1.1 FUNCTIONS

Key management functions	C_GenerateKey	generates a secret key
	C_GenerateKeyPair	generates a public-key/private-key pair
	C_WrapKey	wraps (encrypts) a key
	C_UnwrapKey	unwraps (decrypts) a key
	C_WrapKeyAuthenticated	Authenticated key Wrapping (encrypt) a key
	C_UnWrapKeyAuthenticated	Authenticated key unwrapping (decrypt) a key
	C_DeriveKey	derives a key from a base key

1.2 (5.18) Key management functions

1.2.1 C_WrapKeyAuthenticated

```
CK_DECLARE_FUNCTION(CK_RV, C_WrapKeyAuthenticated)(
  CK_SESSION_HANDLE hSession,
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hWrappingKey,
  CK_OBJECT_HANDLE hKey,
  CK_VOID_PTR pParameter,
  CK_ULONG ulParameterLen,
  CK_BYTE_PTR pAssociatedData,
  CK_ULONG ulAssociatedDataLen,
  CK_BYTE_PTR pWrappedKey,
  CK_ULONG_PTR pulWrappedKeyLen
);
```

C_WrapMessageKey wraps (*i.e.*, encrypts) a private or secret key. *hSession* is the session's handle; *pMechanism* points to the wrapping mechanism; *hWrappingKey* is the handle of the wrapping key; *hKey* is the handle of the key to be wrapped; *pParameter* and *ulParameterLen* specify any mechanism-specific parameters for the message wrap operation; *pAssociatedData* and *ulAssociatedDataLen* specify the associated data for an AEAD mechanism; *pWrappedKey* points to the location that receives the wrapped key; and *pulWrappedKeyLen* points to the location that receives the length of the wrapped key.

C_WrapKeyAuthenticated uses the convention described in Section on producing output.

The **CKA_WRAP** attribute of the wrapping key, which indicates whether the key supports wrapping, MUST be CK_TRUE. The **CKA_EXTRACTABLE** attribute of the key to be wrapped MUST also be CK_TRUE.

If the key to be wrapped cannot be wrapped for some token-specific reason, despite its having its **CKA_EXTRACTABLE** attribute set to CK_TRUE, then **C_WrapKeAuthenticated** fails with error code CKR_KEY_NOT_WRAPPABLE. If it cannot be wrapped with the specified wrapping key and mechanism solely because of its length, then **C_WrapKeyAuthenticated** fails with error code CKR_KEY_SIZE_RANGE.

C_WrapKeyAuthenticated can be used in the following situations:

- To wrap any secret key with a public key that supports encryption and decryption.
- To wrap any secret key with any other secret key. Consideration MUST be given to key size and mechanism strength or the token may not allow the operation.
- To wrap a private key with any secret key.

Of course, tokens vary in which types of keys can actually be wrapped with which mechanisms.

To partition the wrapping keys so they can only wrap a subset of extractable keys the attribute CKA_WRAP_TEMPLATE can be used on the wrapping key to specify an attribute set that will be compared against the attributes of the key to be wrapped. If all attributes match according to the C_FindObject rules of attribute matching then the wrap will proceed. The value of this attribute is an attribute template, and the size is the number of items in the template times the size of CK_ATTRIBUTE. If this attribute is not supplied, then any template is acceptable. If an attribute is not present, it will not be checked. If any attribute mismatch occurs on an attempt to wrap a keykey, then the function SHALL return CKR_KEY_HANDLE_INVALID.

Return Values: CKR_ARGUMENTS_BAD, CKR_BUFFER_TOO_SMALL, 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_HANDLE_INVALID, CKR_KEY_NOT_WRAPPABLE, CKR_KEY_SIZE_RANGE, CKR_KEY_UNEXTRACTABLE, 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_WRAPPING_KEY_HANDLE_INVALID, CKR_WRAPPING_KEY_SIZE_RANGE, CKR_WRAPPING_KEY_TYPE_INCONSISTENT.

Example:

```
#define AUTH_BUF_SZ 100
CK_BYTE auth[2][AUTH_BUF_SZ];
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hWrappingKey, hKey;
CK_BYTE iv[12];
CK_BYTE tag[16];
CK_GCM_MESSAGE_PARAMS gcmParams = {
  iv,
  sizeof(iv) * 8,
  96,
  CKG_GENERATE,
  tag,
  sizeof(tag) * 8
};

CK_MECHANISM mechanism = {
  CKM_AES_GCM, &gcmParams, sizeof(gcmParams)
};
CK_BYTE wrappedKey[32]; /* only the wrapped key returned*/
CK_ULONG ulWrappedKeyLen;
CK_RV rv;
.
.
.
ulWrappedKeyLen = sizeof(wrappedKey);
rv = C_WrapMessageKey(
  hSession, &mechanism,

  hWrappingKey, hKey,
  gcmParams, sizeof(gcmParams),
  &auth[0][0], sizeof(auth[0]),
  wrappedKey, &ulWrappedKeyLen);
if (rv == CKR_OK) {
  .
  .
}
```

1.2.2 C_UnwrapKeyAuthenticated

```
CK_DECLARE_FUNCTION(CK_RV, C_UnwrapMessageKey)(
  CK_SESSION_HANDLE hSession,
```

```
  CK_MECHANISM_PTR pMechanism,
  CK_OBJECT_HANDLE hUnwrappingKey,
  CK_BYTE_PTR pWrappedKey,
  CK_ULONG ulWrappedKeyLen,
  CK_ATTRIBUTE_PTR pTemplate,
  CK_ULONG ulAttributeCount,
  CK_VOID_PTR pParameter,
  CK_ULONG ulParameterLen,
  CK_BYTE_PTR pAssociatedData,
  CK_ULONG ulAssociatedDataLen
  CK_OBJECT_HANDLE_PTR phKey
);
```

C_UnwrapKeyAuthenticated unwraps (*i.e.* decrypts) a wrapped key, creating a new private key or secret key object. *hSession* is the session's handle; *pMechanism* points to the unwrapping mechanism; *hUnwrappingKey* is the handle of the unwrapping key; *pWrappedKey* points to the wrapped key; *ulWrappedKeyLen* is the length of the wrapped key; *pTemplate* points to the template for the new key; *ulAttributeCount* is the number of attributes in the template; *pParameter* and *ulParameterLen* specify any mechanism-specific parameters for the ~~message~~ unwrap
operation; *pAssociatedData* and *ulAssociatedDataLen* specify the associated data for an AEAD mechanism; *phKey* points to the location that receives the handle of the ~~recovered~~ key.

The **CKA_UNWRAP** attribute of the unwrapping key, which indicates whether the key supports unwrapping, MUST be CK_TRUE.

The new key will have the **CKA_ALWAYS_SENSITIVE** attribute set to CK_FALSE, and the **CKA_NEVER_EXTRACTABLE** attribute set to CK_FALSE. The **CKA_EXTRACTABLE** attribute is by default set to CK_TRUE.

Some mechanisms may modify, or attempt to modify. the contents of the pMechanism structure at the same time that the key is unwrapped.

If a call to **C_UnwrapKeyAuthenticated** cannot support the precise template supplied to it, it will fail and return without creating any key object.

The key object created by a successful call to ~~**C_UnwrapKeyAuthenticted**~~ will have its **CKA_LOCAL** attribute set to CK_FALSE. In addition, the object created will have a value for CKA_UNIQUE_ID generated and assigned (See Section **Error! Reference source not found.**).

To partition the unwrapping keys so they can only unwrap a subset of keys the attribute CKA_UNWRAP_TEMPLATE can be used on the unwrapping key to specify an attribute set that will be added to attributes of the key to be unwrapped. If the attributes do not conflict with the user supplied attribute template, in 'pTemplate', then the unwrap will proceed. The value of this attribute is an attribute template and the size is the number of items in the template times the size of CK_ATTRIBUTE. If this attribute is not present on the unwrapping key then no additional attributes will be added. If any attribute conflict occurs on an attempt to unwrap a key then the function SHALL return CKR_TEMPLATE_INCONSISTENT.

Return values: CKR_ARGUMENTS_BAD, CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, CKR_BUFFER_TOO_SMALL, CKR_CRYPTOKI_NOT_INITIALIZED, CKR_CURVE_NOT_SUPPORTED, CKR_DEVICE_ERROR, CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_DOMAIN_PARAMS_INVALID, 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_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_UNWRAPPING_KEY_HANDLE_INVALID, CKR_UNWRAPPING_KEY_SIZE_RANGE, CKR_UNWRAPPING_KEY_TYPE_INCONSISTENT, CKR_USER_NOT_LOGGED_IN, CKR_WRAPPED_KEY_INVALID, CKR_WRAPPED_KEY_LEN_RANGE.

Example:

```
#define AUTH_BUF_SZ 100

CK_BYTE auth[2][AUTH_BUF_SZ];
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hUnwrappingKey, hKey;
CK_MECHANISM mechanism = {
  CKM_AES_GCM, NULL_PTR, 0
};
CK_BYTE wrappedKey[32] = {...};
CK_OBJECT_CLASS keyClass = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_AES;
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &keyClass, sizeof1(keyClass)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_DECRYPT, &true, sizeof(true)}
};
CK_RV rv;
CK_BYTE iv[] = {1 , 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 }; /*value from wrap
CKG_GENERATE */
CK_BYTE tag[16];
CK_GCM_MESSAGE_PARAMS gcmParams = {
  iv,
  sizeof(iv) * 8,
  0, /* ignored */
  CKG_NO_GENERATE, /* ignored */
  tag, /* Tag returned from Wrap */
  sizeof(tag) * 8
};
.
.
rv = C_UnwrapKeyAuthenticated(
  hSession, &mechanism, hUnwrappingKey,
  gcmParams, sizeof(gcmParams),
  &auth[0][0], sizeof(auth[0]),
  wrappedKey, sizeof(wrappedKey),
  template, 4, &hKey);
if (rv == CKR_OK) {
  .
  .
}
```

1.3 (6.13) Additional AES Mechanisms

Table 1, Additional AES Mechanisms vs. Functions

Mechanism	Functions						
	Encrypt & Decrypt	**Sign & Verify**	**SR & VR**[1]	**Digest**	**Gen. Key/ Key Pair**	**Wrap & Unwrap**	**Derive**
CKM_AES_GCM	✓					✓	
CKM_AES_CCM	✓					✓	
CKM_AES_GMAC		✓					

1.3.1 Definitions

Mechanisms:

CKM_AES_GCM

CKM_AES_CCM

CKM_AES_GMAC

Generator Functions:

CKG_NO_GENERATE

CKG_GENERATE

CKG_GENERATE_COUNTER

CKG_GENERATE_RANDOM

CKG_GENERATE_COUNTER_XOR

1.3.2 AES-GCM Authenticated Encryption / Decryption

Generic GCM mode is described in [GCM]. To set up for AES-GCM use the following process, where *K* (key) and *AAD* (additional authenticated data) are as described in [GCM]. AES-GCM uses CK_GCM_PARAMS for Encrypt, Decrypt and CK_GCM_MESSAGE_PARAMS for MessageEncrypt and MessageDecrypt.

Encrypt:

- Set the IV length *ulIvLen* in the parameter block.
- Set the IV data *pIv* in the parameter block.
- Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD m*ay be NULL if *ulAADLen* is 0.
- Set the tag length *ulTagBits* in the parameter block.
- Call C_EncryptInit() for **CKM_AES_GCM** mechanism with parameters and key *K*.
- Call C_Encrypt(), or C_EncryptUpdate()*[1] C_EncryptFinal(), for the plaintext obtaining ciphertext and authentication tag output.

Decrypt:

1 “*” indicates 0 or more calls may be made as required

- Set the IV length *ulIvLen* in the parameter block.
- Set the IV data *pIv* in the parameter block.
- Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD m*ay be NULL if ulAADLen is 0.
- Set the tag length *ulTagBits* in the parameter block.
- Call C_DecryptInit() for **CKM_AES_GCM** mechanism with parameters and key *K*.
- Call C_Decrypt(), or C_DecryptUpdate()*[1] C_DecryptFinal(), for the ciphertext, including the appended tag, obtaining plaintext output. Note: since **CKM_AES_GCM** is an AEAD cipher, no data should be returned until C_Decrypt() or C_DecryptFinal().

MessageEncrypt:

- Set the IV length *ulIvLen* in the parameter block.
- Set *pIv* to hold the IV data returned from C_EncryptMessage() and C_EncryptMessageBegin(). If *ulIvFixedBits* is not zero, then the most significant bits of *pIV* contain the fixed IV. If *ivGenerator* is set to CKG_NO_GENERATE, *pIv* is an input parameter with the full IV.
- Set the *ulIvFixedBits* and *ivGenerator* fields in the parameter block.
- Set the tag length *ulTagBits* in the parameter block.
- Set *pTag* to hold the tag data returned from C_EncryptMessage() or the final C_EncryptMessageNext().
- Call C_MessageEncryptInit() for **CKM_AES_GCM** mechanism key *K*.
- Call C_EncryptMessage(), or C_EncryptMessageBegin() followed by C_EncryptMessageNext()*[2]. The mechanism parameter is passed to all three of these functions.
- Call C_MessageEncryptFinal() to close the message decryption.

MessageDecrypt:

- Set the IV length *ulIvLen* in the parameter block.
- Set the IV data *pIv* in the parameter block.
- The *ulIvFixedBits* and *ivGenerator* fields are ignored.
- Set the tag length *ulTagBits* in the parameter block.
- Set the tag data *pTag* in the parameter block before C_DecryptMessage() or the final C_DecryptMessageNext().
- Call C_MessageDecryptInit() for **CKM_AES_GCM** mechanism key *K*.
- Call C_DecryptMessage(), or C_DecryptMessageBegin followed by C_DecryptMessageNext()*[3]. The mechanism parameter is passed to all three of these functions.
- Call C_MessageDecryptFinal() to close the message decryption.

In *pIv* the least significant bit of the initialization vector is the rightmost bit. *ulIvLen* is the length of the initialization vector in bytes.

On MessageEncrypt, the meaning of *ivGenerator* is as follows: CKG_NO_GENERATE means the IV is passed in on MessageEncrypt and no internal IV generation is done. CKG_GENERATE means that the non-fixed portion of the IV is generated by the module internally. The generation method is not defined.

2 "*" indicates 0 or more calls may be made as required

3 "*" indicates 0 or more calls may be made as required

CKG_GENERATE_COUNTER means that the non-fixed portion of the IV is generated by the module internally by use of an incrementing counter, the initial IV counter is zero.

CKG_GENERATE_COUNTER_XOR means that the non-fixed portion of the IV is xored with a counter. The value of the non-fixed portion passed must not vary from call to call. Like CKG_GENERATE_COUNTER, the counter starts at zero.

CKG_GENERATE_RANDOM means that the non-fixed portion of the IV is generated by the module internally using a PRNG. In any case the entire IV, including the fixed portion, is returned in *pIV*.

Modules must implement CKG_GENERATE. Modules may also reject *ulIvFixedBits* values which are too large. Zero is always an acceptable value for *ulIvFixedBits*.

In Encrypt and Decrypt the tag is appended to the cipher text and the least significant bit of the tag is the rightmost bit and the tag bits are the rightmost *ulTagBits* bits. In MessageEncrypt the tag is returned in the *pTag* field of CK_GCM_MESSAGE_PARAMS. In MesssageDecrypt the tag is provided by the *pTag* field of CK_GCM_MESSAGE_PARAMS.

The key type for *K* must be compatible with **CKM_AES_ECB** and the C_EncryptInit()/C_DecryptInit()/C_MessageEncryptInit()/C_MessageDecryptInit() calls shall behave, with respect to *K*, as if they were called directly with **CKM_AES_ECB**, *K* and NULL parameters.

1.3.3 AES-GCM Authenticated Wrap / Unwrap

Generic GCM mode is described in [GCM]. To set up for AES-GCM use the following process, where w*K* (wrapping key) and *AAD* (additional authenticated data) are as described in [GCM]. AES-GCM uses CK_GCM_WRAP_PARAMS for WrapKey, UnWrapkey and CK_GCM_MESSAGE_PARAMS for WrapMessageKey, and UnWrapMessageKey.

Wrap:

- Set the IV length *ulIvLen* in the parameter block.
- Set *pIv* to hold the IV data returned from C_Wrapkey() . If *ulIvFixedBits* is not zero, then the most significant bits of *pIV* contain the fixed IV. If *ivGenerator* is set to CKG_NO_GENERATE, *pIv* is an input parameter with the full IV.
- Set the *ulIvFixedBits* and *ivGenerator* fields in the parameter block.
- Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD m*ay be NULL if *ulAADLen* is 0.
- Set the tag length *ulTagBits* in the parameter block.
- Call C_WrapKey() for **CKM_AES_GCM** mechanism with parameters and wrapping key w*K* and key to be wrapped *K*, obtaining a wrapped key and authentication tag output.

UnWrap:

- Set the IV length *ulIvLen* in the parameter block.
- Set the IV data *pIv* in the parameter block..
- The *ulIvFixedBits* and *ivGenerator* fields are ignored.
- Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD m*ay be NULL if ulAADLen is 0.
- Set the tag length *ulTagBits* in the parameter block.
- Call C_UnWrapKey() for **CKM_AES_GCM** mechanism with parameters and wrapping key *K* and wrapped key+ authenticated tag output from wrap,, template for the new key, obtaining a key handle.

WrapKeyAuthenticated:

- Set the IV length *ulIvLen* in the parameter block.
- Set *pIv* to hold the IV data returned from C_Wrapkey() . If *ulIvFixedBits* is not zero, then the most significant bits of *pIV* contain the fixed IV. If *ivGenerator* is set to CKG_NO_GENERATE, *pIv* is an input parameter with the full IV.
- Set the *ulIvFixedBits* and *ivGenerator* fields in the parameter block.
- Set the tag length *ulTagBits* in the parameter block.
- Set *pTag* to hold the tag data returned from C_WrapKeyAuthenticated().
- Call C_WrapMessageKey() for **CKM_AES_GCM** mechanism wrapping key w*K*. wrapped key mechanism, parameters and obtaining a wrapped key and authentication tag output in the parameter block.
-

UnWrapKeyAuthenticated:

- Set the IV length *ulIvLen* in the parameter block.
- Set the IV data *pIv* in the parameter block.
- The *ulIvFixedBits* and *ivGenerator* fields are ignored.
- Set the tag length *ulTagBits* in the parameter block.
- Set the tag data *pTag* in the parameter block
- Call C_UnWrapKeyAuthenticated() for **CKM_AES_GCM** mechanism, -wrapping key w*K*, Wrapped key, parameter, template for the new key, obtaining a key handle.

In pIv the least significant bit of the initialization vector is the rightmost bit. ulIvLen is the length of the initialization vector in bytes.

On WrapKeyAuthenticated, the meaning of ivGenerator is as follows: CKG_NO_GENERATE means the IV is passed in on MessageEncrypt and no internal IV generation is done. CKG_GENERATE means that the non-fixed portion of the IV is generated by the module internally. The generation method is not defined.

CKG_GENERATE_COUNTER means that the non-fixed portion of the IV is generated by the module internally by use of an incrementing counter, the initial IV counter is zero.

CKG_GENERATE_COUNTER_XOR means that the non-fixed portion of the IV is xored with a counter. The value of the non-fixed portion passed must not vary from call to call. Like CKG_GENERATE_COUNTER, the counter starts at zero.

CKG_GENERATE_RANDOM means that the non-fixed portion of the IV is generated by the module internally using a PRNG. In any case the entire IV, including the fixed portion, is returned in pIV.

Modules must implement CKG_GENERATE. Modules may also reject ulIvFixedBits values which are too large. Zero is always an acceptable value for ulIvFixedBits.

In Encrypt and Decrypt the tag is appended to the cipher text and the least significant bit of the tag is the rightmost bit and the tag bits are the rightmost ulTagBits bits. In MessageEncrypt the tag is returned in the pTag field of CK_GCM_MESSAGE_PARAMS. In MesssageDecrypt the tag is provided by the pTag field of CK_GCM_MESSAGE_PARAMS.

The key type for K must be compatible with CKM_AES_ECB and the C_WrapKey()/C_UNWrapKey()/C_WrapMessageKey()/C_UnWrapMessageKey() calls shall behave, with respect to K, as if they were called directly with CKM_AES_ECB, K and NULL parameters.

1.3.31.3.4 AES-CCM authenticated Encryption / Decryption

For IPsec (RFC 4309) and also for use in ZFS encryption. Generic CCM mode is described in [RFC 3610].

To set up for AES-CCM use the following process, where *K* (key), nonce and additional authenticated data are as described in [RFC 3610]. AES-CCM uses CK_CCM_PARAMS for Encrypt and Decrypt, and CK_CCM_MESSAGE_PARAMS for MessageEncrypt and MessageDecrypt.

Encrypt:

- Set the message/data length *ulDataLen* in the parameter block.
- Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block.
- Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD* may be NULL if *ulAADLen* is 0.
- Set the MAC length *ulMACLen* in the parameter block.
- Call C_EncryptInit() for **CKM_AES_CCM** mechanism with parameters and key *K*.
- Call C_Encrypt(), C_EncryptUpdate(), or C_EncryptFinal(), for the plaintext obtaining the final ciphertext output and the MAC. The total length of data processed must be *ulDataLen*. The output length will be *ulDataLen* + *ulMACLen.*

Decrypt:

- Set the message/data length *ulDataLen* in the parameter block. This length must not include the length of the MAC that is appended to the cipher text.
- Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block.
- Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD m*ay be NULL if *ulAADLen* is 0.
- Set the MAC length *ulMACLen* in the parameter block.
- Call C_DecryptInit() for **CKM_AES_CCM** mechanism with parameters and key *K*.
- Call C_Decrypt(), C_DecryptUpdate(), or C_DecryptFinal(), for the ciphertext, including the appended MAC, obtaining plaintext output. The total length of data processed must be *ulDataLen* + *ulMACLen*. Note: since **CKM_AES_CCM** is an AEAD cipher, no data should be returned until C_Decrypt() or C_DecryptFinal().

MessageEncrypt:

- Set the message/data length *ulDataLen* in the parameter block.
- Set the nonce length *ulNonceLen*.
- Set *pNonce* to hold the nonce data returned from C_EncryptMessage() and C_EncryptMessageBegin(). If *ulNonceFixedBits* is not zero, then the most significant bits of *pNonce* contain the fixed nonce. If *nonceGenerator* is set to CKG_NO_GENERATE, *pNonce* is an input parameter with the full nonce.
- Set the *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block.
- Set the MAC length *ulMACLen* in the parameter block.
- Set *pMAC* to hold the MAC data returned from C_EncryptMessage() or the final C_EncryptMessageNext().
- Call C_MessageEncryptInit() for **CKM_AES_CCM** mechanism key *K*.
- Call C_EncryptMessage(), or C_EncryptMessageBegin() followed by C_EncryptMessageNext()*[4]. The mechanism parameter is passed to all three functions.
- Call C_MessageEncryptFinal() to close the message encryption.

---

4 “*” indicates 0 or more calls may be made as required

- The MAC is returned in *pMac* of the CK_CCM_MESSAGE_PARAMS structure.

MessageDecrypt:

- Set the message/data length *ulDataLen* in the parameter block.
- Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block
- The *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block are ignored.
- Set the MAC length *ulMACLen* in the parameter block.
- Set the MAC data *pMAC* in the parameter block before C_DecryptMessage() or the final C_DecryptMessageNext().
- Call C_MessageDecryptInit() for **CKM_AES_CCM** mechanism key *K*.
- Call C_DecryptMessage(), or C_DecryptMessageBegin() followed by C_DecryptMessageNext()*[5]. The mechanism parameter is passed to all three functions.
- Call C_MessageDecryptFinal() to close the message decryption.

In *pNonce* the least significant bit of the nonce is the rightmost bit. *ulNonceLen* is the length of the nonce in bytes.

On MessageEncrypt, the meaning of *nonceGenerator* is as follows: CKG_NO_GENERATE means the nonce is passed in on MessageEncrypt and no internal MAC generation is done. CKG_GENERATE means that the non-fixed portion of the nonce is generated by the module internally. The generation method is not defined.

CKG_GENERATE_COUNTER means that the non-fixed portion of the nonce is generated by the module internally by use of an incrementing counter, the initial IV counter is zero.

CKG_GENERATE_COUNTER_XOR means that the non-fixed portion of the IV is xored with a counter. The value of the non-fixed portion passed must not vary from call to call. Like CKG_GENERATE_COUNTER, the counter starts at zero.

CKG_GENERATE_RANDOM means that the non-fixed portion of the nonce is generated by the module internally using a PRNG. In any case the entire nonce, including the fixed portion, is returned in *pNonce*.

Modules must implement CKG_GENERATE. Modules may also reject *ulNonceFixedBits* values which are too large. Zero is always an acceptable value for *ulNonceFixedBits*.

In Encrypt and Decrypt the MAC is appended to the cipher text and the least significant byte of the MAC is the rightmost byte and the MAC bytes are the rightmost *ulMACLen* bytes. In MessageEncrypt the MAC is returned in the *pMAC* field of CK_CCM_MESSAGE_PARAMS. In MesssageDecrypt the MAC is provided by the *pMAC* field of CK_CCM_MESSAGE_PARAMS.

The key type for K must be compatible with **CKM_AES_ECB** and the C_EncryptInit()/C_DecryptInit()/C_MessageEncryptInit()/C_MessageDecryptInit() calls shall behave, with respect to K, as if they were called directly with **CKM_AES_ECB**, K and NULL parameters.

1.3.5 AES-CCM Authenticated Wrap / Unwrap

To set up for AES-CCM use the following process, where *K* (key), nonce and additional authenticated data are as described in [RFC 3610]. AES-CCM uses CK_CCM_WAP_PARAMS for WrapKey and UnWrapKey, and CK_CCM_MESSAGE_PARAMS for WrapKeyAuthenticated and UnWrapKeyAuthenticated.

Wrap:

- Set the message/data length *ulDataLen* in the parameter block.

5 "*" indicates 0 or more calls may be made as required

- Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block.
- Set *pNonce* to hold the nonce data returned from C_WrapKey(). If *ulNonceFixedBits* is not zero, then the most significant bits of *pNonce* contain the fixed nonce. If *nonceGenerator* is set to CKG_NO_GENERATE, *pNonce* is an input parameter with the full nonce.
- Set the *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block.
- Set the MAC length *ulMACLen* in the parameter block.
- Call C_WarpKey() for **CKM_AES_CCM** mechanism with parameters wrapping key w*K, key to be wrapped mK*, obtaining the final Wrappedkey output and the MAC. The total length of data processed must be *ulDataLen*. The output length will be *ulDataLen + ulMACLen.*

UnWrap:

- Set the message/data length *ulDataLen* to Zero in the parameter block.This returns a key handle.
- Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block.
- The *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block are ignored.
- Set the AAD data *pAAD* and size *ulAADLen* in the parameter block. *pAAD m*ay be NULL if *ulAADLen* is 0.
- Set the MAC length *ulMACLen* in the parameter block.
- Call C_UnwrapKey() for **CKM_AES_CCM** mechanism with parameters, unwrapping key w*K, template, and wrapped key*. Including the appended MAC, obtaining a obtaining a new key handle

WrapKeyAuthenticated:

- Set the message/data length *ulDataLen* in the parameter block.
- Set the nonce length *ulNonceLen*.
- Set *pNonce* to hold the nonce data returned from C_WrapKeyAuthenticated(). If *ulNonceFixedBits* is not zero, then the most significant bits of *pNonce* contain the fixed nonce. If *nonceGenerator* is set to CKG_NO_GENERATE, *pNonce* is an input parameter with the full nonce.
- Set the *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block.
- Set the MAC length *ulMACLen* in the parameter block.
- Set *pMAC* to hold the MAC data returned from C_WrapkeyAuthenticated()
- Call C_WrapKeyAuthenticated() for **CKM_AES_CCM** mechanism wrapping key w*K* the key to be wrapped *mK*, parameter block
- The MAC is returned in *pMac* of the CK_CCM_MESSAGE_PARAMS structure.

UnWrapKeyAuthenicated:

- Set the message/data length *ulDataLen to Zero as a key handle will be returned*.
- Set the nonce length *ulNonceLen* and the nonce data *pNonce* in the parameter block
- The *ulNonceFixedBits* and *nonceGenerator* fields in the parameter block are ignored.
- Set the MAC length *ulMACLen* in the parameter block.
- Set the MAC data *pMAC* in the parameter block before C_UnWrapKeyAuthenticated().
- Call C_UnWrapMessageKey() for **CKM_AES_CCM** mechanism key w*K*, wrapped key mK, parameter block and template, obtaining a new key handle.

In *pNonce* the least significant bit of the nonce is the rightmost bit. *ulNonceLen* is the length of the nonce in bytes.

On MessageEncrypt, the meaning of *nonceGenerator* is as follows: CKG_NO_GENERATE means the nonce is passed in on MessageEncrypt and no internal MAC generation is done. CKG_GENERATE means that the non-fixed portion of the nonce is generated by the module internally. The generation method is not defined.

CKG_GENERATE_COUNTER means that the non-fixed portion of the nonce is generated by the module internally by use of an incrementing counter, the initial IV counter is zero.

CKG_GENERATE_COUNTER_XOR means that the non-fixed portion of the IV is xored with a counter. The value of the non-fixed portion passed must not vary from call to call. Like CKG_GENERATE_COUNTER, the counter starts at zero.

CKG_GENERATE_RANDOM means that the non-fixed portion of the nonce is generated by the module internally using a PRNG. In any case the entire nonce, including the fixed portion, is returned in *pNonce*.

Modules must implement CKG_GENERATE. Modules may also reject *ulNonceFixedBits* values which are too large. Zero is always an acceptable value for *ulNonceFixedBits*.

In Encrypt and Decrypt, the MAC is appended to the cipher text and the least significant byte of the MAC is the rightmost byte and the MAC bytes are the rightmost *ulMACLen* bytes. In MessageEncrypt, the MAC is returned in the *pMAC* field of CK_CCM_MESSAGE_PARAMS. In MesssageDecrypt, the MAC is provided by the *pMAC* field of CK_CCM_MESSAGE_PARAMS.

The key type for K must be compatible with **CKM_AES_ECB** and the C_WrapKey()/C_UnWrapKey()/C_WrapMessageKey()/C_UnWrapMessageKey() calls shall behave, with respect to K, as if they were called directly with **CKM_AES_ECB**, K and NULL parameters

~~1.3.4~~1.3.6 AES GCM and CCM Mechanism parameters

♦ CK_GENERATOR_FUNCTION

Functions to generate unique IVs and nonces.

```
typedef CK_ULONG CK_GENERATOR_FUNCTION;
```

♦ CK_GCM_PARAMS; CK_GCM_PARAMS_PTR

CK_GCM_PARAMS is a structure that provides the parameters to the CKM_AES_GCM mechanism when used for Encrypt or Decrypt. It is defined as follows:

```
typedef struct CK_GCM_PARAMS {
   CK_BYTE_PTR   pIv;
   CK_ULONG      ulIvLen;
   CK_ULONG      ulIvBits;
   CK_BYTE_PTR   pAAD;
   CK_ULONG      ulAADLen;
   CK_ULONG      ulTagBits;
}  CK_GCM_PARAMS;
```

The fields of the structure have the following meanings:

pIv	pointer to initialization vector
ulIvLen	length of initialization vector in bytes. The length of the initialization vector can be any number between 1 and (2^32) - 1. 96-bit (12 byte) IV values can be processed more efficiently, so that length is recommended for situations in which efficiency is critical.
ulIvBits	length of initialization vector in bits. Do no use ulIvBits to specify the length of the initialization vector, but ulIvLen instead.

pAAD	pointer to additional authentication data. This data is authenticated but not encrypted.
ulAADLen	length of pAAD in bytes. The length of the AAD can be any number between 0 and (2^32) – 1.
ulTagBits	length of authentication tag (output following cipher text) in bits. Can be any value between 0 and 128.

CK_GCM_PARAMS_PTR is a pointer to a **CK_GCM_PARAMS**.

♦ CK_GCM_MESSAGE_PARAMS; CK_GCM_MESSAGE_PARAMS_PTR

CK_GCM_MESSAGE_PARAMS is a structure that provides the parameters to the CKM_AES_GCM mechanism when used for MessageEncrypt or MessageDecrypt. It is defined as follows:

```
typedef struct CK_GCM_MESSAGE_PARAMS {
   CK_BYTE_PTR   pIv;
   CK_ULONG      ulIvLen;
   CK_ULONG      ulIvFixedBits;
   CK_GENERATOR_FUNCTION   ivGenerator;
   CK_BYTE_PTR   pTag;
   CK_ULONG      ulTagBits;
}  CK_GCM_MESSAGE_PARAMS;
```

The fields of the structure have the following meanings:

pIv	pointer to initialization vector
ulIvLen	length of initialization vector in bytes. The length of the initialization vector can be any number between 1 and (2^32) - 1. 96-bit (12 byte) IV values can be processed more efficiently, so that length is recommended for situations in which efficiency is critical.
ulIvFixedBits	number of bits of the original IV to preserve when generating an new IV. These bits are counted from the Most significant bits (to the right).
ivGenerator	Function used to generate a new IV. Each IV must be unique for a given session.
pTag	location of the authentication tag which is returned on MessageEncrypt, and provided on MessageDecrypt.
ulTagBits	length of authentication tag in bits. Can be any value between 0 and 128.

CK_GCM_MESSAGE_PARAMS_PTR is a pointer to a **CK_GCM_MESSAGE_PARAMS**.

♦ CK_GCM_WRAP_PARAMS; CK_GCM_WRAP_PARAMS_PTR

CK_GCM_MESSAGE_PARAMS is a structure that provides the parameters to the CKM_AES_GCM mechanism when used for C_WrapKey to ~~provide~~ return a token generated IV for input into C_UnWrapKey. It is defined as follows:

```
typedef struct CK_GCM_WRAP_PARAMS {
   CK_BYTE_PTR   pIv;
   CK_ULONG      ulIvLen;
   CK_ULONG      ulIvFixedBits;
```

```
    CK_GENERATOR_FUNCTION    ivGenerator;
    CK_BYTE_PTR   pAAD;
    CK_ULONG      ulAADLen;
    CK_ULONG      ulTagBits;
}   CK_GCM_WRAP_PARAMS;
```

The fields of the structure have the following meanings:

pIv	pointer to initialization vector
ulIvLen	length of initialization vector in bytes. The length of the initialization vector can be any number between 1 and (2^32) - 1. 96-bit (12 byte) IV values can be processed more efficiently, so that length is recommended for situations in which efficiency is critical.
ulIvFixedBits	number of bits of the original IV to preserve when generating an new IV. These bits are counted from the Most significant bits (to the right).
ivGenerator	Function used to generate a new IV. Each IV must be unique for a given session.
pAAD	pointer to additional authentication data. This data is authenticated but not encrypted.
ulAADLen	length of pAAD in bytes. The length of the AAD can be any number between 0 and (2^32) – 1.
ulTagBits	length of authentication tag in bits. Can be any value between 0 and 128.

CK_GCM_WRAP_PARAMS_PTR is a pointer to a **CK_GCM_WRAP_PARAMS**.

♦ CK_CCM_PARAMS; CK_CCM_PARAMS_PTR

CK_CCM_PARAMS is a structure that provides the parameters to the **CKM_AES_CCM** mechanism when used for Encrypt or Decrypt. It is defined as follows:

```
typedef struct CK_CCM_PARAMS {
   CK_ULONG     ulDataLen; /*plaintext or ciphertext*/
   CK_BYTE_PTR  pNonce;
   CK_ULONG     ulNonceLen;
   CK_BYTE_PTR  pAAD;
   CK_ULONG     ulAADLen;
   CK_ULONG     ulMACLen;
}   CK_CCM_PARAMS;
```

The fields of the structure have the following meanings, where L is the size in bytes of the data length's length ($2 \le L \le 8$):

ulDataLen	length of the data where 0 ≤ ulDataLen < 2^(8L).
pNonce	the nonce.
ulNonceLen	length of pNonce in bytes where 7 ≤ ulNonceLen ≤ 13.
pAAD	Additional authentication data. This data is authenticated but not encrypted.
ulAADLen	length of pAAD in bytes where 0 ≤ ulAADLen ≤ (2^32) - 1.

ulMACLen length of the MAC (output following cipher text) in bytes. Valid values are 4, 6, 8, 10, 12, 14, and 16.

CK_CCM_PARAMS_PTR is a pointer to a **CK_CCM_PARAMS**.

♦ CK_CCM_MESSAGE_PARAMS; CK_CCM_MESSAGE_PARAMS_PTR

CK_CCM_MESSAGE_PARAMS is a structure that provides the parameters to the **CKM_AES_CCM** mechanism when used for MessageEncrypt or MessageDecrypt. It is defined as follows:

```
typedef struct CK_CCM_MESSAGE_PARAMS {
   CK_ULONG     ulDataLen; /*plaintext or ciphertext*/
   CK_BYTE_PTR  pNonce;
   CK_ULONG     ulNonceLen;
   CK_ULONG     ulNonceFixedBits;
   CK_GENERATOR_FUNCTION   nonceGenerator;
   CK_BYTE_PTR  pMAC;
   CK_ULONG     ulMACLen;
}  CK_CCM_MESSAGE_PARAMS;
```

The fields of the structure have the following meanings, where L is the size in bytes of the data length's length ($2 \leq L \leq 8$):

ulDataLen length of the data where 0 ≤ ulDataLen < 2^(8L).

pNonce the nonce.

ulNonceLen length of pNonce in bytes where 7 ≤ ulNonceLen ≤ 13.

ulNonceFixedBits number of bits of the original nonce to preserve when generating a new nonce. These bits are counted from the Most significant bits (to the right).

nonceGenerator Function used to generate a new nonce. Each nonce must be unique for a given session.

pMAC location of the CCM MAC returned on MessageEncrypt, provided on MessageDecrypt

ulMACLen length of the MAC (output following cipher text) in bytes. Valid values are 4, 6, 8, 10, 12, 14, and 16.

CK_CCM_MESSAGE_PARAMS_PTR is a pointer to a **CK_CCM_MESSAGE_PARAMS**.

♦ CK_CCM_WRAP_PARAMS; CK_CCM_WAP_PARAMS_PTR

CK_CCM_PARAMS is a structure that provides the parameters to the **CKM_AES_CCM** mechanism when used for C_WrapKey only to provide a token generated nonce and the number of bits to ~~preservse~~. It is defined as follows:

```
typedef struct CK_CCM_PARAMS {
   CK_ULONG     ulDataLen; /*wrappedkey data*/
   CK_BYTE_PTR  pNonce;
   CK_ULONG     ulNonceLen;
   CK_ULONG     ulNonceFixedBits;
   CK_GENERATOR_FUNCTION   nonceGenerator;
   CK_BYTE_PTR  pAAD;
```

```
    CK_ULONG      ulAADLen;
    CK_ULONG      ulMACLen;
}   CK_CCM_PARAMS;
```

The fields of the structure have the following meanings, where L is the size in bytes of the data length's length ($2 \le L \le 8$):

ulDataLen	length of the data where $0 \le ulDataLen < 2^{8L}$.
pNonce	the nonce.
ulNonceLen	length of pNonce in bytes where $7 \le ulNonceLen \le 13$.
ulNonceFixedBits	number of bits of the original nonce to preserve when generating a new nonce. These bits are counted from the Most significant bits (to the right).
nonceGenerator	Function used to generate a new nonce. Each nonce must be unique for a given session.
pAAD	Additional authentication data. This data is authenticated but not wrapped.
ulAADLen	length of pAAD in bytes where $0 \le ulAADLen \le (2^{32}) - 1$.
ulMACLen	length of the MAC (output following cipher text) in bytes. Valid values are 4, 6, 8, 10, 12, 14, and 16.

CK_CCM_WRAP_PARAMS_PTR is a pointer to a **CK_CCM_WRAP_PARAMS**