PKCS #11 Message-Based Encryption and Decryption

Wan-Teh Chang <wtc@google.com>, 2014-05-02

Introduction

Message-based encryption refers to the process of encrypting multiple messages using the same encryption mechanism and encryption key. The encryption mechanism can be either an authenticated encryption with associated data (AEAD) algorithm or a pure encryption algorithm.

The main goal of this proposal is to optimize message-based encryption by reducing the number of PKCS #11 calls required. A second goal is to allow a crypto token to generate the initialization vector (IV) or nonce for the encryption mechanism when encrypting each message.

This proposal incorporates ideas of Michael St.Johns and Bob Relyea. The API design aims for simplicity, easy of use, and readability.

Proposal

The new functions for message-based encryption are specified. A new AES-GCM mechanism is also specified as a concrete example.

C_MessagedEncryptInit

CK_RV C_MessagedEncryptInit(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey
);

The **C_MessagedEncryptInit** function prepares a session for one or more encryption operations that use the same encryption mechanism and encryption key. We use the term **message-based encryption** to refer to the process of encrypting multiple messages using the same mechanism and key.

**C_MessagedEncryptInit** is an opportunity to perform any setup common to the encryption of multiple messages, such as key expansion (generation of round keys according to a key schedule). After calling **C_MessagedEncryptInit**, an application can call **C_EncryptMessage** to encrypt a message in a single part, or call **C_EncryptMessageBegin** and
**C_EncryptMessageNext** to encrypt a message in multiple parts. This may be repeated several times. Finally, an application calls **C_MessageEncryptFinal** to finish the message-based encryption process.

An encryption mechanism may request that the crypto token generates the initialization vectors (IVs) or nonces used in subsequent encryption operations by specifying an IV generator mechanism in the encryption mechanism’s parameter.

---

**C_EncryptMessage**

```c
CK_RV C_EncryptMessage(  
    CK_SESSION_HANDLE hSession,  
    CK_VOID_PTR pParameter,  
    CKULONG ulParameterLen,  
    CK_BYTE_PTR pAssociatedData,  
    CKULONG ulAssociatedDataLen,  
    CK_BYTE_PTR pPlaintext,  
    CKULONG ulPlaintextLen,  
    CK_BYTE_PTR pCiphertext,  
    CKULONG_PTR pulCiphertextLen
);
```

The **C_EncryptMessage** function encrypts a message in a single part. It does not finish the message-based encryption process. Additional **C_EncryptMessage** or **C_EncryptMessageBegin** and **C_EncryptMessageNext** calls may be made on the session.

`pParameter` and `ulParameterLen` specify any mechanism-specific parameters for the message encryption operation. Typically this is an initialization vector (IV) or nonce. Depending on the mechanism parameter passed to **C_MessageEncryptInit**, `pParameter` may be either an input or an output parameter. For example, if the mechanism parameter specifies an IV generator mechanism, the IV generated by the IV generator will be output to the `pParameter` buffer.

`pAssociatedData` and `ulAssociatedDataLen` specify the associated data for an AEAD mechanism. If the mechanism is not AEAD, `pAssociatedData` and `ulAssociatedDataLen` are not used and should be set to (NULL, 0).

---

**C_EncryptMessageBegin**

```c
CK_RV C_EncryptMessageBegin(  
    CK_SESSION_HANDLE hSession,
```
The `C_EncryptMessageBegin` function begins a multiple-part message encryption operation. It is to be followed by one or more `C_EncryptMessageNext` calls.

`pParameter` and `ulParameterLen` specify any mechanism-specific parameters for the message encryption operation. Typically this is an initialization vector (IV) or nonce. Depending on the mechanism parameter passed to `C_MessageEncryptInit`, `pParameter` may be either an input or an output parameter. For example, if the mechanism parameter specifies an IV generator mechanism, the IV generated by the IV generator will be output to the `pParameter` buffer.

`pAssociatedData` and `ulAssociatedDataLen` specify the associated data for an AEAD mechanism. If the mechanism is not AEAD, `pAssociatedData` and `ulAssociatedDataLen` are not used and should be set to (NULL, 0).

---

### `C_EncryptMessageNext`

```c
CK_RV C_EncryptMessageNext(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pPlaintextPart,
    CK_ULONG ulPlaintextPartLen,
    CK_BYTE_PTR pCiphertextPart,
    CK_ULONG_PTR pulCiphertextPartLen,
    CK_FLAGS flags
);
```

# define CKF_END_OF_MESSAGE 0x00000001

The `C_EncryptMessageNext` function continues a multiple-part message encryption operation. The `flags` argument is set to 0 if there is more plaintext data to follow, or set to `CKF_END_OF_MESSAGE` if this is the last plaintext part. Although the last `C_EncryptMessageNext` call ends the encryption of a message, it does not finish the message-based encryption process. Additional `C_EncryptMessage` or `C_EncryptMessageBegin` and `C_EncryptMessageNext` calls may be made on the session.
C_MessageEncryptFinal

CK_RV C_MessageEncryptFinal(
    CK_SESSION_HANDLE hSession
);

The C_MessageEncryptFinal function finishes a message-based encryption process initiated by an earlier C_MessageEncryptInit call.

C_MessageDecryptInit

CK_RV C_MessageDecryptInit(
    CK_SESSION_HANDLE hSession,
    CK_MECHANISM_PTR pMechanism,
    CK_OBJECT_HANDLE hKey
);

The C_MessageDecryptInit function prepares a session for one or more decryption operations that use the same decryption mechanism and decryption key. We use the term message-based decryption to refer to the process of decrypting multiple messages using the same mechanism and key.

C_MessageDecryptInit is an opportunity to perform any setup common to the decryption of multiple messages, such as key expansion (generation of round keys according to a key schedule). After calling C_MessageDecryptInit, an application can call C_DecryptMessage to decrypt a message in a single part, or call C_DecryptMessageBegin and C_DecryptMessageNext to decrypt a message in multiple parts. This may be repeated several times. Finally, an application calls C_MessageDecryptFinal to finish the message-based decryption process.

In subsequent decryption operations, IVs are provided by the caller, so the decryption mechanism’s parameter must not specify an IV generator mechanism.

Notes:

1. It is recommended or even mandated that AEAD decryption should not release any plaintext if the authenticity of the input cannot be verified. Therefore, each decryption mechanism must specify whether or not it can be used with the multiple-part message decryption functions C_DecryptMessageBegin and C_DecryptMessageNext.

C_DecryptMessage
CK_RV C_DecryptMessage(
    CK_SESSION_HANDLE hSession,
    CK_VOID_PTR pParameter,
    CK_ULONG ulParameterLen,
    CK_BYTE_PTR pAssociatedData,
    CK_ULONG ulAssociatedDataLen,
    CK_BYTE_PTR pCiphertext,
    CK_ULONG ulCiphertextLen,
    CK_BYTE_PTR pPlaintext,
    CK_ULONG_PTR pulPlaintextLen
);

The **C_DecryptMessage** function decrypts a message in a single part. It does not finish the message-based decryption process. Additional **C_DecryptMessage** or **C_DecryptMessageBegin** and **C_DecryptMessageNext** calls may be made on the session.

pParameter and ulParameterLen specify any mechanism-specific parameters for the message decryption operation. Typically this is an initialization vector (IV) or nonce. Unlike the pParameter parameter of **C_EncryptMessage**, pParameter is always an input parameter.

pAssociatedData and ulAssociatedDataLen specify the associated data for an AEAD mechanism. If the mechanism is not AEAD, pAssociatedData and ulAssociatedDataLen are not used and should be set to (NULL, 0).

If the decryption mechanism is an AEAD algorithm and the authenticity of the associated data or ciphertext cannot be verified, **C_DecryptMessage** returns the error code CRK_AEAD_DECRYPT_FAILED.

#define CKR_AEAD_Decrypt_FAILED 0x00000042

---

**C_DecryptMessageBegin**

CK_RV C_DecryptMessageBegin(
    CK_SESSION_HANDLE hSession,
    CK_VOID_PTR pParameter,
    CK_ULONG ulParameterLen,
    CK_BYTE_PTR pAssociatedData,
    CK_ULONG ulAssociatedDataLen
);

The **C_DecryptMessageBegin** function begins a multiple-part message decryption operation. It is to be followed by one or more **C_DecryptMessageNext** calls.
pParameter and ulParameterLen specify any mechanism-specific parameters for the message decryption operation. Typically this is an initialization vector (IV) or nonce. Unlike the pParameter parameter of C_EncryptMessageBegin, pParameter is always an input parameter.

pAssociatedData and ulAssociatedDataLen specify the associated data for an AEAD mechanism. If the mechanism is not AEAD, pAssociatedData and ulAssociatedDataLen are not used and should be set to (NULL, 0).

C_DecryptMessageNext

CK_RV C_DecryptMessageNext(
    CK_SESSION_HANDLE hSession,
    CK_BYTE_PTR pCiphertextPart,
    CK_ULONG ulCiphertextPartLen,
    CK_BYTE_PTR pPlaintextPart,
    CK_ULONG_PTR pulPlaintextPartLen,
    CK_FLAGS flags
);

The C_DecryptMessageNext function continues a multiple-part message decryption operation. The flags argument is set to 0 if there is more ciphertext data to follow, or set to CKF_END_OF_MESSAGE if this is the last ciphertext part. Although the last C_DecryptMessageNext call ends the decryption of a message, it does not finish the message-based decryption process. Additional C_DecryptMessage or C_DecryptMessageBegin and C_DecryptMessageNext calls may be made on the session.

If the decryption mechanism is an AEAD algorithm and the authenticity of the associated data or ciphertext cannot be verified, the last C_DecryptMessageNext call of the message returns the error code CKR_AEAD_DECRYPT_FAILED.

Notes:
1. If an AEAD mechanism allows multiple-part message decryption, a crypto token may return unauthenticated plaintext to the caller of C_DecryptMessageNext. An application must take care to not consume any of the returned plaintext until the last C_DecryptMessageNext call returns CKR_OK. To avoid application mistakes, it is recommended that AEAD mechanisms be specified to disallow multiple-part message decryption.
C_MessageDecryptFinal

CK_RV C_MessageDecryptFinal(
    CK_SESSION_HANDLE hSession
);

The **C_MessageDecryptFinal** function finishes a message-based decryption process initiated by an earlier **C_MessageDecryptInit** call.

---

AES-GCM Mechanism for Message-Based Encryption

This section specifies a new AES-GCM mechanism for message-based encryption and decryption.

---

**CKM_AES_GCM_V2**

```c
#define CKM_AES_GCM_V2 0x00000700
```

**CKM_AES_GCM_V2** is an AES-GCM mechanism for message-based encryption and decryption. It only supports single-part message encryption and decryption. It must not be used with multiple-part message encryption and decryption functions.

Notes:
1. Although only multiple-part message decryption is error-prone for **CKM_AES_GCM_V2**, for symmetry we also prohibit multiple-part message encryption with **CKM_AES_GCM_V2**.

---

**CK_GCM_PARAMS_V2**

The mechanism parameter structure for **CKM_AES_GCM_V2** is **CK_GCM_PARAMS_V2**. It is not AES-specific and therefore can also be used with Camellia-GCM.

```c
typedef struct CK_GCM_PARAMS_V2 {
    CK_ULONG ulIVLen; /* Length in bytes of the IV */
    CK_ULONG ulTagLen; /* Length in bytes of the authentication tag */
    CK_MECHANISM_PTR pIVGeneratorMech; /* IV generator mechanism */
} CK_GCM_PARAMS_V2;
```

The **pIVGeneratorMech** field specifies the IV generator mechanism for encryption.
- If NULL, the IV is provided by the caller. The **pParameter** parameter of **C_EncryptMessage** is an input.
If not NULL, it points to a CK_MECHANISM structure that specifies how the IV is generated partially or fully by the crypto token. The pParameter parameter of C_EncryptMessage is an output.

For decryption, the pIVGeneratorMech field is not used and must be set to NULL.

### IV Generator Mechanisms

This section specifies two representative IV generator mechanisms.

#### CKM_IV_GEN_DETERMINISTIC

#define CKM_IV_GEN_DETERMINISTIC 0x00000750

CKM_IV_GEN_DETERMINISTIC is an IV generator mechanism specified in NIST SP 800-38D, Sec. 8.2.1 Deterministic Construction. The fixed field is on the left and the invocation field is on the right. The caller provides the fixed field. The crypto token generates and outputs the invocation field.

#### CKM_IV_GEN_DETERMINISTIC_PARAMS

The mechanism parameter structure for CKM_IV_GEN_DETERMINISTIC is the CKM_IV_GEN_DETERMINISTIC_PARAMS structure:

```c
typedef struct CKM_IV_GEN_DETERMINISTIC_PARAMS {
    CK_BYTE_PTR pFixed;  /* the fixed field */
    CKULONG ulFixedLen;
} CKM_IV_GEN_DETERMINISTIC_PARAMS;
```

In each C_EncryptMessage call, pParameter should point to an output buffer receiving the invocation field generated by the crypto token. ulParameterLen is the length of that output buffer and should be `blocksize - ulFixedLen`, where `blocksize` is the symmetric cipher's block size (16 for AES).

#### CKM_IV_GEN_RGB_BASED

#define CKM_IV_GEN_RGB_BASED 0x00000751

CKM_IV_GEN_RGB_BASED is an IV generator mechanism specified in NIST SP 800-38D, Sec. 8.2.2 RGB-based Construction. The random field is on the left and the free field is on the right.
The caller provides the free field. The crypto token generates and outputs the random field. The free field is recommended to be empty.

**CK_IV_GEN_RGB_BASED_PARAMS**

The mechanism parameter structure for CKM_IV_GEN_RGB_BASED is the `CK_IV_GEN_RGB_BASED_PARAMS` structure:

```c
typedef struct CK_IV_GEN_RGB_BASED_PARAMS {
    CK_CONST_BYTE_PTR pFree;   /* the free field */
    CK_ULONG ulFreeLen;
} CK_IV_GEN_RGB_BASED_PARAMS;
```

In each `C_EncryptMessage` call, `pParameter` should point to an output buffer receiving the random field generated by the crypto token. `ulParameterLen` is the length of that output buffer and should be `blocksize - ulFreeLen`, where `blocksize` is the symmetric cipher's block size (16 for AES).