FIPS Indicators in PKCS #11 Proposal:

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 calls, 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 3.1 is proposed:

- **CK_SESSION_INFO; CK_SESSION_INFO_PTR**

CK_SESSION_INFO provides information about a session. It is defined as follows:

```c
typedef struct CK_SESSION_INFO {
  CK_SLOT_ID slotID;
  CK_STATE state;
  CK_FLAGS flags;
  CK_ULONG ulDeviceError;
} CK_SESSION_INFO;
```

The fields of the structure have the following meanings:

- **slotID** the ID of the slot that interfaces with the token
- **state** the state of the session
- **flags** bit flags that define the type of session; the flags are defined below
- **ulDeviceError** an error code defined by the cryptographic device. Used for errors not covered by Cryptoki.

The following table defines the **flags** field:

<table>
<thead>
<tr>
<th>Bit Flag</th>
<th>Mask</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKF_RW_SESSION</td>
<td>0x00000002</td>
<td>True if the session is read/write; false if the session is read-only</td>
</tr>
<tr>
<td>CKF_SERIAL_SESSION</td>
<td>0x00000004</td>
<td>This flag is provided for backward compatibility, and should always be set to true</td>
</tr>
<tr>
<td>CKF_FIPS_OK</td>
<td>TBD</td>
<td>This flag is true if the current operation is currently conforming to the module’s NIST approved security policy.</td>
</tr>
<tr>
<td>CKF_FIPS_LAST_OK</td>
<td>TBD</td>
<td>This flag is true if the last operation conformed the the module’s NIST approved security policy.</td>
</tr>
</tbody>
</table>

CK_SESSION_INFO_PTR is a pointer to a CK_SESSION_INFO.
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 2, Common Key Attributes

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Data Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CKA_KEY_TYPE</td>
<td>CK_KEY_TYPE</td>
<td>Type of key</td>
</tr>
<tr>
<td>CKA_ID</td>
<td>Byte array</td>
<td>Key identifier for key (default empty)</td>
</tr>
<tr>
<td>CKA_START_DATE</td>
<td>CK_DATE</td>
<td>Start date for the key (default empty)</td>
</tr>
<tr>
<td>CKA_END_DATE</td>
<td>CK_DATE</td>
<td>End date for the key (default empty)</td>
</tr>
<tr>
<td>CKA_DERIVE</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</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 • 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</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_FIPS_OK</td>
<td>CK_BBOOL</td>
<td>CK_TRUE if the key was created by a FIPS approved method under the token’s security policy</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_SESION_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_FIPS_INVALID**: The requested operation violates the token’s NIST approved security policy. Tokens may choose to return a more specific error (like `CKR_ATTRIBUTE_VALUE_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

**FIPS Indicators:**

FIPS-140-3 requires a runtime indicator if a particular operation is FIPS compliant or not. A number of factors can play into whether or not a particular operation is FIPS compliant, including whether or not the algorithm and key sizes are FIPS approved, whether or not they are used in a FIPS approved way.
whether or not if they have been properly validated, and whether or not they are running in the constraints indicated in the security policy.

The PKCS #11 FIPS Indicators provide application a runtime view of what is allowed under the token’s security policy. Tokens must not present FIPS indicators if the token does a FIPS validation, and the operation isn’t compliant with the token’s security policy.

Session state
CKF_FIPS_OK should be off for any session the does not have a valid cryptographic operation pending. Once an operation completes, (a C_XXXFinal operation, completes) CKF_FIPS_OK gets copied to CKF_FIPS_LAST_OK. Single shot operations also affect the state of CKF_FIPS_LAST_OK. If more than one operation is active at the same time, both operations must be compliant with the token’s security policy for CKF_FIPS_OK to be set. If any non-approved action happens on the operation (like digesting a non-FIPS key), the entire operation is not approved until that operation is closed with C_XXXXFinal.

Key object state
CKA_FIPS_OK can only be set in ways approved by the token’s security policy. Some tokens may accept CKA_FIPS_OK on unwrap, others may only implicitly set that value if a FIPS approved algorithm is used in the unwrap operation.

Application notes:
Applications should be prepared for changes in semantics as NIST guidance changes.

Notes to application. Many of these operations chain, so in SSL, if the final key objects have CKA_FIPS_OK set to true, it generally means that all the operations (unwrap, key_derive, etc.) occurred in a manner that matches the token’s security policy, so the application would generally only need to query the FIPS_OK state of the final keys.