TRUSTED COMPUTING GROUP

ERRATA

Errata for TCG Trusted Platform Module Library

Family “2.0”
Level 00 Revision 01.59
November 8, 2019

Version 1.8
August 11, 2026

Contact: admin@trustedcomputinggroup.org

Published

DISCLAIMERS, NOTICES, AND LICENSE TERMS

CHANGE HISTORY

VERSION	DATE	DESCRIPTION
1.0	December 18, 2019	Added • 2.1 TPM_SPEC Date Constants [specification text, code] • 2.2 Non-orderly Shutdown – failedTries [code]
1.1	June 18, 2020	Added • 2.3 ACT preserveSignaled [specification text, code]
1.2	November 10, 2020	Added • 2.4 RSAES_Decode - padding [code]
1.3	March 11, 2022	Added • 2.5 TPM_EO – two's complement [code] • 3.1.1 TPM2_NV_WriteLock, TPM2_NV_ReadLock [specification text]
1.4	January 9, 2023	Added • 2.6.1 CryptParameterEncryption/ Decryption [code] • 2.6.2 TPM2_PolicyAuthorize [code] • 2.6.3 CryptGenerateKeyDes [code]
1.5	January 25, 2024	Added • Disclaimers about proper use of CreateLoaded
1.7	June 10, 2025	Added • 2.7 CryptHmacSign [code] • 3.3 1 TPM2_NV_Certify with signHandle = NULL [specification text, code] • 3.3.2 TPM2_Quote with signHandle = NULL [specification text, code]
1.8	February 1, 2026	Added • 2.8 Sequence object • 2.9 TPM2_ActivateCredential label • 2.10 RSA-OEAP decode • 2.11 ObjectFlush

CONTENTS

1 Introduction

This document describes errata and clarifications for the TCG Trusted Platform Module Library Family “2.0” Level 00 Revision 01.59 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.

The heading of each errata in section 2 indicates whether the errata affects the specification text or the reference code implementation. This is indicated by the word “specification text” or “code” in square brackets (“[]”).

2 Errata

2.1 TPM_SPEC Date Constants [specification text, code]

Table 6 in Part 2, 6.1 TPM_SPEC (Specification Version Values) should be replaced with:

Table 6 — 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	159	the version number of the spec (001.59 * 100)
TPM_SPEC_YEAR	2026	the year of the version
TPM_SPEC_DAY_OF_YEAR	32	the day of the year (February 1)

NOTE The date in Table 6 reflects an internal publication date for Errata 1.8.

That is, the spec date fields TPM_SPEC_YEAR and TPM_SPEC_DAY_OF_YEAR should be set to the date of the most recent Errata document that revised the behavior of the TPM.

2.2 Non-orderly Shutdown - *failedTries* [code]

The following text in Part 1, 19.8.6 Non-orderly Shutdown describes the reference code implementation of TPM2_Startup() after a non-orderly Shutdown:

An alternative implementation sets an NV flag indicating that access to a DA protected object occurred during this boot cycle. After a non-orderly restart, if the flag is set, the TPM increments *failedTries* and clears the flag. If the flag is clear, there is no need to increment *failedTries*.

EXAMPLE This handles the case where a platform repeatedly does a non-orderly shutdown, possibly due to a low battery. Without the flag, *failedTries* would increment on each reboot and the TPM would go into lockout.

The reference code does not correctly implement the behavior described above if a DA protected object is accessed after a TPM2_Shutdown(). In this case, the NV flag (indicating that access to a DA protected object occurred during this boot cycle) is not set correctly. When a power loss happens, *failedTries* is not incremented on the next TPM2_Startup(). The reference code should be fixed.

The check and increment of *failedTries* on TPM2_Startup() ensures that a failed authorization attempt is recorded by the TPM (e.g. because NV memory is unavailable).

2.3 ACT *preserveSignaled* [specification text, code]

The ACT *preserveSignaled* attribute is incorrectly described in the Library Spec Part 2 and 3, and is incorrectly implemented in the reference code in Part 4, 7.8.3.2 ActStartup(). The reference code always returns zero for the *preserveSignaled* attribute.

In Part 2, 8.12 TPMA_ACT, the following text should be added to the description of the ACT attribute structure.

The *preserveSignaled* action over a power cycle is:

- Cold (with power loss between Shutdown and Startup) TPM Reset, TPM Restart, TPM Resume
  - *preservedSignaled* is set to CLEAR
- Warm (no power loss between Shutdown and Startup) TPM Reset, TPM Restart, TPM Resume
  - *preserveSignaled* holds the state of *signaled* before the power cycle

NOTE 1: *preserveSignaled* allows startup software to determine if the startup cycle was likely initiated by an ACT event. If power was lost, it doesn't care.

In Part 2, 8.12 TPMA_ACT, Table 40 should be replaced with the following table:

Table 40 — Definition of (UINT32) TPMA_ACT Bits

Bit	Name	Definition
0	signaled	**SET (1):** The ACT has signaled **CLEAR (0):** The ACT has not signaled
1	preserveSignaled	Preserves the state of *signaled*, depending on the power cycle
31:2	Reserved	shall be zero

In Part 3, 9.3 TPM2_Startup, in the general description of the command actions on TPM Reset and on TPM Restart, the following bullet point should be changed.

From:

- For each ACT the timeout is reset to zero, the *signaled* attribute is set to CLEAR (if *preserveSignaled* is CLEAR), and the *authPolicy* is set to the Empty Buffer and its hashAlg is set to TPM_ALG_NULL.

To:

- For each ACT the timeout is reset to zero, the *signaled* attribute is set to CLEAR, its *authPolicy* is set to the Empty Buffer, and its hashAlg is set to TPM_ALG_NULL.

That is, the condition in brackets "(if *preserveSignaled* is CLEAR)" should be removed.

In Part 3, 32.2 TPM2_ACT_SetTimeout, in the general description, the following sentence should be added:

When this command is successful, *preserveSignaled* will be CLEAR.

The reference code should be fixed following the (above) corrections in Part 2 and 3.

2.4 RSAES_Decode - padding [code]

The RSAES_Decode() function in Part 4, 10.2.17.4.12, which performs the decoding for RSAES-PKCS1-V1_5-DECRYPT as defined in PKCS#1V2.1, behaves incorrectly if the padding string (PS) is 7 octets.

According to PKCS#1V2.1, the length of the padding string (PS) must be at least 8 octets. However, the RSAES_Decode() reference code function permits a padding string of 7 bytes. The RSAES_Decode() function should be corrected to reject a padding string that is less than 8 bytes.

2.5 TPM_EO – two's complement [code]

According to Part 3, 23.9 TPM2_PolicyNV and 23.10 TPM2_PolicyCounterTimer,

"The signed arithmetic operations are performed using twos-complement."

For two negative values, the reference code (in Part 4, 9.11.2.2 SignedCompareB()) implements the signed arithmetic operations using sign-magnitude. The reference code should be fixed to match the description in Part 3.

2.6 Size Checks

2.6.1 CryptParameterEncryption/Decryption [code]

The functions CryptParameterEncryption() and CryptParameterDecryption() in the reference code in Part 4, 10.2.6.6.5 and 10.2.6.6.6 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.

2.6.2 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.

2.6.3 CryptGenerateKeyDes [code]

The function CryptGenerateKeyDes() in the reference code in Part 4, 10.2.9.2.3 does not correctly check the symmetric key size provided in the *sensitive* parameter. To fix the issue, the function will check that the size of the requested TDES key is a multiple of 8 bytes or otherwise the TPM will return TPM_RC_SYMMETRIC.

2.7 CryptHmacSign [code]

The function CryptHmacSign() in the reference code in Part 4, 10.2.6.3.1 does not check that the TPMT_SIGNATURE provided in the *signature* parameter has the expected *sigAlg* selector value. To fix the issue, the function will check that *signature*'s *sigAlg* is TPM_ALG_HMAC or otherwise the TPM will return TPM_RC_SCHEME.

2.8 Sequence object [code]

The reference code for Part 3 does not check if certain handle variables reference a sequence object. The affected cases are:

- *activateHandle* in TPM2_ActivateCredential(),
- *objectHandle* in TPM2_Certify() and TPM2_CertifyCreation(), and
- *sendObject* in TPM2_AC_Send().

To fix the issue, the commands will check that the object referenced by these handles is not a (hash, MAC, or Event) sequence object or otherwise the TPM will return TPM_RC_TYPE or TPM_RC_SEQUENCE.

A minimal patch that adds these checks in the reference code is:

In TPM2_Certify:

```
     if(!CryptSelectSignScheme(signObject, &in->inScheme))
         return TPM_RCS_SCHEME + RC_Certify_inScheme;

+    // Cannot certify a sequence object
+    if(ObjectIsSequence(certifiedObject))
```



```
+        return TPM_RCS_TYPE + RC_Certify_objectHandle;
+
     // Command Output
     // Filling in attest information
     // Common fields
```

In TPM2_CertifyCreation:

```
     OBJECT*          certified  = HandleToObject(in->objectHandle);
     OBJECT*          signObject = HandleToObject(in->signHandle);
     // Input Validation
+    // Cannot certify creation of a sequence object
+    if(ObjectIsSequence(certified))
+        return TPM_RCS_TYPE + RC_CertifyCreation_objectHandle;
+
     if(!IsSigningObject(signObject))
         return TPM_RCS_KEY + RC_CertifyCreation_signHandle;
     if(!CryptSelectSignScheme(signObject, &in->inScheme))
```

In TPM2_ActivateCredential:

```
    if(!CryptIsAsymAlgorithm(object->publicArea.type)
       || object->publicArea.objectAttributes.decrypt == CLEAR
       || object->publicArea.objectAttributes.restricted == CLEAR)
        return TPM_RCS_TYPE + RC_ActivateCredential_keyHandle;
+
+    // Cannot activate credential of sequence object
+    if(ObjectIsSequence(activateObject))
+        return TPM_RCS_TYPE + RC_ActivateCredential_activateHandle;
+
     // Command output
     // Decrypt input credential data via asymmetric decryption.  A
     // TPM_RC_VALUE, TPM_RC_KEY or unmarshal errors may be returned at this
     // point
```

In TPM2_AC_Send:

```
      // permanent handle.
      else if(HandleGetType(in->authHandle) != TPM_HT_PERMANENT)
          return TPM_RCS_HANDLE + RC_AC_Send_authHandle;
 +
 +    // Cannot send a sequence object to an attached component
 +    if(ObjectIsSequence(object))
 +        return TPM_RCS_TYPE + RC_AC_Send_sendObject;
      // Make sure that the object to be duplicated has the right attributes
      if(IS_ATTRIBUTE(
             object->publicArea.objectAttributes, TPMA_OBJECT, encryptedDuplication)
```

2.9 TPM2_ActivateCredential label

TPM2_ActivateCredential() may support the label "IDENTITY2" in addition to "IDENTITY" as defined in Part 1, B.10.4 and C.6.4 to decrypt the credential blob to identify that a TPM firmware version has fixed the errata documented in section 2.8 Sequence object. The use of “IDENTITY2” is not intended to be a general change to

TPM2_ActivateCredential and is not expected to be added as an alternative in future versions of the TPM Specification.

2.10 RSA-OAEP decode [code]

In Part 4, the reference code uses a timing-dependent implementation for RSA-OAEP decryption. RSA private key operations (including padding) should be provided by an implementation's crypto library. Cryptographic libraries are expected to take special care to avoid timing side-channels like this. A minimal patch that significantly reduces (but does not totally eliminate) secret-dependent execution time in OAEP decryption in the reference code is:

```
static TPM_RC
OaepDecode(
    TPM2B           *dataOut,       // OUT: the recovered data
    TPM_ALG_ID       hashAlg,       // IN: algorithm to use for padding
    const TPM2B     *label,         // IN: null-terminated string (may be NULL)
    TPM2B           *padded         // IN: the padded data
    )
{
    UINT32       i;
    BYTE         seedMask[MAX_DIGEST_SIZE];
    UINT32       hLen = CryptHashGetDigestSize(hashAlg);
    BYTE         mask[MAX_RSA_KEY_BYTES];
    BYTE        *pp;
    BYTE        *pm;
    TPM_RC       retVal = TPM_RC_SUCCESS;
    // Strange size (anything smaller can't be an OAEP padded block)
-   // Also check for no leading 0
-   if((padded->size < (unsigned)((2 * hLen) + 2)) || (padded->buffer[0] != 0))
+   if(padded->size < (unsigned)((2 * hLen) + 2))
        ERROR_RETURN(TPM_RC_VALUE);
...
    if(memcmp(seedMask, mask, hLen) != 0)
        ERROR_EXIT(TPM_RC_VALUE);

+
+    // Check for the leading 0x00 byte.
+    if(padded->buffer[0] != 0)
+       ERROR_RETURN(TPM_RC_VALUE);

    // find the start of the data
    pm = &mask[hLen];
...
```

2.11 Object Flush [code]

The reference code only marks the occupied flag in an OBJECT structure during ObjectFlush. The implementation should entirely clear the structure to ensure no potentially sensitive information remains. A patch for the reference code is below.

```
 void ObjectFlush(OBJECT* object)
 {
+    MemorySet(object, 0, sizeof(*object));
     object->attributes.occupied = CLEAR;
 }
```


3 Clarifications

3.1 Error Codes

3.1.1 TPM2_NV_WriteLock, TPM2_NV_ReadLock [specification text]

There is an ambiguity in the command descriptions of TPM2_NV_WriteLock and TPM2_NV_ReadLock: whether the TPM shall return success or an (authorization) error code if TPM2_NV_WriteLock or TPM2_NV_ReadLock is executed with improper authorization for an NV index that is already write or read-locked.

The description in Part 3, 31.11 TPM2_NV_WriteLock should be interpreted as if

> TPM2_NV_WriteLock may either return TPM_RC_SUCCESS or TPM_RC_NV_AUTHORIZATION if TPMA_NV_WRITELOCKED for the NV Index is already SET, and proper write authorization (as determined by TPMA_NV_PPWRITE, TPMA_NV_OWNERWRITE, TPMA_NV_AUTHWRITE, and the *authPolicy* of the NV Index)

is not provided.

The description in Part 3, 31.14 TPM2_NV_ReadLock should be interpreted as if

> TPM2_NV_ReadLock may either return TPM_RC_SUCCESS or TPM_RC_NV_AUTHORIZATION if TPMA_NV_READLOCKED for the NV Index is already SET, and proper read authorization (as determined by TPMA_NV_PPREAD, TPMA_NV_OWNERREAD, TPMA_NV_AUTHREAD, and the *authPolicy* of the NV Index)

is not provided.

3.2 Notes

3.2.1 TPM2_CreateLoaded [specification text]

3.2.1.1 Command Description

The following note should be added to Part 3, 12.9.1 General Description:

NOTE If parentHandle references a Derivation Parent, the bits of the Label and Context are used in the creation of the key. This differs from TPM2_CreatePrimary(), where the bits of the template are used. This means that different templates (specifically, different public attributes) will result in the same key.

3.2.1.2 Derivation Parameters

The following text should be added to Part 1, 28.2 Derivation Parameters:

Since TPM2_CreateLoaded() does not use the public attributes in the KDF, it can create child keys with the same private key but different attributes.

NOTE TPM2_PolicyTemplate() on the parent can be used to restrict the child attributes.

EXAMPLE Once the parent is duplicated, one TPM can derive a key that can only be used for encryption and a different TPM can derive the same key that is restricted to be used for decryption. Or an HMAC key can be restricted to signing on one TPM and verification on another.

3.2.1.3 Entropy for Derived Objects

The following section should be added to Part 1, 28.4 Entropy for Derived Objects:

Caution on use of Derivation Parents

Users of Derived Objects are advised to ensure that *label* and *context* are not re-used between different objects derived from the same Derivation Parent.

If the same Derivation Parent, *label*, and *context* are provided in two different invocations of CreateLoaded, the Derived Objects resulting from the derivation will share the same keying material (that is, output from the KDF used to create the Derived Object's sensitive and seedValue). This is true even if two different templates are provided to CreateLoaded.

Authorization values and/or policies can be used to protect Derivation Parents from misuse by attackers. TPM2_PolicyTemplate can be used to restrict the template(s) that can be used with a given Derivation Parent.

Although the TPM can produce attestations of Derived Objects (e.g., with TPM2_Certify), these attestations are untrustworthy because sensitiveDataOrigin can never be SET for a Derived Object. Verifiers should always ensure that sensitiveDataOrigin is SET for attested objects.

3.3 Digest computation when signHandle is NULL

3.3.1 TPM2_NV_Certify with signHandle = NULL [specification text, code]

If TPM2_NV_Certify is used to certify the hash digest of the NV Index data content (i.e., *size* and *offset* are both zero) and *signHandle* is TPM_RH_NULL, then this command can either:

- Option 1: If the signing scheme or the scheme hash algorithm is not TPM_ALG_NULL, compute *nvDigest* using the hash algorithm of the selected scheme. Otherwise, return TPM_RC_SCHEME.
- Option 2: Set *nvDigest* in the TPMS_DIGEST_CERTIFY_INFO structure to Empty Buffer.

The reference code sets *nvDigest* to Empty Buffer (option 2), while the description in Part 3, section 18.1 Introduction (of Attestation Commands) indicates that all of the actions of the command are performed and only the signature is absent (option 1).

A future TPM Library specification version (greater than 1.83) may only allow option 1.

3.3.2 TPM2_Quote with signHandle = NULL [specification text, code]

According to Part 3, section 18.4 (NOTE 2 and 3) TPM2_Quote allows two different behaviors if *signHandle* set to TPM_RH_NULL:

- Option 1: If the signing scheme or the scheme hash algorithm is not TPM_ALG_NULL, compute pcr*Digest* using the hash algorithm of the selected scheme (and return the TPMS_ATTEST structure). Otherwise, return TPM_RC_SCHEME.
- Option 2: Optionally, always return TPM_RC_SCHEME.

The reference code implements option 2.

A future TPM Library specification version (greater than 1.83) may only allow option 1.

