ERRATA

ERRATA

Errata Version 1.0
January 16, 2017

FOR

TCG Trusted Platform Module Library

Specification Version 2.0
Revision 1.38
September 29, 2016

Contact: admin@trustedcomputinggroup.org

TCG Published

Disclaimers, Notices, and License Terms

CONTENTS

1. Introduction

This document describes errata and clarifications for the TCG Trusted Platform Module Library Version 2.0 Revision 1.38 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 Object Derivation

2.1.1 Introduction

This section summarizes errata with regards to Object Derivation in TPM2_CreateLoaded(). For interoperability of Derived Objects, it is essential that all parties, given the same Derivation Parent and the same Derivation Parameters, derive the same key. Therefore, external software that uses the Library Spec reference code to implement Object Derivation outside of the TPM needs to consider the code fixes in this section as well.

2.1.2 Incorrect KDF Seed

The reference code in Part 3, 12.9 TPM2_CreateLoaded uses an incorrect key in the key derivation function (KDF) to generate a Derived Object. The reference code uses the Derivation Parent's *seedValue* instead of the Derivation Parent's *sensitive* value. This affects the key generation of all types of Derived Objects (TPM_ALG_SYMCIPHER, TPM_ALG_KEYEDHASH, and TPM_ALG_ECC).

This issue is caused by an incorrect parameter in the function call to DRBG_InstantiateSeededKdf(). To fix this, the seed used for the KDF should be replaced with the *sensitive* value (see code fix in 2.1.2.1). The correct KDF parameters for Object derivation are specified in Part 1, 28.4 Entropy for Derived Objects.

2.1.2.1 Code Fix

Part 3, 12.9.3 Detailed Actions (of TPM2_CreateLoaded), line 73

```
        DRBG_InstantiateSeededKdf((KDF_STATE *)rand,
                                  scheme->details.xor.hashAlg,
                                  scheme->details.xor.kdf,
-                                 &parent->sensitive.seedValue.b,
+                                 &parent->sensitive.sensitive.bits.b,
                                  &publicArea->unique.derive.label.b,
                                  &publicArea->unique.derive.context.b);
```

2.1.3 Incorrect Label and Context

The reference code in Part 3, 12.9 TPM2_CreateLoaded does not correctly include *label* and *context* in the key derivation function (KDF) when a Derived Object of the type TPM_ALG_ECC is generated.

The reference code reuses the *unique* field in the public area of the object to store the *label* and *context* parameters that are provided by the caller. However, the *unique* field is also used during the key generation to output the ECC public key. As a result, the *label* and *context* values are overwritten and incorrect parameters are used in the derivation of the *sensitive* value and *seedValue*. To fix this, a separate structure variable needs to be allocated to store *context* and *label* (see code fixes in 2.1.3.1).

2.1.3.1 Code Fix

Part 3, 12.9.3 Detailed Actions (of TPM2_CreateLoaded), line 16

```
    TPMT_PUBLIC                 *publicArea;
```

```
     RAND_STATE                   randState;
     RAND_STATE                  *rand = &randState;
+    TPMS_DERIVE                  labelContext;
// Input Validation
```

Part 3, 12.9.3 Detailed Actions (of TPM2_CreateLoaded), line 66

```
             return RcSafeAddToResult(result, RC_CreateLoaded_inPublic);
         // Process the template and sensitive areas to get the actual 'label' and
         // 'context' values to be used for this derivation.
-        result = SetLabelAndContext(publicArea, &in->inSensitive.sensitive.data);
+        result = SetLabelAndContext(&labelContext, publicArea,
+                                    &in->inSensitive.sensitive.data);
         if(result != TPM_RC_SUCCESS)
             return result;
         // Set up the KDF for object generation
```

Part 3, 12.9.3 Detailed Actions (of TPM2_CreateLoaded), line 73

```
         DRBG_InstantiateSeededKdf((KDF_STATE *)rand,
                                   scheme->details.xor.hashAlg,
                                   scheme->details.xor.kdf,
                                   &parent->sensitive.sensitive.bits.b,
-                                  &publicArea->unique.derive.label.b,
-                                  &publicArea->unique.derive.context.b);
+                                  &labelContext.label.b,
+                                  &labelContext.context.b);
         // Clear the sensitive size so that the creation functions will not try
         // to use this value.
         in->inSensitive.sensitive.data.t.size = 0;
```

Part 4, 7.6.3.18 SetLabelAndContext(), line 1070

```
 TPM_RC
 SetLabelAndContext(
+    TPMS_DERIVE             *labelContext,  // OUT: the recovered label and context
     TPMT_PUBLIC             *publicArea,    // IN/OUT: the public area containing
                                             //         the unmarshaled template
     TPM2B_SENSITIVE_DATA    *sensitive      // IN: the sensitive data
```

Part 4, 7.6.3.18 SetLabelAndContext(), line 1077

```
     TPM_RC                   result;
     INT32                    size;
     BYTE                    *buff;
-    TPM2B_LABEL              label;
+//
+    // In case neither the sensitive nor publicArea have a label or a context
+    labelContext->label.b.size = 0;
+    labelContext->context.b.size = 0;

     // Unmarshal a TPMS_DERIVE from the TPM2B_SENSITIVE_DATA buffer
-    size = sensitive->t.size;
     // If there is something to unmarshal...
-    if(size != 0)
+    if(sensitive->t.size != 0)
     {
```

```
+       size = sensitive->t.size;
        buff = sensitive->t.buffer;
-       result = TPM2B_LABEL_Unmarshal(&label, &buff, &size);
-       if(result != TPM_RC_SUCCESS)
-           return result;
-       // If there is a label in the publicArea, it overrides
-       if(publicArea->unique.derive.label.t.size == 0)
-           MemoryCopy2B(&publicArea->unique.derive.label.b, &label.b,
-                        sizeof(publicArea->unique.derive.label.t.buffer));
-       result = TPM2B_LABEL_Unmarshal(&label, &buff, &size);
+       result = TPMS_DERIVE_Unmarshal(labelContext, &buff, &size);
        if(result != TPM_RC_SUCCESS)
            return result;
-       if(publicArea->unique.derive.context.t.size == 0)
-           MemoryCopy2B(&publicArea->unique.derive.context.b, &label.b,
-                        sizeof(publicArea->unique.derive.context.t.buffer));
    }
+   // If there is a label string in publicArea, it overrides
+   if(publicArea->unique.derive.label.t.size != 0)
+       MemoryCopy2B(&labelContext->label.b, &publicArea->unique.derive.label.b,
+                    sizeof(labelContext->label.t.buffer));
+   // if there is a context string in publicArea, it overrides
+   if(publicArea->unique.derive.context.t.size != 0)
+       MemoryCopy2B(&labelContext->context.b,
+                    &publicArea->unique.derive.context.b,
+                    sizeof(labelContext->label.t.buffer));
    return TPM_RC_SUCCESS;
}
```

2.1.4 Incorrect Byte Order

When the reference code creates a Derived Object using TPM2_CreateLoaded(), the byte order of the generated *sensitive* value and *seedValue* of the object is processor dependent. With the same Derivation Parent and the same derivation parameters, a different Derived Object is generated on a big endian and little endian TPM. This affects the key generation of all types of Derived Objects (TPM_ALG_SYMCIPHER, TPM_ALG_KEYEDHASH, and TPM_ALG_ECC).

The reference code generates the random bits that are used as secret (ECC private key or symmetric key) of the Derived Object in an internal format (bigNum). When later converted to canonical form (TPM2B), the byte order changes dependent on the endianness of the TPM. To fix this, the random bits in BnGetRandomBits() should be generated in canonical form (TPM2B) and then converted to internal format for processing (see code fix in 2.1.4.1).

2.1.4.1 Code Fix

Part 4, 10.2.4.3.20 BnGetRandomBits(), line 353

```
RAND_STATE      *rand
    )
{
-   n->size = BITS_TO_CRYPT_WORDS(bits);
-   if(n->size > n->allocated)
-       n->size = n->allocated;
-   DRBG_Generate(rand, (BYTE *)n->d, (UINT16)(n->size * RADIX_BYTES));
+   TPM2B_TYPE(LARGEST, LARGEST_NUMBER);
+   TPM2B_LARGEST   large;
+   large.b.size = (UINT16)BITS_TO_BYTES(bits);
+   DRBG_Generate(rand, large.t.buffer, large.t.size);
+   BnFrom2B(n, &large.b);
```

```
    BnMaskBits(n, bits);
    return TRUE;
}
```

2.1.5 Derivation Parameters

Part 1, 28.2 Derivation Parameters contains an incorrect statement which says, “If (*label* or *context* is) provided in the *unique* field, the corresponding value in the *inPrivate.data* field is required to be an empty buffer.”

It should say, “If provided in the *unique* field, the corresponding value in the *inSensitive.data* field is ignored.”

2.2 Attribute Check for KEYEDHASH Objects

It is recommended to add the following attribute check to the reference code in Part 4, 7.6.3.3 CreateChecks().

When a *restricted decrypt* or *restricted sign* TPM_ALG_KEYEDHASH Object is created with *sensitiveDataOrigin* CLEAR (i.e. the sensitive data is provided by the caller), then *fixedParent* and *fixedTPM* are required to be CLEAR, otherwise the TPM will return TPM_RC_ATTRIBUTES.

This attribute check is implemented in the reference code for TPM_ALG_SYMCIPHER Objects, but is missing for TPM_ALG_KEYEDHASH Objects.

2.3 Attribute Check in TPM2_CreatePrimary

The following attribute check is missing in the reference code in Part 3, 24.1 TPM2_CreatePrimary.

When a TPM_ALG_KEYEDHASH or TPM_ALG_SYMCIPHER Object is created using TPM2_CreatePrimary with *sensitiveDataOrigin* CLEAR (i.e. the sensitive data is provided by the caller), then *sensitive.data* must be not empty, otherwise the TPM will return TPM_RC_ATTRIBUTES.

2.4 TPM2_ECC_Parameters

Part 1, C.8 ECC Point Padding contains an inaccurate statement which says, “When the ECC parameters are returned by the command TPM2_ECC_Parameters(), they have to match the exact format as specified in the TCG Algorithm registry.”

Only the numerical values of the ECC curve parameters returned by TPM2_ECC_Parameters() must be the same as listed in the TCG Algorithm Registry. The size may not be the same.

An ECC parameter with a numerical value of zero is incorrectly returned by the reference code as Empty Buffer. It should be returned as a sized buffer with only the data value set to zero.

2.5 TPM2_DictionaryAttackParameters

According to the description and reference code in Part 3, 25.3, TPM2_DictionaryAttackParameters will set the authorization failure count (*failedTries*) to zero.

This is incorrect. TPM2_DictionaryAttackParameters must not set the authorization failure count (*failedTries*) to zero but leave *failedTries* unmodified. As a result, the TPM2_DictionaryAttackParameters() command may cause the TPM to enter lockout. If *maxTries* is changed to a value that is less than the current value of *failedTries*, the TPM goes into lockout until *failedTries* is less than *maxTries*.

EXAMPLE For this example, (m, n) is used as notation for (*maxTries,* decrement rate in minutes). If the parameters are (32, 120) and *failedTries* is 30, and the parameters are changed to (10, 10), then the TPM will be in lockout until *failedTries* counts down to 9 at one count every 10 minutes, or 210 minutes (3.5 hours).

2.6 Self-healing

According to Part 1, 19.8.2 Lockout Mode Configuration Parameters, paragraph a); 2), *failedTries* is decremented by one after *recoveryTime* seconds if there is no power interruption. This is inaccurate and paragraph 2) should be removed.

It is allowed for the self-healing (*failedTries* decrement) to accumulate between TPM Reset, TPM Restart, and TPM Resume. In the current reference implementation, the self-healing does not accumulate between boots because *selfHealTimer* and *lockoutTimer* are stored in volatile memory. Instead these values could be stored in the orderly data structure which is saved to non-volatile memory on each TPM2_Shutdown. When the DA parameters are initialized at TPM2_Startup, credit can be given for the accumulated time.

2.7 TDES Key Parity Calculation

The following description on the parity calculation of TDES keys should be added to Part 1.

A TDES key is generated by getting 24 bytes from the random number generator appropriate for the type of key generation (such as a KDF for a derived key). The 24 bytes are treated as 3, 64-bit values in canonical TPM form (big-endian bytes). The odd parity is then generated for each byte with the parity replacing the least significant bit in each byte to create 3 DES keys. The resulting three DES keys are then validated to make sure that none of them is on the list of prohibited DES key values. If any of the generated key values is prohibited, then the TPM will repeat the key generating process by generating 24 new bytes.

2.8 Mode validation in TPM2_EncryptDecrypt, and TPM2_EncryptDecrypt2

The reference code in Part 3, 15.2 TPM2_EncryptDecrypt and 15.3 TPM2_EncryptDecrypt2 incorrectly validate the mode. If the symmetric mode specified in the *mode* input parameter is TPM_ALG_NULL and the mode of the key is not TPM_ALG_NULL, then the check for the input IV and the input data block size are performed with a wrong mode variable (set to TPM_ALG_NULL instead of the actual value). As a result, the TPM might return TPM_RC_SIZE even though input IV and input data are correctly set for the selected mode.

2.9 TPM2_Import – encryptedDuplication Check

The General Description in Part 3, 13.3 TPM2_Import says, “If *encryptedDuplication* is SET in the object referenced by *parentHandle*, then *encryptedDuplication* shall be SET in *objectPublic* (TPM_RC_ATTRIBUTES).”

In the reference code, TPM2_Load() verifies that if a parent object has *fixedTPM* CLEAR, the child must have the same *encryptedDuplication* value as its parent and otherwise return TPM_RC_ATTRIBUTES. This check may be done at TPM2_Import(). On TPM2_Load() this must be checked unless it was checked at TPM2_Import().

The parent and child object must have the same value for *encryptedDuplication* (both SET or CLEAR) if they are in the same duplication group. All objects in a duplication group are required to have the same setting for *encryptedDuplication*. Therefore, if a parent object has *fixedTPM* CLEAR, the child must have the same *encryptedDuplication* value as its parent.

2.10 Error Codes

2.10.1 Introduction

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

2.10.2 TPM2_StartAuthSession – key scheme

The General Description in Part 3, 11.1 TPM2_StartAuthSession specifies that the TPM shall return TPM_RC_SCHEME if the scheme of the key (referenced by *tpmKey*) is not TPM_ALG_OAEP or TPM_ALG_NULL. However, the reference code returns TPM_RC_VALUE.

The preferred error code for this failure is TPM_RC_VALUE. But TPM_RC_SCHEME is also acceptable.

2.10.3 Lockout Mode

The text in Part 3, 25.1 Introduction of Dictionary Attack Functions says, “While the TPM is in Lockout mode, the TPM will return TPM_RC_LOCKED if the command requires use of an object’s or Index’s *authValue* unless the authorization applies to an entry in the Platform hierarchy.”

The error code should be TPM_RC_LOCKOUT.

2.10.4 NV Locked

In Part 3, 5.4 Handle Area Validation, paragraph b; 3) the text says,

i) If the command requires write access to the index data then TPMA_NV_WRITELOCKED is not SET (TPM_RC_LOCKED)
ii) If the command requires read access to the index data then TPMA_NV_READLOCKED is not SET (TPM_RC_LOCKED)

Both error codes should be TPM_RC_NV_LOCKED.

2.10.5 BnPointMul

In Part 4, 10.2.11.2.19 BnPointMul(), the entry in the return code table for TPM_RC_VALUE is incorrect. It says, TPM_RC_VALUE is returned if “d or u is not 0 < d < n”.

The values for the scalars d and u are allowed to be zero. This type of error is returned if d and u are NULL, S is present but d is NULL, only one of u or Q is present, or the curve parameters are NULL.