PKCS #11 v2.20 Amendment 3 - Draft 2

Additional PKCS#11 Mechanisms

RSA Laboratories

October 2006

Editor's note: This is the second draft of this amendment. Comments and feedback are welcome and should be sent to the Cryptoki mailing list (Cryptoki@rsasecurity.com) or the editor (pkcs-editor@rsasecurity.com)

Table of Contents

List of Tables

1 Introduction

This document is an amendment to PKCS #11 v2.20 [1] and describes extensions to PKCS #11 to support additional mechanisms.

2 Definitions

AES Advanced Encryption Standard, as defined in FIPS PUB 197.

CAMELLIA The Camellia encryption algorithm, as defined in RFC 3713.

SHA-224 The Secure Hash Algorithm with a 224-bit message digest, as defined in RFC 3874.

3 Mechanisms

The following table shows, for the mechanisms defined in this document, their support by different cryptographic operations. For any particular token, of course, a particular operation may well support only a subset of the mechanisms listed. There is also no guarantee that a token that supports one mechanism for some operation supports any other mechanism for any other operation (or even supports that same mechanism for any other operation).

Table 1, Mechanisms vs. Functions

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR[1]	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_SHA224				✓			
CKM_SHA224_HMAC		✓					
CKM_SHA224_HMAC_GENERAL		✓					
CKM_SHA224_RSA_PKCS		✓					
CKM_SHA224_RSA_PKCS_PSS		✓					
CKM_SHA224_KEY_DERIVATION							✓
CKM_AES_CTR	✓					✓	
CKM_CAMELLIA_KEY_GEN					✓		
CKM_CAMELLIA_ECB	✓					✓	
CKM_CAMELLIA_CBC	✓					✓	
CKM_CAMELLIA_CBC_PAD	✓					✓	
CKM_CAMELLIA_MAC_GENERAL		✓					
CKM_CAMELLIA_MAC		✓					
CKM_CAMELLIA_ECB_ENCRYPT_DATA							✓
CKM_CAMELLIA_CBC_ENCRYPT_DATA							✓

The remainder of this section will present in detail the mechanisms and the parameters which are supplied to them.

3.1 RSA additional variants

For completeness and consistency with all the other SHA variants the following additions have been made to include the SHA-224 variant of these mechanisms.

3.1.1 Definitions

Mechanisms:

```
CKM_SHA224_RSA_PKCS
CKM_SHA224_RSA_PKCS_PSS
```

3.1.2 PKCS #1 RSA OAEP mechanism parameters

The following table lists the added MGF functions.

Table 2, PKCS #1 Mask Generation Functions

Source Identifier	Value
CKG_MGF1_SHA224	0x00000005

3.1.3 PKCS #1 v1.5 RSA signature with SHA-224

The PKCS #1 v1.5 RSA signature with SHA-224 mechanism, denoted **CKM_SHA224_RSA_PKCS** perform similarly as the other **CKM_SHA*X*_RSA_PKCS** mechanisms but using the SHA-224 hash functions.

3.1.4 PKCS #1 RSA PSS signature with SHA-224

The PKCS #1 RSA PSS signature with SHA-224 mechanism, denoted **CKM_SHA224_RSA_PKCS_PSS**, perform similarly as the other **CKM_SHA*X*_RSA_PSS** mechanisms but using the SHA-224, hash functions.

3.2 SHA-224

3.2.1 Definitions

Mechanisms:

```
CKM_SHA224
CKM_SHA224_HMAC
CKM_SHA224_HMAC_GENERAL
CKM_SHA224_KEY_DERIVATION
```

3.2.2 SHA-224 digest

The SHA-224 mechanism, denoted **CKM_SHA224**, is a mechanism for message digesting, following the Secure Hash Algorithm with a 224-bit message digest defined in [2].

It does not have a parameter.

Constraints on the length of input and output data are summarized in the following table. For single-part digesting, the data and the digest may begin at the same location in memory.

Table 3, SHA-224: Data Length

Function	Input length	Digest length
C_Digest	any	28

3.2.3 General-length SHA-224-HMAC

The general-length SHA-224-HMAC mechanism, denoted **CKM_SHA224_HMAC_GENERAL**, is the same as the general-length SHA-1-HMAC mechanism except that it uses the HMAC construction based on the SHA-224 hash function and length of the output should be in the range 0-28 The keys it uses are generic secret keys. FIPS-198 compliant tokens may require the key length to be at least 14 bytes; that is, half the size of the SHA-224 hash output.

It has a parameter, a **CK_MAC_GENERAL_PARAMS**, which holds the length in bytes of the desired output. This length should be in the range 0-28 (the output size of SHA-224 is 28 bytes). FIPS-198 compliant tokens may constrain the output length to be at least 4 or 14 (half the maximum length). Signatures (MACs) produced by this mechanism will be taken from the start of the full 28 byte HMAC output.

Table 4, General-length SHA-224-HMAC: Key And Data Length

Function	Key type	Data length	Signature length
C_Sign	generic secret	Any	0-28, depending on parameters
C_Verify	generic secret	Any	0-28, depending on parameters

3.2.4 SHA-224-HMAC

The SHA-224-HMAC mechanism, denoted **CKM_SHA224_HMAC**, is a special case of the general-length SHA-224-HMAC mechanism.

It has no parameter, and always produces an output of length 28.

3.2.5 SHA-224 key derivation

SHA-224 key derivation, denoted **CKM_SHA224_KEY_DERIVATION**, is the same as the SHA-1 key derivation mechanism in Section **Error! Reference source not found.**, except that it uses the SHA-224 hash function and the relevant length is 28 bytes.

3.3 AES with Counter

3.3.1 Definitions

Mechanisms:

CKM_AES_CTR

3.3.2 AES with Counter mechanism parameters

♦ **CK_ AES_CTR _PARAMS; CK_ AES_CTR _PARAMS_PTR**

CK_ AES_CTR _PARAMS is a structure that provides the parameters to the **CKM_AES_CTR** mechanism. It is defined as follows:

```
typedef struct CK_AES_CTR_PARAMS {
        CK_ULONG ulCounterBits;
        CK_BYTE cb[16];
} CK_AES_CTR_PARAMS;
```

ulCounterBits specifies the number of bits in the counter block (cb) that shall be incremented. This number shall be such that 0 < **ulCounterBits** <= 128. For any values outside this range the mechanism shall return **CKR_MECHANISM_PARAM_INVALID**.

It's up to the caller to initialize all of the bits in the counter block including the counter bits. The counter bits are the least significant bits of the counter block (cb). They are a big-endian value usually starting with 1. The rest of 'cb' is for the nonce, and maybe an optional IV.

E.g. as defined in RFC 3686 [4]:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Nonce                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                  Initialization Vector (IV)                   |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                         Block Counter                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

This construction permits each packet to consist of up to 2^{32}-1 blocks = 4,294,967,295 blocks = 68,719,476,720 octets.

CK_ AES_CTR _PARAMS_PTR is a pointer to a **CK_ AES_CTR _PARAMS**.

3.3.3 AES with Counter Encryption / Decryption

Generic AES counter mode is described in NIST Special Publication 800-38A [3], and in RFC 3686 [4]. These describe encryption using a counter block which may include a nonce to guarantee uniqueness of the counter block. Since the nonce is not incremented, the mechanism parameter must specify the number of counter bits in the counter block.

The block counter is incremented by 1 after each block of plaintext is processed. There is no support for any other increment functions in this mechanism.

If an attempt to encrypt/decrypt is made which will cause an overflow of the counter block's counter bits to be used then the mechanism shall return **CKR_DATA_LEN_RANGE**. Note that the mechanism should allow the final post increment of the counter to overflow (if it implements it this way) but not allow any further processing after this point. E.g. if ulCounterBits = 2 and the counter bits start as 1 then only 3 blocks of data can be processed.

3.4 CAMELLIA

Camellia is a block cipher with 128-bit block size and 128-, 192-, and 256-bit keys, similar to AES. Camellia is described e.g. in IETF RFC 3713 ([6]).

3.4.1 Definitions

This section defines the key type "CKK_CAMELLIA" for type CK_KEY_TYPE as used in the CKA_KEY_TYPE attribute of key objects.

Mechanisms:

```
CKM_CAMELLIA_KEY_GEN
CKM_CAMELLIA_ECB
CKM_CAMELLIA_CBC
CKM_CAMELLIA_MAC
CKM_CAMELLIA_MAC_GENERAL
CKM_CAMELLIA_CBC_PAD
```

3.4.2 Camellia secret key objects

Camellia secret key objects (object class **CKO_SECRET_KEY,** key type **CKK_CAMELLIA**) hold Camellia keys. The following table defines the Camellia secret key object attributes, in addition to the common attributes defined for this object class:

Table 5, Camellia Secret Key Object Attributes

Attribute	Data type	Meaning
CKA_VALUE[1,4,6,7]	Byte array	Key value (16, 24, or 32 bytes)
CKA_VALUE_LEN[2,3,6]	CK_ULONG	Length in bytes of key value

- Refer to table 15 of [1] for footnotes.

The following is a sample template for creating a Camellia secret key object:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_CAMELLIA;
CK_UTF8CHAR label[] = "A Camellia secret key object";
CK_BYTE value[] = {...};
CK_BBOOL true = CK_TRUE;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
  {CKA_TOKEN, &true, sizeof(true)},
  {CKA_LABEL, label, sizeof(label)-1},
  {CKA_ENCRYPT, &true, sizeof(true)},
  {CKA_VALUE, value, sizeof(value)}
};
```

3.4.3 Camellia key generation

The Camellia key generation mechanism, denoted **CKM_CAMELLIA_KEY_GEN**, is a key generation mechanism for Camellia.

It does not have a parameter.

The mechanism generates Camellia keys with a particular length in bytes, as specified in the **CKA_VALUE_LEN** attribute of the template for the key.

The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_VALUE** attributes to the new key. Other attributes supported by the Camellia key type (specifically, the flags indicating which functions the key supports) may be specified in the template for the key, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of Camellia key sizes, in bytes.

3.4.4 Camellia-ECB

Camellia-ECB, denoted **CKM_CAMELLIA_ECB**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on Camellia and electronic codebook mode.

It does not have a parameter.

This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the **CKA_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size minus one null bytes so that the resulting length is a multiple of the block size. The output data is the same length as the padded input data. It does not wrap the key type, key length, or any other information about the key; the application must convey these separately.

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA_KEY_TYPE** attribute of the template and, if it has one, and the key type supports it, the **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE** attribute of the new key; other attributes required by the key type must be specified in the template.

Constraints on key types and the length of data are summarized in the following table:

Table 6, Camellia-ECB: Key And Data Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	CKK_CAMELLIA	multiple of block size	same as input length	no final part
C_Decrypt	CKK_CAMELLIA	multiple of block size	same as input length	no final part
C_WrapKey	CKK_CAMELLIA	any	input length rounded up to multiple of block size	
C_UnwrapKey	CKK_CAMELLIA	multiple of block size	determined by type of key being unwrapped or CKA_VALUE_LEN	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of Camellia key sizes, in bytes.

3.4.5 CAMELLIA-CBC

Camellia-CBC, denoted **CKM_CAMELLIA_CBC**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on Camellia and cipher-block chaining mode.

It has a parameter, a 16-byte initialization vector.

This mechanism can wrap and unwrap any secret key. Of course, a particular token may not be able to wrap/unwrap every secret key that it supports. For wrapping, the mechanism encrypts the value of the **CKA_VALUE** attribute of the key that is wrapped, padded on the trailing end with up to block size minus one null bytes so that the resulting length is a multiple of the block size. The output data is the same length as the padded input data. It does not wrap the key type, key length, or any other information about the key; the application must convey these separately.

For unwrapping, the mechanism decrypts the wrapped key, and truncates the result according to the **CKA_KEY_TYPE** attribute of the template and, if it has one, and the key type supports it, the **CKA_VALUE_LEN** attribute of the template. The mechanism contributes the result as the **CKA_VALUE** attribute of the new key; other attributes required by the key type must be specified in the template.

Constraints on key types and the length of data are summarized in the following table:

Table 7, Camellia-CBC: Key And Data Length

Function	Key type	Input length	Output length	Comments
C_Encrypt	CKK_CAMELLIA	multiple of block size	same as input length	no final part
C_Decrypt	CKK_CAMELLIA	multiple of block size	same as input length	no final part
C_WrapKey	CKK_CAMELLIA	any	input length rounded up to multiple of the block size	
C_UnwrapKey	CKK_CAMELLIA	multiple of block size	determined by type of key being unwrapped or CKA_VALUE_LEN	

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of Camellia key sizes, in bytes.

3.4.6 Camellia-CBC with PKCS padding

Camellia-CBC with PKCS padding, denoted **CKM_CAMELLIA_CBC_PAD**, is a mechanism for single- and multiple-part encryption and decryption; key wrapping; and key unwrapping, based on Camellia; cipher-block chaining mode; and the block cipher padding method detailed in PKCS #7.

It has a parameter, a 16-byte initialization vector.

The PKCS padding in this mechanism allows the length of the plaintext value to be recovered from the ciphertext value. Therefore, when unwrapping keys with this mechanism, no value should be specified for the **CKA_VALUE_LEN** attribute.

In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA, Diffie-Hellman, X9.42 Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section TBA for details). The entries in the table below for data length constraints when wrapping and unwrapping keys do not apply to wrapping and unwrapping private keys.

Constraints on key types and the length of data are summarized in the following table:

Table 8, Camellia-CBC with PKCS Padding: Key And Data Length

Function	Key type	Input length	Output length
C_Encrypt	CKK_CAMELLIA	any	input length rounded up to multiple of the block size
C_Decrypt	CKK_CAMELLIA	multiple of block size	between 1 and block size bytes shorter than input length
C_WrapKey	CKK_CAMELLIA	any	input length rounded up to multiple of the block size
C_UnwrapKey	CKK_CAMELLIA	multiple of block size	between 1 and block length bytes shorter than input length

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of Camellia key sizes, in bytes.

3.4.7 General-length Camellia-MAC

General-length Camellia -MAC, denoted **CKM_CAMELLIA_MAC_GENERAL**, is a mechanism for single- and multiple-part signatures and verification, based on Camellia and data authentication as defined in [5].

It has a parameter, a **CK_MAC_GENERAL_PARAMS** structure, which specifies the output length desired from the mechanism.

The output bytes from this mechanism are taken from the start of the final Camellia cipher block produced in the MACing process.

Constraints on key types and the length of data are summarized in the following table:

Table 9, General-length Camellia-MAC: Key And Data Length

Function	Key type	Data length	Signature length
C_Sign	CKK_CAMELLIA	any	0-block size, as specified in parameters
C_Verify	CKK_CAMELLIA	any	0-block size, as specified in parameters

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of Camellia key sizes, in bytes.

3.4.8 Camellia-MAC

Camellia-MAC, denoted by **CKM_CAMELLIA_MAC**, is a special case of the general-length Camellia-MAC mechanism. Camellia-MAC always produces and verifies MACs that are half the block size in length.

It does not have a parameter.

Constraints on key types and the length of data are summarized in the following table:

Table 10, Camellia-MAC: Key And Data Length

Function	Key type	Data length	Signature length
C_Sign	CKK_CAMELLIA	any	½ block size (8 bytes)
C_Verify	CKK_CAMELLIA	any	½ block size (8 bytes)

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the supported range of Camellia key sizes, in bytes.

3.5 Key derivation by data encryption - Camellia

These mechanisms allow derivation of keys using the result of an encryption operation as the key value. They are for use with the C_DeriveKey function.

Definitions

Mechanisms:

```
CKM_CAMELLIA_ECB_ENCRYPT_DATA
CKM_CAMELLIA_CBC_ENCRYPT_DATA

typedef struct CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS {
  CK_BYTE      iv[16];
  CK_BYTE_PTR  pData;
  CK_ULONG     length;
} CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS;
typedef CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS CK_PTR
CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS_PTR;
```

Mechanism Parameters

Uses CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS, and CK_KEY_DERIVATION_STRING_DATA as defined in section TBA

Table 11, Mechanism Parameters for Camellia-based key derivation

CKM_CAMELLIA_ECB_ENCRYPT_DATA	Uses CK_KEY_DERIVATION_STRING_DATA structure. Parameter is the data to be encrypted and must be a multiple of 16 long.
CKM_CAMELLIA_CBC_ENCRYPT_DATA	Uses CK_CAMELLIA_CBC_ENCRYPT_DATA_PARAMS. Parameter is an 16 byte IV value followed by the data. The data value part must be a multiple of 16 bytes long.

A. Manifest constants

The following definitions can be found in the appropriate header file.

```
#define CKM_SHA224                      0x00000255
#define CKM_SHA224_HMAC                 0x00000256
#define CKM_SHA224_HMAC_GENERAL         0x00000257
#define CKM_SHA224_RSA_PKCS             0x00000046
#define CKM_SHA224_RSA_PKCS_PSS         0x00000047
#define CKM_SHA224_KEY_DERIVATION       0x00000396
#define CKG_MGF1_SHA224                 0x00000005

#define CKM_AES_CTR                     0x00001086

#define CKK_CAMELLIA                    0x00000025
#define CKM_CAMELLIA_KEY_GEN            0x00000550
#define CKM_CAMELLIA_ECB                0x00000551
#define CKM_CAMELLIA_CBC                0x00000552
#define CKM_CAMELLIA_MAC                0x00000553
#define CKM_CAMELLIA_MAC_GENERAL        0x00000554
#define CKM_CAMELLIA_CBC_PAD            0x00000555
#define CKM_CAMELLIA_ECB_ENCRYPT_DATA   0x00000556
#define CKM_CAMELLIA_CBC_ENCRYPT_DATA   0x00000557
```

B. Intellectual property considerations

RSA makes no patent claims on the general constructions described in this document, although specific underlying techniques may be covered.

C. References

[1] RSA Laboratories. *PKCS #11: Cryptographic Token Interface Standard*. Version 2.20, June 2004. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20.pdf.

[2] Smit et al, "*A 224-bit One-way Hash Function: SHA-224*," IETF RFC 3874, June 2004. URL: http://ietf.org/rfc/rfc3874.txt.

[3] National Institute for Standards and Technology, "Recommendation for Block Cipher Modes of Operation," NIST SP 800-38A. URL: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf.

[4] Housley, "*Using Advanced Encryption Standard (AES) Counter Mode With IPsec Encapsulating Security Payload (ESP)*," IETF RFC 3686, January 2004. URL: http://ietf.org/rfc/rfc3686.txt.

[5] FIPS Publication 113, "Computer Data Authentication," U.S. DoC/NIST, May 1985. URL: http:// http://www.itl.nist.gov/fipspubs/fip113.htm.

[6] Matsui, et al, "A Description of the Camellia Encryption Algorithm," IETF RFC 3717, April 2004. URL: http://ietf.org/rfc/rfc3713.txt.

D. About PKCS

The *Public Key Cryptography Standards* are documents produced by RSA in cooperation with secure systems developers for the purpose of simplifying integration and management of accelerating the deployment of public-key cryptography and strong authentication technology into secure applications, and to enhance the user experience of these technologies.

RSA plans further development of the PKCS series through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. Results may also be submitted to standards forums. For more information, contact:

PKCS Editor
RSA, The Security Division of EMC
174 Middlesex Turnpike
Bedford, MA 01730 USA
pkcss-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/