C\_EncryptCancel proposal draft 3, March 29, 2017

Author: Darren Johnson

Summary:

Draft 1 of this proposal introduced multiple functions that could be used to terminate indiviual session based operations. That idea was rejected in favour of a single API that can be used to cancel any/all session based operations.

Draft 2 of proposal introduced a new function **C\_SessionCancel** that can be used to cancel any session based operation as well as all session based operations that are currently active in a session.

Draft 3 of this proposal introduces some additional text to existing APIs (C\_EncryptInit, C\_DecryptInit, C\_DigestInit, C\_SignInit, C\_VerifyInit, C\_SignRecoverInit, C\_VerifyRecoverInit, C\_MessageEncryptInit, C\_MessageDecryptInit, C\_MessageSignInit, C\_MessageVerifyInit) to allow them to be used to cancel their corresponding session based operation.

Note:

This proposal depends on the “New function table proposal” as it introduces a new function.

A new error code will be added to “Section 5.1.6 All other Cryptoki function return values“

* CKR\_OPERATION\_CANCEL\_FAILED: This value can only be returned by **C\_SessionCancel**. It means that one or more of the requested operations could not be cancelled for implementation or vendor-specific reasons.

A new mechanism information flag will be added to Table 8, Mechanism Information Flags.

| **Bit Flag** | **Mask** | **Meaning** |
| --- | --- | --- |
| CKF\_FIND\_OBJECTS | 0x00000040 | This flag can be passed in as a parameter to **C\_CancelSession** to cancel an active object search operation. Any other use of this flag is outside the scope of this standard. |

1. C\_SessionCancel

CK\_DECLARE\_FUNCTION(CK\_RV, C\_SessionCancel)(  
 CK\_SESSION\_HANDLE hSession   
 CK\_FLAGS flags   
);

**C\_SessionCancel** terminates active session based operations. *hSession* is the session’s handle; *flags* indicates the operations to cancel.

To identify which operation(s) should be terminated, the *flags* parameter should be assigned the logical bitwise OR of one or more of the bit flags defined in the **CK\_MECHANISM\_INFO** structure.

If no flags are set, the session state will not be modified and CKR\_OK will be returned.

If a flag is set for an operation that has not been initialized in the session, the operation flag will be ignored and **C\_SessionCancel** will behave as if the operation flag was not set.

If any of the operations indicated by the *flags* parameter cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned. If multiple operation flags were set and CKR\_OPERATION\_CANCEL\_FAILED is returned, this function does not provide any information about which operation(s) could not be cancelled. If an application desires to know if any single operation could not be cancelled, the application should not call **C\_SessionCancel** with multiple flags set.

If **C\_SessionCancel** is called from an application callback (see Section 5.16), no action will be taken by the library and CKR\_FUNCTION\_FAILED must be returned.

If **C\_SessionCancel** is used to cancel one half of a dual-function operation, the remaining operation should still be left in an active state. However, it is expected that some Cryptoki implementations may not support this and return CKR\_OPERATION\_CANCEL\_FAILED unless flags for both operations are provided.

Example:

CK\_SESSION\_HANDLE hSession;

CK\_RV rv;

rv = C\_EncryptInit(hSession, &mechanism, hKey);

if (rv != CKR\_OK)

{

.

.

}

rv = C\_SessionCancel (hSession, CKF\_ENCRYPT);

if (rv != CKR\_OK)

{

.

.

}

rv = C\_EncryptInit(hSession, &mechanism, hKey);

if (rv != CKR\_OK)

{

.

.

}

Below are modifications to existing API descriptions to allow an alternate method of cancelling individual operations. The additional text is highlighted.

1. C\_EncryptInit

CK\_DECLARE\_FUNCTION(CK\_RV, C\_EncryptInit)(  
 CK\_SESSION\_HANDLE hSession,  
 CK\_MECHANISM\_PTR pMechanism,  
 CK\_OBJECT\_HANDLE hKey  
);

**C\_EncryptInit** initializes an encryption operation. *hSession* is the session’s handle; *pMechanism* points to the encryption mechanism; *hKey* is the handle of the encryption key.

The **CKA\_ENCRYPT** attribute of the encryption key, which indicates whether the key supports encryption, MUST be CK\_TRUE.

After calling **C\_EncryptInit**, the application can either call **C\_Encrypt** to encrypt data in a single part; or call **C\_EncryptUpdate** zero or more times, followed by **C\_EncryptFinal**, to encrypt data in multiple parts. The encryption operation is active until the application uses a call to **C\_Encrypt** or **C\_EncryptFinal** *to actually obtain* the final piece of ciphertext. To process additional data (in single or multiple parts), the application MUST call **C\_EncryptInit** again.

**C\_EncryptInit** can be called with *pMechanism* set to NULL\_PTR to terminate an active encryption operation. If an active operation operations cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

Example: see **C\_EncryptFinal**.

1. C\_DecryptInit

CK\_DECLARE\_FUNCTION(CK\_RV, C\_DecryptInit)(  
 CK\_SESSION\_HANDLE hSession,  
 CK\_MECHANISM\_PTR pMechanism,  
 CK\_OBJECT\_HANDLE hKey  
);

**C\_DecryptInit** initializes a decryption operation. *hSession* is the session’s handle; *pMechanism* points to the decryption mechanism; *hKey* is the handle of the decryption key.

The **CKA\_DECRYPT** attribute of the decryption key, which indicates whether the key supports decryption, MUST be CK\_TRUE.

After calling **C\_DecryptInit**, the application can either call **C\_Decrypt** to decrypt data in a single part; or call **C\_DecryptUpdate** zero or more times, followed by **C\_DecryptFinal**, to decrypt data in multiple parts. The decryption operation is active until the application uses a call to **C\_Decrypt** or **C\_DecryptFinal** *to actually obtain* the final piece of plaintext. To process additional data (in single or multiple parts), the application MUST call **C\_DecryptInit** again.

**C\_DecryptInit** can be called with *pMechanism* set to NULL\_PTR to terminate an active decryption operation. If an active operation cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

Example: see **C\_DecryptFinal**.

1. C\_DigestInit

CK\_DECLARE\_FUNCTION(CK\_RV, C\_DigestInit)(  
 CK\_SESSION\_HANDLE hSession,  
 CK\_MECHANISM\_PTR pMechanism  
);

**C\_DigestInit** initializes a message-digesting operation. *hSession* is the session’s handle; *pMechanism* points to the digesting mechanism.

After calling **C\_DigestInit**, the application can either call **C\_Digest** to digest data in a single part; or call **C\_DigestUpdate** zero or more times, followed by **C\_DigestFinal**, to digest data in multiple parts. The message-digesting operation is active until the application uses a call to **C\_Digest** or **C\_DigestFinal** *to actually obtain* the message digest. To process additional data (in single or multiple parts), the application MUST call **C\_DigestInit** again.

**C\_DigestInit** can be called with *pMechanism* set to NULL\_PTR to terminate an active message-digesting operation. If an operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

Example: see **C\_DigestFinal**.

1. C\_SignInit

CK\_DECLARE\_FUNCTION(CK\_RV, C\_SignInit)(  
 CK\_SESSION\_HANDLE hSession,  
 CK\_MECHANISM\_PTR pMechanism,  
 CK\_OBJECT\_HANDLE hKey  
);

**C\_SignInit** initializes a signature operation, where the signature is an appendix to the data. *hSession* is the session’s handle; *pMechanism* points to the signature mechanism; *hKey* is the handle of the signature key.

The **CKA\_SIGN** attribute of the signature key, which indicates whether the key supports signatures with appendix, MUST be CK\_TRUE.

After calling **C\_SignInit**, the application can either call **C\_Sign** to sign in a single part; or call **C\_SignUpdate** one or more times, followed by **C\_SignFinal,** to sign data in multiple parts. The signature operation is active until the application uses a call to **C\_Sign** or **C\_SignFinal** *to actually obtain* the signature. To process additional data (in single or multiple parts), the application MUST call **C\_SignInit** again.

**C\_SignInit** can be called with *pMechanism* set to NULL\_PTR to terminate an active signature operation. If an operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED,CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

Example: see **C\_SignFinal**.

1. C\_SignRecoverInit

CK\_DECLARE\_FUNCTION(CK\_RV, C\_SignRecoverInit)(  
 CK\_SESSION\_HANDLE hSession,  
 CK\_MECHANISM\_PTR pMechanism,  
 CK\_OBJECT\_HANDLE hKey  
);

**C\_SignRecoverInit** initializes a signature operation, where the data can be recovered from the signature. *hSession* is the session’s handle; *pMechanism* points to the structure that specifies the signature mechanism; *hKey* is the handle of the signature key.

The **CKA\_SIGN\_RECOVER** attribute of the signature key, which indicates whether the key supports signatures where the data can be recovered from the signature, MUST be CK\_TRUE.

After calling **C\_SignRecoverInit**, the application may call **C\_SignRecover** to sign in a single part. The signature operation is active until the application uses a call to **C\_SignRecover** *to actually obtain* the signature. To process additional data in a single part, the application MUST call **C\_SignRecoverInit** again.

**C\_SignRecoverInit** can be called with *pMechanism* set to NULL\_PTR to terminate an active signature with data recovery operation. If an active operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

Example: see **C\_SignRecover**.

1. C\_VerifyInit

CK\_DECLARE\_FUNCTION(CK\_RV, C\_VerifyInit)(  
 CK\_SESSION\_HANDLE hSession,  
 CK\_MECHANISM\_PTR pMechanism,  
 CK\_OBJECT\_HANDLE hKey  
);

**C\_VerifyInit** initializes a verification operation, where the signature is an appendix to the data. *hSession* is the session’s handle; *pMechanism* points to the structure that specifies the verification mechanism; *hKey* is the handle of the verification key.

The **CKA\_VERIFY** attribute of the verification key, which indicates whether the key supports verification where the signature is an appendix to the data, MUST be CK\_TRUE.

After calling **C\_VerifyInit**, the application can either call **C\_Verify** to verify a signature on data in a single part; or call **C\_VerifyUpdate** one or more times, followed by **C\_VerifyFinal,** to verify a signature on data in multiple parts. The verification operation is active until the application calls **C\_Verify** or **C\_VerifyFinal**. To process additional data (in single or multiple parts), the application MUST call **C\_VerifyInit** again.

**C\_VerifyInit** can be called with *pMechanism* set to NULL\_PTR to terminate an active verification operation. If an active operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

Example: see **C\_VerifyFinal**.

1. C\_VerifyRecoverInit

CK\_DECLARE\_FUNCTION(CK\_RV, C\_VerifyRecoverInit)(  
 CK\_SESSION\_HANDLE hSession,  
 CK\_MECHANISM\_PTR pMechanism,  
 CK\_OBJECT\_HANDLE hKey  
);

**C\_VerifyRecoverInit** initializes a signature verification operation, where the data is recovered from the signature. *hSession* is the session’s handle; *pMechanism* points to the structure that specifies the verification mechanism; *hKey* is the handle of the verification key.

The **CKA\_VERIFY\_RECOVER** attribute of the verification key, which indicates whether the key supports verification where the data is recovered from the signature, MUST be CK\_TRUE.

After calling **C\_VerifyRecoverInit**, the application may call **C\_VerifyRecover** to verify a signature on data in a single part. The verification operation is active until the application uses a call to **C\_VerifyRecover** *to actually obtain* the recovered message.

**C\_VerifyRecoverInit** can be called with *pMechanism* set to NULL\_PTR to terminate an active verification with data recovery operation. If an active operations has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

Example: see **C\_VerifyRecover**.

**C\_MessageEncryptInit**

CK\_DEFINE\_FUNCTION(CK\_RV,C\_MessageEncryptInit)(

CK\_SESSION\_HANDLE hSession,

CK\_MECHANISM\_PTR pMechanism,

CK\_OBJECT\_HANDLE hKey

);

C\_MessageEncryptInit prepares a session for one or more encryption operations that use the same encryption mechanism and encryption key. hSession is the session’s handle; pMechanism points to the encryption mechanism; hKey is the handle of the encryption key.

The CKA\_ENCRYPT attribute of the encryption key, which indicates whether the key supports encryption, MUST be CK\_TRUE.

After calling C\_MessageEncryptInit, the application can either call C\_EncryptMessage to encrypt a message in a single part, or call C\_EncryptMessageBegin, followed by C\_EncryptMessageNext one or more times, to encrypt a message in multiple parts. This may be repeated several times. The message-based encryption process is active until the application calls C\_MessageEncryptFinal to finish the message-based encryption process.

**C\_MessageEncryptInit** can be called with *pMechanism* set to NULL\_PTR to terminate a message-based encryption process. If a multi-part message encryption operation is active, it will also be terminated. If an active operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

**C\_MessageDecryptInit**

CK\_DEFINE\_FUNCTION(CK\_RV,C\_MessageDecryptInit)(

CK\_SESSION\_HANDLE hSession,

CK\_MECHANISM\_PTR pMechanism,

CK\_OBJECT\_HANDLE hKey

);

C\_MessageDecryptInit initializes a message-based decryption process, preparing a session for one or more decryption operations that use the same decryption mechanism and decryption key. ***hSession*** is the session’s handle; ***pMechanism*** points to the decryption mechanism; ***hKey*** is the handle of the decryption key.

The CKA\_DECRYPT attribute of the decryption key, which indicates whether the key supports decryption, MUST be CK\_TRUE.

After calling C\_MessageDecryptInit, the application can either call C\_DecryptMessage to decrypt an encrypted message in a single part; or call C\_DecryptMessageBegin, followed by C\_DecryptMessageNext one or more times, to decrypt an encrypted message in multiple parts. This may be repeated several times. The message-based decryption process is active until the application uses a call to C\_MessageDecryptFinal to finish the message-based

decryption process.

**C\_MessageDecryptInit** can be called with *pMechanism* set to NULL\_PTR to terminate a message-based decryption process. If a multi-part message decryption operation is active, it will also be terminated. If an active operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

**C\_MessageSignInit**

CK\_DEFINE\_FUNCTION(CK\_RV,C\_MessageSignInit)(

CK\_SESSION\_HANDLE hSession,

CK\_MECHANISM\_PTRpMechanism,

CK\_OBJECT\_HANDLEhKey

);

C\_MessageSignInit initializes a message-based signature process, preparing a session for one or more signature operations (where the signature is an appendix to the data) that use the same signature mechanism and signature key. hSession is the session’s handle; pMechanism points to the signature mechanism; hKey is the handle of the signature key.

The CKA\_SIGN attribute of the signature key, which indicates whether the key supports signatures with appendix, MUST be CK\_TRUE.

After calling C\_MessageSignInit, the application can either call C\_SignMessage to sign a message in a single part; or call C\_SignMessageBegin, followed by C\_SignMessageNext one or more times, to sign a message in multiple parts. This may be repeated several times. The message-based signature process is active until the application calls C\_MessageSignFinal to finish the message-based signature process.

**C\_MessageSignInit** can be called with *pMechanism* set to NULL\_PTR to terminate a message-based signature process. If a multi-part message signature operation is active, it will also be terminated. If an active operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED,CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.

**C\_MessageVerifyInit**

CK\_DEFINE\_FUNCTION(CK\_RV,C\_MessageVerifyInit)(

CK\_SESSION\_HANDLE hSession,

CK\_MECHANISM\_PTR pMechanism,

CK\_OBJECT\_HANDLE hKey

);

**C\_MessageVerifyInit i**nitializes a message-based verification process, preparing a session for one or more verification operations (where the signature is an appendix to the data) that use the same verification mechanism and verification key. hSession is the session’s handle; pMechanism points to the structure that specifies the verification mechanism; hKey is the handle of the verification key.

The CKA\_VERIFY attribute of the verification key, which indicates whether the key supports verification where the signature is an appendix to the data, MUST be CK\_TRUE.

After calling **C\_MessageVerifyInit**, the application can either call **C\_VerifyMessage** to verify a signature on a message in a single part; or call **C\_VerifyMessageBegin**, followed by **C\_VerifyMessageNext** one or more times, to verify a signature on a message in multiple parts. This may be repeated several times. The message-based verification process is active until the application calls **C\_MessageVerifyFinal** to finish the message-based verification process.

**C\_MessageVerifyInit** can be called with *pMechanism* set to NULL\_PTR to terminate a message-based verification process. If a multi-part message verification operation is active, it will also be terminated. If an active operation has been initialized and it cannot be cancelled, CKR\_OPERATION\_CANCEL\_FAILED must be returned.

Return values: CKR\_ARGUMENTS\_BAD, CKR\_CRYPTOKI\_NOT\_INITIALIZED, CKR\_DEVICE\_ERROR, CKR\_DEVICE\_MEMORY, CKR\_DEVICE\_REMOVED, CKR\_FUNCTION\_CANCELED, CKR\_FUNCTION\_FAILED, CKR\_GENERAL\_ERROR, CKR\_HOST\_MEMORY, CKR\_KEY\_FUNCTION\_NOT\_PERMITTED, CKR\_KEY\_HANDLE\_INVALID, CKR\_KEY\_SIZE\_RANGE, CKR\_KEY\_TYPE\_INCONSISTENT, CKR\_MECHANISM\_INVALID, CKR\_MECHANISM\_PARAM\_INVALID, CKR\_OK, CKR\_OPERATION\_ACTIVE, CKR\_PIN\_EXPIRED, CKR\_SESSION\_CLOSED, CKR\_SESSION\_HANDLE\_INVALID, CKR\_USER\_NOT\_LOGGED\_IN, CKR\_OPERATION\_CANCEL\_FAILED.