When the TPM is updated to include the errata, the date fields are required to indicate the date of the applicable errata. The TPM vendor is required to implement all errata as of the indicated date.

If a TPM is implemented according to the Library specification version 1.16, and all issues documented in this errata are fixed, then the TPM would report this document's date as TPM_SPEC_YEAR and TPM_SPEC_DAY_OF_YEAR as shown in the table below.

**Table 3 — Definition of (UINT32) TPM_SPEC Constants <>**

| Name | Value | Comments |
|---|---|---|
| TPM_SPEC_FAMILY | 0x322E3000 | ASCII “2.0” with null terminator |
| TPM_SPEC_LEVEL | 00 | the level number for the specification |
| TPM_SPEC_VERSION | 116 | the version number of the spec (001.16 * 100) |
| TPM_SPEC_YEAR | 2023 | the year of the applicable errata version |
| TPM_SPEC_DAY_OF_YEAR | 9 | the day of the year of the applicable errata version (9th of January) |

### 2.15 Bound audit session

If a bound session is first used for both audit and authorization and then subsequently used for authorization, the reference code computes the response HMAC incorrectly for the subsequent uses for authorization.

### 2.16 Trailing zeros in TPM2_LoadExternal

The password authorization of an object might fail even though the correct password is provided, if the object was loaded with TPM2_LoadExernal and the *authValue* had trailing zeros in its private area. This is because TPM2_LoadExternal (Part 3, 12.3) does not remove trailing zeros from the *authValue* before loading the object.

However, trailing zeros are removed from the password during session processing in CheckPWAuthSession (Part 4, 6.4.4.2) before it is compared to the *authValue* of the authorized entity.

### 2.17 TPM_CAP_ALGS

In Part 4, section 9.1.2 Includes and Defines of AlgorithmCap.c, TPM_ALG_KDF1_SP800_56A is mistyped in the algorithm property definition. As a result, the command TPM2_GetCapability (capability = TPM_CAP_ALGS) does not return TPM_ALG_KDF1_SP800_56A in the list of supported algorithms even though it might be implemented.

### 2.18 TPM2_StartAuthSession

The general description in Part 3, 11.1 TPM2_StartAuthSession contains an incorrect statement. Paragraph c) states that the TPM shall return TPM_RC_VALUE if *tpmKey* references a symmetric block cipher or a *keyedHash* object and *encryptedSalt* contains a value that is larger than the size of the digest produced by the *nameAlg* of *tpmKey*.

It should state that the TPM shall return TPM_RC_KEY if *tpmkey* does not reference an asymmetric key.

### 2.19 phEnableNV

The general description in Part 3, 9.3 TPM2_Startup indicates that *phEnableNV* is SET on any TPM2_Startup. This is incorrect. *phEnableNV* is SET on TPM Reset or TPM Restart and preserved by TPM Resume. The reference code implements it correctly.

### 2.20 _PRIVATE

Table 190 in Part 2 is incorrect. The type of *sensitive* should be TPM2B_SENSITIVE instead of TPMT_SENSITIVE. Therefore the table should be replaced with:

**Table 190 — Definition of _PRIVATE Structure <>**

| Parameter | Type | Description |
|---|---|---|
| integrityOuter | TPM2B_DIGEST | |
| integrityInner | TPM2B_DIGEST | could also be a TPM2B_IV |
| sensitive | TPM2B_SENSITIVE | the sensitive area |

The wrong type of *sensitive* might lead to an error in TPM2_Import and TPM2_Duplicate if the sensitive data is the maximum allowed size (e.g. RSA 2048 bits) as both commands expect the object type to be TPM2B_SENSITIVE.

### 2.21 ECDSA Signature Verification

In the reference implementation the ECDSA signature verification might fail in TPM2_VerifySignature or TPM2_PolicySigned if the digest size of the signed message is larger than the curve order. This is because the reference implementation does not modify the digest consistently when it is larger than the curve order before it is used in the ECDSA sign or verify operation.

The ECDSA sign and verify operations are implemented in Part 4, B.13.3.2.17. SignEcdsa() and B.13.3.2.22. ValidateSignatureEcdsa().

### 2.22 Attestation block Obfuscation

In the reference implementation FillInAttestInfo() (Part 4, 7.2.2.1) does not always compute an obfuscation value. As a result, the privacy-sensitive information returned by attestation commands is not always obfuscated when it should be. If a key is loaded with TPM2_LoadExternal that has *nameAlg* set to TPM_ALG_NULL in its public area template and is then used for signing an attestation, the returned *firmwareVersion*, *resetCount*, and *restartCount* in the attestation block might not be obfuscated. The KDFa used to calculate the obfuscation value uses the *nameAlg* of the signing key (which might be TPM_ALG_NULL) as hash algorithm.

The correct behavior is to always use the context integrity hash algorithm to calculate the obfuscation value as the privacy protection should be provided by the TPM algorithms and not by the algorithms chosen by a caller.

### 2.23 sensitiveDataOrigin

According to Part 1, section 25.2.3, the *sensitiveDataOrigin* attribute may not be SET in an asymmetric object. This is incorrect, it should say when an asymmetric key is created, this attribute must be SET.

The requirement that *sensitiveDataOrigin* be SET for asymmetric objects is enforced indirectly. When an asymmetric key is created, the caller is not allowed to provide the sensitive data of the key. If the caller does not provide the sensitive data, then *sensitiveDataOrigin* is required to be SET. Since this relationship is only checked when the object is created, *sensitiveDataOrigin* is allowed to have any setting when an object is loaded or imported.

### 2.24 Qualified Name in TPM2_LoadExternal

The reference code calculates the Qualified Name incorrectly in TPM2_LoadExternal. If a public area or both a public and sensitive area is loaded with TPM2_LoadExternal, the Qualified Name is calculated as $QN \coloneqq \mathbf{H}_{nameAlg}$ (TPM_RH_NULL **||** *Name*).

As correctly specified in Part 3, 12.3 TPM2_LoadExternal, the Qualified Name should be the same as the Name.

### 2.25 TPM2_ECDH_KeyGen

Part 1 and Part 3 specify different requirements for the attributes of *keyHandle* in TPM2_ECDH_KeyGen. According to Part 1, C.8.2 TPM2_ECDH_KeyGen(), the key may be either *sign* or *encrypt.* According to the reference implementation in Part 3, 14.4 TPM2_ECDH_KeyGen, *keyHandle* must have *restricted* CLEAR and *decrypt* SET.

Future versions of TPM2_ECDH_KeyGen will not check the *restricted*, *sign*, or *decrypt* attribute of *keyHandle.* As only the public portion of the key needs to be loaded for this command, no attribute checks are necessary.

## 2.26 TPM2_RSA_Encrypt

According to Part 3, 14.2 TPM2_RSA_Encrypt, the key referenced by *keyHandle* is required to be an RSA key (TPM_RC_KEY) with the decrypt attribute SET (TPM_RC_ATTRIBUTES).

Future versions of TPM2_RSA_Encrypt will not check the *restricted*, *sign*, or *decrypt* attribute of *keyHandle.* As only the public portion of the key needs to be loaded for this command, no attribute checks are necessary.

## 2.27 Read offset for counter and bits indexes

For an NV counter or bits index, the reference implementation will read the counter or bit field value starting at offset zero, even if an offset other than zero is specified as long as the offset is within the valid range of the NV Index. This does not indicate that parameter checks for *offset* are skipped.

To be consistent with the reference code, Part 3, section 23.9 TPM2_PolicyNV, 31.13 TPM2_NV_Read, and 31.16 TPM2_NV_Certify should say that for an NV Index with the TPMA_NV_COUNTER or TPMA_NV_BITS attribute SET, the TPM may ignore the *offset* parameter and use an offset of 0. Therefore, it is recommended that the caller set the *offset* parameter to 0 for interoperability.

## 2.28 EC Schnorr

The EC Schnorr Sign and EC Schnorr Signature Validate description defined in Part 1, C.4.3.2, and C.4.3.3 should be replaced with the description below to be more consistent with ISO/IEC 14888-3:2016. The change is in step d) of EC Schnorr Sign, and step c) of EC Schnorr Signature Validate. The changes compared to 1.16 are:

1) Leading bytes of zero are no longer removed from $e$ and $e'$ as previously indicated in NOTE 1, instead the function FE2BS() is used to convert $x_E$.
2) The modular reduction of the digest produced by $\mathbf{H}_{schemeHash}$ is changed to a truncation.
3) The order of $x_E$ and $P$ in the hash computation is reversed.

### 2.28.1 EC Schnorr Sign

Part 1, C.4.3.2 EC Schnorr Sign should be replaced with the following description:

An EC Schnorr signature is generated when the signing scheme for a key is TPM_ALG_ECSCHNORR. The scheme many be used in any signing operation

To sign a digest $P$

a) set $k$ to a random value such that 0 < $k$ < $n$
b) compute $E \coloneqq (x_E, y_E) \coloneqq [k]G$
c) if $E$ is the point at infinity, go to a)
d) compute $r \coloneqq \mathbf{TRUNC}(\mathbf{H}_{schemeHash}(\mathbf{FE2BS}(x_E) \mid\mid P), n)$

NOTE 1 $x_E$ is.a field element with the same number of bits as the curve order $n$

NOTE 2 **TRUNC**() is a function that reduces the number of octets in the first argument until it has no more octets than the second argument. Tuncation occurs from the less significant end of the number. If the digest produced by $\mathbf{H}_{schemeHash}$ has the same number of octets as the curve order $n$, then no truncation occurs.

NOTE 3 **FE2BS**() is a function that converts the number $x_E$ (a field element) into a canonical value (octet or byte string) with the same number of octets as the field order $n$. This may result in a value with leading octets of zero. As $x_E$ is computed (mod $p$) the value may be greater than $n$

e) compute integer $s \coloneqq (k + rd_S)$ (mod $n$)

NOTE 4 This is the same computation as step 0 in 2.29.

f) if $s = 0$ or $s = k$ go to a)

NOTE 5 The $s = k$ check is to eliminate the possibility that $0 = r$ (mod $n$). Optionally, an implementation could check after d) that $0 \neq r$ (mod $n$).

The signature is the tuple $(r, s)$.

### 2.28.2 EC Schnorr Signature Validate

Part 1, C.4.3.3 EC Signature Validate should be replaced with the following description:

To validate a Schnorr signature $(r, s)$ over digest $P$

a) verify that $0 < s < n$

b) compute $(x_E, y_E) \coloneqq [s]G + [-r]Q_S$

c) compute $r' \coloneqq \textbf{TRUNC}(\mathbf{H}_{schemeHash}(\textbf{FE2BS}(x_E) \mid\mid P), n)$

d) the signature is valid if $r' = r$

NOTE The comparison of r' and r is done assuming that both values are numeric and not octet strings. This reduces the chance of interoperability problems due to padding performed on r.

## 2.29 ECDAA Sign Operation

The steps of the ECDAA Sign Operation defined in Part 1, C.4.2 should be replaced with the steps described below. The change is that a random value is included in the computation of T to allow a security proof.

The signature is created using a modified Schnorr signature using the $P$ and $r$ values described above:

a) set $k$ to a random value such that $0 < k < n$

b) compute $T \coloneqq \mathbf{H}(k \mid\mid P)$(mod $n$)

c) compute integer $s \coloneqq (r + Td_S)$(mod $n$)

d) if $s = 0$, output failure (negligible probability)

The signature is the tuple $(k, s)$.

NOTE The $k$ value is returned in the $R$ parameter of the TPMT_SIGNATURE structure.

## 2.30 Error Codes

### 2.30.1 Introduction

The following section resolves ambiguities with regards to errors codes where the specification text and the reference code specify something different.

### 2.30.2 TPM2_HMAC, TPM2_HMAC_Start

In Part 3, section 15.4 TPM2_HMAC and 17.2 TPM2_HMAC_Start, the general description states that the TPM shall return TPM_RC_ATTRIBUTES if the key referenced by handle is not a signing key. The correct error code is TPM_RC_KEY (same as for all signing commands including attestation commands where the key is not a signing key). The reference code implements the error codes correctly.

### 2.30.3 TPM2_PolicySigned

The Error Return Code Table in Part 3, section 23.3 TPM2_PolicySigned indicates that TPM_RC_KEY is returned if authObject is not a signing scheme. This is incorrect. The *sign* attribute is not required to be SET for authObject in TPM2_PolicySigned. The reference code is implemented correctly.

### 2.30.4 TPM2_NV_DefineSpace

The current description in Part 3, section 31.3 TPM2_NV_DefineSpace indicates that if TPMA_NV_EXTEND is SET, then *publicInfo→dataSize* shall match the digest size of the *publicInfo.nameAlg* or the TPM shall return TPM_RC_SIZE. However, the reference code returns TPM_RC_ATTRIBUTES. The correct response for this error is TPM_RC_SIZE as described in the specification text.

### 2.30.5 TPM2_NV_Read/ Write/ Certify, TPM2_PolicyNV, TPM2_PolicyCounterTimer

According to the command description in Part 3, TPM2_NV_Read, TPM2_NV_Write and TPM2_NV_Certify should return the error code TPM_RC_NV_RANGE when the range defined by the *size* and *offset* parameter is outside the range of the referenced NV Index.

However, TPM_RC_NV_RANGE is not allowed to have a response code modifier that would provide additional information about the type of error. In the interest of finer differentiation, TPM_RC_VALUE should be returned if the failure is caused by the *size* parameter or the *offset* parameter and TPM_RC_NV_RANGE should be used to indicate a failure caused by the combination of *size* and *offset*.

In addition, TPM2_PolicyNV and TPM2_PolicyCounterTimer may also return the response code TPM_RC_VALUE if an offset check fails. This return code is neither captured in the error return code table, nor mentioned in the description of these commands in Part 3.

### 2.30.6 Authorization Checks

In Part 3, section 5.6 Authorization Checks, Paragraph d, e, and f document incorrect error codes.

Paragraph d) indicates that if the command requires a handle to have DUP role authorization, then the associated authorization session is a policy session (TPM_RC_POLICY_FAIL). The correct response for this error is TPM_RC_AUTH_TYPE as implemented in the reference code.

Paragraph e; 1) indicates that if the command requires a handle to have ADMIN role authorization and if the entity being authorized is an object and its adminWithPolicy attribute is SET, or a hierarchy, then the authorization session is a policy session (TPM_RC_POLICY_FAIL). The correct response for this error is TPM_RC_AUTH_TYPE as implemented in the reference code.

Paragraph f; 1) indicates that if the command requires a handle to have USER role authorization and if the entity being authorized is an object and its userWithAuth attribute is CLEAR, then the associated authorization session is a policy session (TPM_RC_POLICY_FAIL). The correct response for this error is TPM_RC_AUTH_UNAVAILABLE as implemented in the reference code.

The authorization checks are done in SessionProcess.c, function CheckAuthSession() (Part 4, section 6.4.4.8).

### 2.30.7 TPM2_ECDH_ZGen, TPM2_ECDH_KeyGen

According to Part 3, section 14.5 TPM2_ECDH_ZGen the parameter *keyHandle* shall refer to a loaded, ECC key (TPM_RC_KEY) with the *restricted* attribute CLEAR and the *decrypt* attribute SET (TPM_RC_ATTRIBUTES). The reference code returns TPM_RC_KEY in both cases.

When the key selected has the wrong attributes the preferred error code is TPM_RC_ATTRIBUTES. However, TPM_RC_KEY is also acceptable. The same applies to TPM2_ECDH_KeyGen.

### 2.30.8 Handle Area Validation

In Part 3, section 5.4 Handle Area Validation, paragraph b; 4) the text indicates that if a handle in the handle area of a command references a session, then the session context shall be present in TPM memory (TPM_RC_REFERENCE_S0 + N).

The correct response for this error is (TPM_RC_REFERENCE_H0 + N) as implemented in the reference code.

### 2.30.9 TPM2_Commit

According to Part 3, section 19.2.1 TPM2_Commit the *signHandle* parameter shall refer to an ECC key with the sign attribute (TPM_RC_ATTRIBUTES) and the signing scheme must be anonymous (TPM_RC_SCHEME). In addition, the error return code table of this command defines the error condition for TPM_RC_ATTRIBUTES as "*keyHandle* references a restricted key that is not a signing key".

Both statements are incorrect. The only requirement for *signHandle* is that the signing scheme must be anonymous. The reference code implements the checks for the key correctly.

### 2.30.10 Password authorization

According to Part 1, section 18.6.1, a password authorization may not be used for anything but authorization and the TPM will return an error (TPM_RC_ATTRIBUTES) if *encrypt*, *decrypt*, or *audit* is SET in a password authorization.

The check for these attributes is missing in the session processing code of the reference implementation. Therefore a TPM might not return an error if *encrypt*, *decrypt*, or *audit* is SET in a password authorization.

### 2.30.11 TPM2_EC_Ephemeral

In the reference implementation, Part 3, section 19.3, TPM2_EC_Ephemeral does not return the result of the point multiply operation. If the point multiply operation fails because the result is a point at infinity (which has a statistically insignificant chance of occurring), no error code is returned, The values returned for the public key Q in the response might be undetermined.

To fix this, TPM2_EC_Ephemeral should return the result of the point multiply operation. The error return code table in section 19.3.3 should indicate that TPM_RC_NO_RESULT is returned if the TPM is not able to generate an ephemeral key $r$ or the point multiply $[r]G$ produced a point at infinity.

### 2.30.12 TPM2_RSA_Decrypt

In Part 3, 14.3 TPM2_RSA_Decrypt, the error return code table indicates that TPM_RC_KEY is returned if *keyHandle* does not reference an unrestricted decrypt key.

The reference implementation returns TPM_RC_KEY if *keyHandle* does not reference an RSA key and TPM_RC_ATTRIBUTES if *restricted* is SET or if *decrypt* is CLEAR.

The error return code table should be fixed to match the implementation.

### 2.30.13 TPM2_Sign/ Attestation commands

TPM2_Sign (Part 3, 20.2), as well as the attestation commands (Part 3, 18) should return TPM_RC_KEY if the signing key references a symmetric block cipher (TPM_ALG_SYMCIPHER) key.

## 2.31 Size Checks

### 2.31.1 CryptParameterEncryption/Decryption [code]

The functions CryptParameterEncryption() and CryptParameterDecryption() in the reference code in Part 4, 10.2.9.8 and 10.2.9.9 do not correctly check the size of the parameter buffer to be encrypted or decrypted. To fix the issue, the functions should be corrected to check that the parameter buffer (a TPM2B type field) is at least 2 bytes in length and should use the function UINT16_Unmarshal() to read the size of the buffer instead of BYTE_ARRAY_TO_UINT16().

The fixed CryptParameterDecryption() function will return TPM_RC_INSUFFICIENT if the input buffer does not contain enough data to read the UINT16 size field.

The fixed CryptParameterEncryption() function will enter failure mode and return TPM_RC_FAILURE if the internal response buffer does not contain enough data for the UINT16 size field.

### 2.31.2 TPM2_PolicyAuthorize [code]

TPM2_PolicyAuthorize() in the reference code in Part 3, 23.16 does not correctly check the size of the *keySign* parameter. To fix the issue, the TPM will check that *keySign* (a TPM2B type field) is at least 2 bytes in length or otherwise return TPM_RC_INSUFFICIENT.