1 Introduction

This document specifies conformance clauses in accordance with the OASIS TC Process ([TC-PROC] section 2.2.6 for the KMIP Specification [KMIP-SPEC] for a KMIP server or KMIP client through profiles that define the use of KMIP objects, attributes, operations, message elements and authentication methods within specific contexts of KMIP server and client interaction.

These profiles define a set of normative constraints for employing KMIP within a particular environment or context of use. They may, optionally, require the use of specific KMIP functionality or in other respects define the processing rules to be followed by profile actors.

1.1 IPR Policy

This specification is provided under the RF on RAND Terms Mode of the OASIS IPR Policy, the mode chosen when the Technical Committee was established. For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the TC’s web page (https://www.oasis-open.org/committees/kmip/ipr.php).

1.2 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] and [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.3 Normative References

The fields in structured parameters such as CK_GCM_PARAMS are simply stored sequentially, with any contained byte strings also preceded by a 4-byte big-endian length.

Values and functions that are only meaningful to the API itself are not encapsulated, nor are void pointers that do not have any well-defined meaning. In particular:

- C_Initialize does not pass through the pInitArgs structure. (It contains pointers to multi-threading functions.)
- C_Initialize passes a single BYTE parameter that encodes the version of this encoding. The version SHALL be set to 1 for implementations conformant to this profile.
- C_Finalize does not pass through its parameter.
- C_GetFunctionList is not used. A PKCS#11 stub is expected to provide the function list without reference to the KMIP server.
- C_GetInterface is not used. A PKCS#11 stub is expected to provide the interface without reference to the KMIP server.
- C_WaitForSlotEvent is not used.
- Callbacks are not supported. Accordingly, the Notify parameter on C_OpenSession is not passed to the server.

### 5.18.2 PKCS#11 XML Encoding

The XML encoding from [PKCS11-PROF] MAY be used as an alternative to the KMIP XML encoding of the elements for PKCS#11 in order to improve readability. The PKCS#11 Root Element "PKCS11" SHALL enclose the alternate encoding with the first element holding the correlation value (where a correlation value is present).

### 5.18.2.5.18.3 PKCS#11 Examples

#### 5.18.2.15.18.3.1 PKCS#11 Initialization

```c
CK_RV rv;
CK_FUNCTION_LIST_PTR pFunctionList;
CK_C_Initialize pC_Initialize;

rv = C_GetFunctionList(&pFunctionList); /* C_GetFunctionList in V3.0 */
pC_Initialize = pFunctionList -> C_Initialize;

/* Call the C_Initialize function in the library */
CK_C_INITIALIZATION_ARGS InitArgs;
InitArgs.CreateMutex = &MyCreateMutex;
InitArgs.DestroyMutex = &MyDestroyMutex;
InitArgs.LockMutex = &MyLockMutex;
InitArgs.UnlockMutex = &MyUnlockMutex;
InitArgs.flags = CKF_OS_LOCKING_OK;
InitArgs.pReserved = NULL_PTR;
rv = (*pC_Initialize)((CK_VOID_PTR)&InitArgs);

CK_INFO info;
rv = (*pFunctionList -> C_GetInfo)&info;
if(info.version.major == 2) {...}

CK_SLOT_ID pSlotList[64];
CKULONG ulSlotCount = 64;
```
rv = (*pFunctionList -> C_GetSlotList)(CK_TRUE, pSlotList, ulSlotCount);

C_GetFunctionList

Not passed through

Input C_Initialize

```xml
<PKCS_11Function type="Enumeration" value="C_Initialize"/>
<PKCS_11InputParameters type="Byte String" value="...">
  Version of encoding
</PKCS_11InputParameters>
</PKCS11>
```

Output C_Initialize

```xml
<PKCS_11Function type="Enumeration" value="C_Initialize"/>
<PKCS11>
  <C_Initialize/>
</PKCS11>
```

Input C_GetInfo

```xml
<PKCS_11Function type="Enumeration" value="C_GetInfo"/>
<CorrelationValue type="ByteString" value="ABCD1234"/>
<!-- No Input Parameters -->
</PKCS11>
```

Output C_GetInfo

```xml
<PKCS_11Function type="Enumeration" value="C_GetInfo"/>
<CorrelationValue type="ByteString" value="ABCD1234"/>
<CorrelationValue type="ByteString" value="ABCD1234"/>
<!-- Fields defined by CK_INFO structure. -->
<CorrelationValue type="ByteString" value="ABCD1234"/>
</PKCS11>
```
<PKCS11>
  <CorrelationValue type="ByteString" value="ABCD1234"/>
  <GetInfo rv="OK">
    <Info>
      <CryptokiVersion major="3" minor="1"/>
      <ManufacturerID value="OASIS PKCS#11 TC"/>
      <LibraryDescription value=""/>
      <LibraryVersion major="1" minor="0"/>
    </Info>
  </GetInfo>
</PKCS11>

**Input C_GetSlotList**

<PKCS11Function type="Enumeration" value="C_GetSlotList"/>
<CorrelationValue type="ByteString" value="ABCD1234"/>
<PKCS11InputParameters type="Byte String" value="">
  01  CK_TRUE, tokenPresent indicator
  01  Slot info required
  0000 0040  64 slots in the request array
</PKCS11>

<PKCS11>
  <CorrelationValue type="ByteString" value="ABCD1234"/>
  <GetSlotList>
    <TokenPresent value="true"/>
    <SlotList length=64"/>
  </GetSlotList>
</PKCS11>

**Output C_GetSlotList**

<PKCS11Function type="Enumeration" value="C_GetSlotList"/>
<CorrelationValue type="ByteString" value="ABCD1234"/>
<PKCS11OutputParameters type="Byte String" value="">
  <!-- Fields defined by parameter list. -->
  01  Slot values are present
  0000 0002  ulSlotCount returned
  0000 0000 1234 5678 First CK_SLOT_ID
  0000 0000 ABCD 0987 Second CK_SLOT_ID
  <PKCS11ReturnCode type="Enumeration" value="OK"/>
</PKCS11>

<PKCS11>
  <CorrelationValue type="ByteString" value="ABCD1234"/>
</PKCS11>
5.18.2.25.18.3.2 PKCS#11 C_Encrypt

```c
#define PLAINTEXT_BUF_SZ 195
#define CIPHERTEXT_BUF_SZ 256

CK_ULONG firstPartLen, secondPartLen;
CK_SESSION_HANDLE hSession = 0x12345678; /* For example only */
CK_OBJECT_HANDLE hKey = 0x87654321;
CK_BYTE iv[8] = {1, 2, 3, 4, 5, 6, 7, 8};
CK_MECHANISM mechanism = {
    CKM_DES_CBC_PAD, iv, sizeof(iv)};
CK_BYTE data[PLAINTEXT_BUF_SZ] = {01, 02, 03, ...};
CK_BYTE encryptedData[CIPHERTEXT_BUF_SZ];
CK_ULONG ulEncryptedData1Len; /* Output only(!) */
CK_ULONG ulEncryptedData2Len;
CK_ULONG ulEncryptedData3Len;
.
firstPartLen = 90;
secondPartLen = PLAINTEXT_BUF_SZ-firstPartLen;
C_EncryptInit(hSession, &mechanism, hKey);

/* Encrypt first Part */
ulEncryptedData1Len = sizeof(encryptedData);
C_EncryptUpdate(
    hSession,
    &data[0], firstPartLen,
    &encryptedData[0], &ulEncryptedData1Len);

/* Encrypt second Part */
ulEncryptedData2Len = sizeof(encryptedData)-ulEncryptedData1Len;
C_EncryptUpdate(
    hSession,
    &data[firstPartLen], secondPartLen,
    &encryptedData[ulEncryptedData1Len], &ulEncryptedData2Len);

/* Get last little encrypted bit */
ulEncryptedData3Len =
    sizeof(encryptedData)-ulEncryptedData1Len-ulEncryptedData2Len;
C_EncryptFinal(
    hSession,
    &encryptedData[ulEncryptedData1Len+ulEncryptedData2Len],
    &ulEncryptedData3Len);
```