FIPS Indicators in PKCS #11 Proposal: Draft

FIPS-140-3 allows this indicator in multiple ways: 1) a call with all the relevant data the application uses to determine if an operation will be FIPS compliant before making an call, 2) a special call or indicator the the current operation is FIPS compliant, or 3) any non-FIPS compliant call will always fail.

To the extent possible, PKCS #11 should allow tokens to provide FIPS indication in the most natural way for the module (either strict failure, or by providing an indication). If modules decide to provide the indication, it should be in a common way so all applications can take advantage independent of the module. In support of this goal, the following changes for 3.2 is proposed:

New types:
```c
typedef CK_ULONG CK_VALIDATION_TYPE;
typedef CK_PTR CK_VALIDATION_TYPE CK_VALIDATION_TYPE_PTR;
```

Identifies the type of validation for CKVALIDATION_OBJECTS: this is valid values are:
```c
{CKV_FIPS140TYPE_UNSPECIFIED, CKV_TYPE_SOFTWARE, CKV_TYPE_HARDWARE, CKV_TYPE_FIRMWARE, CKV_TYPE_HYBRID, {others we want to define initially}}
```

```c
typedef CK_ULONG CK_VALIDATION_AUTHORITY_TYPE;
typedef CK_PTR CK_VALIDATION_AUTHORITY_TYPE CK_VALIDATION_AUTHORITY_TYPE_PTR;
```

Identifies the authority that issues the validation. Valid values are:
```c
{CKV_AUTHORITY_TYPE_UNSPECIFIED, CKV_AUTHORITY_TYPE_NIST_CMVP, CKV_AUTHORITY_TYPE_COMMON_CRITERIA, {others?}}
```

```c
typedef CK_ULONG CK_SESSION_VALIDATION_FLAGS_TYPE;
```

Selects which Validation flags to return from C_GetSessionValidationFlags:
```c
{CKS_VALIDATION_CURRENT, CKS_LAST_VALIDATION_OK, CKS_VALIDATION_CUMULATIVE}
```

See Validation Indicators for meanings of these flags.
4.13 Validation objects

4.13.1 Definitions
This section defines the object class CKO_VALIDATION for type CK_OBJECT_CLASS as used in
the CKA_CLASS attribute of objects.

4.13.2 Overview
Validation objects (object class CKO_VALIDATION) describe which third party validations the
module conforms to. Validation objects are read only, token objects.

Table 27, Validation Object Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_VALIDATION_TYPE</td>
<td>CK_VALIDATION_TYPE</td>
<td>PKCS #11 identifier indication the validation type (BSI, FIPS-140, etc).</td>
</tr>
<tr>
<td>CKA_VALIDATION_VERSION</td>
<td>CK_VERSION</td>
<td>Version of the validation standard or spec.</td>
</tr>
<tr>
<td>CKA_VALIDATION_LEVEL</td>
<td>CK ULONG</td>
<td>Validation level, Meaning is Validation type specific</td>
</tr>
<tr>
<td>CKA_VALIDATION_MODULE_ID</td>
<td>CK_UTF8CHAR</td>
<td>How the module is identified in the validation documentation</td>
</tr>
<tr>
<td>CKA_VALIDATION_FLAG</td>
<td>CK_FLAGS</td>
<td>Flags Identifying this validation is in sessions and objects.</td>
</tr>
<tr>
<td>CKA_VALIDATION_AUTHORITY_TYPE</td>
<td>CK_VALIDATION_AUTHORITY_TYPE</td>
<td></td>
</tr>
<tr>
<td>CKA_VALIDATION_COUNTRY</td>
<td>CK_UTF8CHAR</td>
<td>2 letter ISO country code</td>
</tr>
<tr>
<td>CKA_VALIDATION_CERTIFICATE_IDEN</td>
<td>CK_UTF8CHAR</td>
<td></td>
</tr>
<tr>
<td>CKA_VALIDATION_CERTIFICATE_URI</td>
<td>CK_UTF8CHAR</td>
<td></td>
</tr>
<tr>
<td>CKA_VALIDATION_VENDOR_URI</td>
<td>CK_UTF8CHAR</td>
<td></td>
</tr>
<tr>
<td>CKA_VALIDATION_PROFILE</td>
<td>CK_UTF8CHAR</td>
<td></td>
</tr>
</tbody>
</table>

13 Optional value; May be empty.
1.1 Key objects

1.1.1 Definitions

There is no CKO definition for the base key object class, only for the key types derived from it. This section defines the object class CKO_PUBLIC_KEY, CKO_PRIVATE_KEY and CKO_SECRET_KEY for type CK_OBJECT_CLASS as used in the CKA_CLASS attribute of objects.

1.1.2 Overview

Key objects hold encryption or authentication keys, which can be public keys, private keys, or secret keys. The following common footnotes apply to all the tables describing attributes of keys:

The following table defines the attributes common to public key, private key and secret key classes, in addition to the common attributes defined for this object class:

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_KEY_TYPE(^1,5)</td>
<td>CK_KEY_TYPE</td>
<td>Type of key</td>
</tr>
<tr>
<td>CKA_ID(^8)</td>
<td>Byte array</td>
<td>Key identifier for key (default empty)</td>
</tr>
<tr>
<td>CKA_START_DATE(^8)</td>
<td>CK_DATE</td>
<td>Start date for the key (default empty)</td>
</tr>
<tr>
<td>CKA_END_DATE(^8)</td>
<td>CK_DATE</td>
<td>End date for the key (default empty)</td>
</tr>
<tr>
<td>CKA_DERIVE(^8)</td>
<td>CK_BBOOL</td>
<td>CK_TRUE if key supports key derivation (i.e., if other keys can be derived from this one (default CK_FALSE)</td>
</tr>
<tr>
<td>CKA_LOCAL(^2,4,6)</td>
<td>CK_BBOOL</td>
<td>CK_TRUE only if key was either generated locally (i.e., on the token) with a C_GenerateKey or C_GenerateKeyPair call</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• created with a C_CopyObject call as a copy of a key which had its CKA_LOCAL attribute set to CK_TRUE</td>
</tr>
<tr>
<td>CKA_KEY_GEN_MECHANISM(^2,4,6)</td>
<td>CK_MECHANISM_TYPE</td>
<td>Identifier of the mechanism used to generate the key material.</td>
</tr>
<tr>
<td>CKA_ALLOWED_MECHANISMS</td>
<td>CK_MECHANISM_TYPE_PTR, pointer to a CK_MECHANISM_TYPE array</td>
<td>A list of mechanisms allowed to be used with this key. The number of mechanisms in the array is the ulValueLen component of the attribute divided by the size of CK_MECHANISM_TYPE.</td>
</tr>
<tr>
<td>CKA_VALIDATION_FLAGS(^4,9,12)</td>
<td>CK_FLAGS</td>
<td>Object was created consistent with the validations appearing in flags.</td>
</tr>
</tbody>
</table>

\(^1\) Refer to Error: Reference source not found for footnotes

The CKA_ID field is intended to distinguish among multiple keys. In the case of public and private keys, this field assists in handling multiple keys held by the same subject; the key identifier for a public
key and its corresponding private key should be the same. The key identifier should also be the same as for the corresponding certificate, if one exists. Cryptoki does not enforce these associations, however. (See Section Error: Reference source not found for further commentary.)

In the case of secret keys, the meaning of the **CKA_ID** attribute is up to the application. Note that the **CKA_START_DATE** and **CKA_END_DATE** attributes are for reference only; Cryptoki does not attach any special meaning to them. In particular, it does not restrict usage of a key according to the dates; doing this is up to the application.

The **CKA_DERIVE** attribute has the value CK_TRUE if and only if it is possible to derive other keys from the key.

The **CKA_LOCAL** attribute has the value CK_TRUE if and only if the value of the key was originally generated on the token by a **C_GenerateKey** or **C_GenerateKeyPair** call.

The **CKA_KEY_GEN_MECHANISM** attribute identifies the key generation mechanism used to generate the key material. It contains a valid value only if the **CKA_LOCAL** attribute has the value CK_TRUE. If **CKA_LOCAL** has the value CK_FALSE, the value of the attribute is CK_UNAVAILABLE_INFORMATION.

------------------------------------------------------------------------------------------------------------------

1.1.3 Universal Cryptoki function return values

Any Cryptoki function can return any of the following values:

- **CKR_GENERAL_ERROR**: Some horrible, unrecoverable error has occurred. In the worst case, it is possible that the function only partially succeeded, and that the computer and/or token is in an inconsistent state.
- **CKR_HOST_MEMORY**: The computer that the Cryptoki library is running on has insufficient memory to perform the requested function.
- **CKR_FUNCTION_FAILED**: The requested function could not be performed, but detailed information about why not is not available in this error return. If the failed function uses a session, it is possible that the **CK_SESSION_INFO** structure that can be obtained by calling **C_GetSessionInfo** will hold useful information about what happened in its ulDeviceError field. In any event, although the function call failed, the situation is not necessarily totally hopeless, as it is likely to be when **CKR_GENERAL_ERROR** is returned. Depending on what the root cause of the error actually was, it is possible that an attempt to make the exact same function call again would succeed.
- **CKR_OPERATION_NOT_VALIDATED**. The requested operation violates one or more of the token’s validation policies. Tokens may choose to return a more specific error (like **CKR_ATTRIBUTEVALUE_INVALID** or **CKR_DATA_LEN_RANGE**).
- **CKR_OK**: The function executed successfully. Technically, **CKR_OK** is not quite a “universal” return value; in particular, the legacy functions **C_GetFunctionStatus** and **C_CancelFunction** (see Section Error: Reference source not found) cannot return **CKR_OK**.

The relative priorities of these errors are in the order listed above, e.g., if either of **CKR_GENERAL_ERROR** or **CKR_HOST_MEMORY** would be an appropriate error return, then **CKR_GENERAL_ERROR** should be returned.

------------------------------------------------------------------------------------------------------------------

New section

Validation Indicators:
Validation indicators are runtime indicator if a particular operation meets the appropriate criteria for this module running under the given validation rules. These rules will very by validation type and even by different modules using various validation types.

Session validation flags
Sessions carry three validation flags: Current operation flags, last operation flags, cumulative flags. These can be queried with C_GetSessionValidationFlags.

Current operation flags
The current operation flags applies to the current active operation (between C_XXXInit and C_XXXFinal calls). If more than one operation is active (for instance a Encrypt and Sign operations when doing things like C_EncryptSignUpdate or C_VerifyDecryptUpdate), the returned current operation flags are the bit wise and of the current operation flags for each of the existing operations. The current operation flags are zero if no operation is active.

Last operation flags
The last operation flags freeze the current operation flags for is set if the last operation that completed (the last C_XXXFinal, or the last single short operation (C.WrapKey, C_Derive, etc.) met all the requirements of a validated mechanism. This allows access to the state of operations that don't return a key object.

Cumulative flags
Cumulative flags are the running bit wise or of all the validation flags for all the operation on the session.

Key object state
CKA_VALIDATION_FLAGS can only be set in ways conforming to the module’s validation. Key objects typically take on the flags of the operation that created them, but are subject to the modules requirements under the module’s validation.

Application notes:
Applications should be prepared for changes in semantics as various validations change their guidance.

Notes to application. Many of these operations chain, so in SSL, if the final key objects have the appropriate flags set in CKA_VALIDATION_FLAGS, then it generally means that all the operations (unwrap, key_derive, etc.) occurred in a manner that matches the modules’s validation, so the application would generally only need to query the validation flags of the final keys.

C_GetSessionValidationFlags(CK_HANDLE session, CK_SESSION_VALIDATION_FLAGS_TYPE type, CK_FLAGS_PTR pflags)

Fetch the requested flags from the session. See Validation indicators for meaning and semantics for these flags. Applications are responsible for the appropriate locking to protect session to get a meaningful result from this call.