TRUSTED® COMPUTING GROUP

ERRATA

Errata for TCG Trusted Platform Module Library v185

Version 1
March 12, 2026

Contact: admin@trustedcomputinggroup.org

TCG Published

DISCLAIMERS, NOTICES, AND LICENSE TERMS

CHANGE HISTORY

Version	Date	Description
1	2026/02/10	Initial version

Contents

1 Introduction

This document describes errata and clarifications for the TCG Trusted Platform Module Library Family Version 185 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 Clause 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]

The table in Part 2, 6.1 “TPM_SPEC (Specification Version Values)” should be replaced with

Table 1: 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	185	the version number of the specification
TPM_SPEC_YEAR	0	shall be zero ***Note:*** Prior to version 185, this constant reported the year in which the TPM Library Specification indicated by TPM_SPEC_VERSION (or an Errata) was published. It is set to 0 to indicate that the errata level is reported in TPM_SPEC_ERRATA.
TPM_SPEC_ERRATA	1	shall be 1 for a TPM implementing this version of the errata (this document) ***Note:*** Prior to version 185, this constant was called TPM_SPEC_DAY_OF_YEAR.

2.2 TPM2_SignSequenceStart() checks for *x509sign* [specification text]

Part 3, TPM2_SignSequenceComplete() says:

“The x509sign attribute of keyHandle must not be SET (TPM_RC_ATTRIBUTES).”

The description of TPM2_SignSequenceStart() should reflect that it is acceptable for an implementation to perform this attribute check at the time of TPM2_SignSequenceStart() rather than waiting for TPM2_SignSequenceComplete(), since TPM2_SignSequenceStart() and TPM2_SignSequenceComplete() assert that the same key attributes were provided at the beginning and end of the sequence. This allows for fail-fast behavior.

2.3 TPMU_SIG_SCHEME cases for ML-DSA [specification text]

Part 2, TPMU_SIG_SCHEME does not list a structure for the TPM_ALG_MLDSA and TPM_ALG_HASH_MLDSA cases. This table should be interpreted as if the contents of each of the union elements associated with these algorithms is a TPMS_EMPTY.

2.4 TPM_RC_NO_RESULT emitted during ML-DSA rejection sampling [specification text]

FIPS 204 ([1]), footnote 10 and Appendix C describe an implementation-specific limit on the number of iterations allowed by the rejection-sampling loop used during ML-DSA signing. TPM_RC_NO_RESULT should be interpreted to be the expected error indication for this case, and any sessions should be left in their initial state as if the command had never been executed.

2.5 Ticket produced by TPM2_VerifyDigestSignature() for ML-DSA [specification text]

If the TPM supports external Mu (μ), and *allowExternalMu* is SET on the verification key, then TPM2_VerifyDigestSignature() can be used to verify the signature over Mu (the message representative).

TPM2_VerifyDigestSignature() produces a TPMT_TK_VERIFIED that contains the digest that was verified. Unlike the digests verified by RSA and ECC keys, a Mu value is a digest over multiple values (i.e., the public key hash tr as well as the message that was verified). Consequently, an external Mu TPMT_TK_VERIFIED is not useful to the one command (TPM2_PolicyAuthorize()) that accepts it as an input, because TPM2_PolicyAuthorize() cannot re-compute the tr hash in order to verify the Mu value is correct for the message.

To avoid returning a nonviable ticket to the user, it is preferable for the TPM to return a NULL TPMT_TK_VERIFIED ticket instead.

2.6 Hash algorithm used by TPM2_Quote() for signing keys without a scheme hash [specification text]

TPM2_Quote() creates and signs a TPMS_QUOTE_INFO containing a *pcrDigest*, the digest of the concatenation of the selected PCRs.

The command description in Part 3 says:

“The TPM will hash the list of PCR selected by *PCRselect* using the hash algorithm in the selected signing scheme.”

It should be updated for signatures with no hash algorithm parameter, such as (Pure) ML-DSA and EdDSA.

For signatures with no scheme hash, the TPM will compute *pcrDigest* using the Name algorithm of the signing key. If the Name algorithm is TPM_ALG_NULL, then the TPM shall return TPM_RC_SCHEME.

2.7 Hash algorithm used by TPM2_NV_Certify() for signing keys without a scheme hash [specification text]

If *size* and *offset* are both 0, TPM2_NV_Certify() creates and signs a TPMS_NV_DIGEST_CERTIFY_INFO containing a *nvDigest*, the digest of the contents of the NV index.

The command description in Part 3 says:

“If *size* and *offset* are both zero and *signHandle* is TPM_RH_NULL, the digest is computed using the hash algorithm provided in *inScheme*, unless the scheme or hash algorithm is TPM_ALG_NULL, in which case the TPM shall return TPM_RC_SCHEME.”

It should be updated for signatures with no hash algorithm parameter, such as (Pure) ML-DSA and EdDSA.

For signatures with no scheme hash, the TPM will compute *nvDigest* using the Name algorithm of the signing key. If the Name algorithm is TPM_ALG_NULL, then the TPM shall return TPM_RC_SCHEME.

2.8 Supporting TPMS_NV_CERTIFY_INFO with TPM2_NV_Certify() for ML-DSA [specification text]

If *size* and *offset* are not both 0, TPM2_NV_Certify() creates and signs a TPMS_NV_CERTIFY_INFO containing a *buffer* of size up to MAX_NV_BUFFER_SIZE. This structure is large, and ML-DSA signatures are also large.

To avoid an excessively long response, it is permitted for the TPM to return TPM_RC_SIZE (indicating the *size* parameter) in the case of a large NV index. This indicates that only digest-based NV certification is available (i.e., by setting both *size* and *offset* to 0) with ML-DSA keys on this TPM.

3 Clarifications

3.1 EdDSA validation criteria

All standards-compliant EdDSA signers will produce signatures that are correctly validated (or rejected) by all standards-compliant EdDSA verifiers. However, it is possible for a non-standards-compliant (a.k.a. “non-honest”) signer to produce a signature that will be accepted by some verifiers and rejected by others (e.g. in `TPM2_PolicySigned()`).

It is not practical for the TPM Library to specify one set of validation criteria for all TPMs, because TPM implementations typically bring their own cryptographic libraries. However, we have the following recommendations for implementations in order to promote consistent behavior:

- Prefer the “cofactorless” equation (permitted in both FIPS 186-5 [2] and RFC 8032 [3]).
- As required by both FIPS 186-5 [2] and RFC 8032 [3], reject the signature if the second half `"S"` is not less than the order of the base point. The order is called `"n"` in FIPS 186-5 [2] and `"L"` in RFC 8032 [3].
- As required by both FIPS 186-5 [2] and RFC 8032 [3], reject non-canonically encoded curve points (i.e., compressed y-values greater than or equal to the prime order `"p"` of the scalar field).

We caution users to consider the implications of various EdDSA implementations when deciding where to perform EdDSA verification.

3.2 On the use of restricted MAC keys for non-attestation messages

TPM 2.0 185 deprecated the `TPM2_Sign()` command in favor of the new `TPM2_SignDigest()`, `TPM2_SignSequenceStart()`, and `TPM2_SignSequenceComplete()` commands, which can be used for *nearly* all applications that `TPM2_Sign()` could be used for.

MAC keys are an exception: the new sequenced signing commands (`TPM2_SignSequenceStart()` and `TPM2_SignSequenceComplete()`), along with the existing MAC commands, are intended for use with non-*restricted* MAC keys.

However, *restricted* MAC keys cannot be used with `TPM2_SignSequenceStart()` and `TPM2_SignSequenceComplete()` because they do not take a `TPMT_TK_HASHCHECK` ticket, which proves that the digest is not the digest of a false attestation statement.

`TPM2_Sign()`, which is deprecated, remains the only way to MAC a provided digest with a restricted MAC key.

We note that the primary use-case for allowing signing of non-attestation messages with *restricted* keys is certificate signing requests, which is not a typical workflow for MAC keys.

References

[1] "Module-Lattice-Based Digital Signature Standard." National Institute of Standards and Technology, Aug. 2024. Available: https://doi.org/10.6028/NIST.FIPS.204

[2] "FIPS 186-5: Digital Signature Standard (DSS)." National Institute of Standards and Technology, Feb. 2023. Available: https://doi.org/10.6028/NIST.FIPS.186-5

[3] S. Josefsson and I. Liusvaara, "RFC 8032: Edwards-Curve Digital Signature Algorithm (EdDSA)." RFC Editor, Jan. 2017. Available: https://www.rfc-editor.org/info/rfc8032

