PKCS #11 v2.10 Amendment 1: ECC

RSA Laboratories

DRAFT 2 — November 30, 2000

Editor's note: This is the third and final draft version of this amendment 1 to PKCS #11 v2.10 [1], which is available for a 30-day last call public review period. Please send comments and suggestions, both technical and editorial, to pkcs-editor@rsasecurity.com or cryptoki@rsasecurity.com.

Table of Contents

TABLE OF CONTENTS 1

1. INTRODUCTION 2

2. CHANGES TO SECTION 3, "REFERENCES" 2

3. CHANGES TO SECTION 4, "DEFINITIONS" 3

4. CHANGES TO SECTION 5, "SYMBOLS AND ABBREVIATIONS" 3

5. CHANGES TO SECTION 9, "GENERAL DATA TYPES" 3

5.1 CHANGES TO SECTION 9.4, "OBJECT TYPES" 3
5.2 CHANGES TO SECTION 9.5, "DATA TYPES FOR MECHANISMS" 3
5.3 CHANGES TO SECTION 9.6, "FUNCTION TYPES" 4

6. CHANGES TO SECTION 10, "OBJECTS" 4

6.1 CHANGES TO SECTION 10.8, "PUBLIC KEY OBJECTS" 4
6.2 CHANGES TO SECTION 10.8.3, "ECDSA PUBLIC KEY OBJECTS" 5
6.3 CHANGES TO SECTION 10.9, "PRIVATE KEY OBJECTS" 6
6.4 CHANGES TO SECTION 10.9.3, "ECDSA PRIVATE KEY OBJECTS" 6

7. CHANGES TO SECTION 11, "FUNCTIONS" 7

7.1 CHANGES TO SECTION 11.1.6, "ALL OTHER CRYPTOKI FUNCTION RETURN VALUES" 7
7.2 CHANGES TO SECTION 11.7, "OBJECT MANAGEMENT FUNCTIONS" 8
7.3 CHANGES TO SECTION 11.14, "KEY MANAGEMENT FUNCTIONS" 8

8. CHANGES TO SECTION 12, "MECHANISMS" 9

8.1 CHANGES TO SECTION 12.3, "ABOUT ECDSA" 10
8.2 CHANGES TO SECTION 12.4.1, "ECDSA KEY PAIR GENERATION" 11
8.3 CHANGES TO SECTION 12.4.2, "ECDSA WITHOUT HASHING" 12
8.4 CHANGES TO SECTION 12.4.3, "ECDSA WITH SHA-1" 13
8.5 NEW SECTIONS 12.4.5 THROUGH 12.4.8 13
8.6 CHANGES TO SECTION 12.9, "WRAPPING/UNWRAPPING PRIVATE KEYS (RSA, DIFFIE-HELLMAN, AND DSA)" 18
8.7 CHANGES TO SECTION 12.12.4, "RC2-CBC WITH PKCS PADDING" 20
8.8 CHANGES TO SECTION 12.16.4, "RC5-CBC WITH PKCS PADDING" 20
8.9 CHANGES TO SECTION 12.18.4, "GENERAL BLOCK CIPHER CBC WITH PKCS PADDING" 20

A. INTELLECTUAL PROPERTY CONSIDERATIONS 21

B. REFERENCES 21

C. ABOUT PKCS 21

1. Introduction

This document amends PKCS #11 v2.10 [1] to support elliptic curve cryptography as described in the ANSI X9.62 [2] standard and the ANSI X9.63 [3] draft developed by the ANSI X9F1 working group. This amendment generalizes the definition of public and private key objects and the mechanism for the generation of elliptic curve key pair. This amendment also expands the current definition of elliptic curve domain parameters to be more consistent with ANSI X9.62 [2] and ANSI X9.63 [3]. This amendment adds new EC based mechanisms

Version 2.01 of PKCS#11 [1] added the support for a variety of private keys to be wrapped and unwrapped with a secret key, however this function was not supported for elliptic curve private keys. This amendment adds the support for the wrapping and unwrapping of elliptic curve private keys in general.

This amendment is written as revisions to PKCS #11 v2.10 [1]. Only the affected sections are included.

Editor's note: To ease the review of this last draft version of this amendment, changes to updated sections of PKCS#11 are highlighted in gray. Note that all editor's notes and these highlights will be removed in the final version.

2. Changes to Section 3, "References"

[Update the reference as follows:]

ANSI X9.62 Accredited Standards Committee X9. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. 1999.

[Add the following new references:]

ANSI X9.63 Accredited Standards Committee X9. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*. Working draft, October 10, 2000.

SEC 1 Standards for Efficient Cryptography Group (SECG). *Standards for Efficient Cryptography (SEC) 1: Elliptic Curve Cryptography*. Version 1.0, September 20, 2000.

3. Changes to Section 4, "Definitions"

[Add the following new definitions:]

EC Elliptic Curve

ECDH Elliptic Curve Diffie-Hellman.

ECMQV **E**lliptic Curve Menezes-Qu-Vanstone

4. Changes to Section 5, "Symbols and abbreviations"

[Add the following prefix in Table 2:]

Prefix	Description
CKD_	Key derivation function

5. Changes to Section 9, "General data types"

5.1 Changes to Section 9.4, "Object types"

[Add the following key type to the paragraph on CK_KEY_TYPE:]

```
#define CKK_EC              0x00000003
```

Editor's note: The value for the CKK_EC key type is the same as the value for the CKK_ECDSA key type since according to ANSI X9.62 [2] and ANSI X9.63 [3], elliptic curve private and public keys have the same syntax regardless of their designated use.

[Add the following attribute type to the paragraph on CK_ATTRIBUTE_TYPE:]

```
#define CKA_EC_PARAMS           0x00000180
```

Editor's note: The value for the CKA_EC_PARAMS attribute is the same as the value for the CKA_ECDSA_PARAMS attribute since according to ANSI X9.62 [2] and ANSI X9.63 [3], elliptic curve domain parameters have the same syntax.

5.2 Changes to Section 9.5, "Data types for mechanisms"

[Add the following mechanism types to the paragraph on CK_MECHANISM_TYPE:]

```
#define CKM_EC_KEY_PAIR_GEN             0x00001040
#define CKM_ECDH1_DERIVE                0x00001045
#define CKM_ECDH1_COFACTOR_DERIVE       0x00001046
```

```
#define CKM_ECMQV_DERIVE                0x00001047
```

Editor's note: The value for the CKM_EC_KEY_PAIR_GEN mechanism is the same as the value for the CKM_ECDSA_KEY_PAIR_GEN mechanism since according to ANSI X9.62 [2] and ANSI X9.63 [3], the generation of elliptic curve keys is the same regardless of their designated use. Other EC mechanisms could also be added once it is determined which other one of the 21 schemes defined under ANSI X9.63 [3] should also be defined under Cryptoki, especially the key transport schemes, since none are currently defined in this amendment.

[Add the following mechanism information flags to Table 12:]

Bit Flag	Mask	Meaning
CKF_EC_F_P	0x00100000	TRUE if the mechanism can be used with EC domain parameters over F_p
CKF_EC_F_2M	0x00200000	TRUE if the mechanism can be used with EC domain parameters over F_{2^m}
CKF_EC_NAMEDCURVE	0x00400000	TRUE if the mechanism can be used with EC domain parameters of the choice **namedCurve**; FALSE if the mechanism can only be used with EC domain parameters of the choice **ecParameters**
CKF_EC_COMPRESS	0x00800000	TRUE if the mechanism can be used with elliptic curve point compression

5.3 Changes to Section 9.6, "Function types"

[Add the following return value to the paragraph on CK_RV:]

```
#define CKR_KEY_PARAMS_INVALID          0x0000006B
```

6. Changes to Section 10, "Objects"

6.1 Changes to Section 10.8, "Public key objects"

[Replace the second sentence with the following:]

This version of Cryptoki recognizes the following types of public keys: RSA, DSA, EC (also related to ECDSA), Diffie-Hellman, and KEA.

6.2 Changes to Section 10.8.3, "ECDSA public key objects"

[Replace Section 10.8.3 with the following:]

10.8.3 Elliptic curve public key objects

EC (also related to ECDSA) public key objects (object class **CKO_PUBLIC_KEY,** key type **CKK_EC** or **CKK_ECDSA**) hold EC public keys. See Section 12.3 for more information about EC. The following table defines the EC public key object attributes, in addition to the common attributes listed in Table 14, Table 18, Table 24, and Table 25:

Table 29, Elliptic Curve Public Key Object Attributes

Attribute	Data type	Meaning
CKA_EC_PARAMS[1,3,6] (CKA_ECDSA_PARAMS)	Byte array	DER-encoding of an ANSI X9.62 `Parameters` value
CKA_EC_POINT[1,4,6]	Byte array	DER-encoding of ANSI X9.62 `ECPoint` value *Q*

The **CKA_EC_PARAMS** or **CKA_ECDSA_PARAMS** attribute value is known as the "EC domain parameters" and is defined in ANSI X9.62 as a choice of three parameter representation methods with the following syntax:

```
Parameters ::= CHOICE {
  ecParameters   ECParameters,
  namedCurve     CURVES.&id({CurveNames}),
  implicitlyCA   NULL
}
```

This allows detailed specification of all required values using choice **ecParameters**, the use of a **namedCurve** as an object identifier substitute for a particular set of elliptic curve domain parameters, or **implicitlyCA** to indicate that the domain parameters are explicitly defined elsewhere. The use of a **namedCurve** is recommended over the choice **ecParameters**. The choice **implicitlyCA** must not be used in Cryptoki.

The following is a sample template for creating an EC (ECDSA) public key object:

```
CK_OBJECT_CLASS class = CKO_PUBLIC_KEY;
CK_KEY_TYPE keyType = CKK_EC;
CK_UTF8CHAR label[] = "An EC public key object";
CK_BYTE ecParams[] = {...};
CK_BYTE ecPoint[] = {...};
CK_BBOOL true = 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_EC_PARAMS, ecParams, sizeof(ecParams)},
    {CKA_EC_POINT, ecPoint, sizeof(ecPoint)}
};
```

6.3 Changes to Section 10.9, "Private key objects"

[Replace the second sentence with the following:]

This version of Cryptoki recognizes the following types of private key: RSA, DSA, EC (also related to ECDSA), Diffie-Hellman, and KEA.

6.4 Changes to Section 10.9.3, "ECDSA private key objects"

[Replace Section 10.9.3 with the following:]

10.9.3 Elliptic curve private key objects

EC (also related to ECDSA) private key objects (object class **CKO_PRIVATE_KEY,** key type **CKK_EC** or **CKK_ECDSA**) hold EC private keys. See Section 12.3 for more information about EC. The following table defines the EC private key object attributes, in addition to the common attributes listed in Table 14, Table 18, Table 24, and Table 32:

Table 36, Elliptic Curve Private Key Object Attributes

Attribute	Data type	Meaning
CKA_EC_PARAMS[1,4,6] (CKA_ECDSA_PARAMS)	Byte array	DER-encoding of an ANSI X9.62 `Parameters` value
CKA_VALUE[1,4,6,7]	Big integer	ANSI X9.62 private value *d*

The **CKA_EC_PARAMS** or **CKA_ECDSA_PARAMS** attribute value is known as the "EC domain parameters" and is defined in ANSI X9.62 as a choice of three parameter representation methods with the following syntax:

```
Parameters ::= CHOICE {
  ecParameters   ECParameters,
  namedCurve     CURVES.&id({CurveNames}),
  implicitlyCA   NULL
}
```

This allows detailed specification of all required values using choice **ecParameters**, the use of a **namedCurve** as an object identifier substitute for a particular set of elliptic

curve domain parameters, or **implicitlyCA** to indicate that the domain parameters are explicitly defined elsewhere. The use of a **namedCurve** is recommended over the choice **ecParameters**. The choice **implicitlyCA** must not be used in Cryptoki.

Note that when generating an EC private key, the EC domain parameters are *not* specified in the key's template. This is because EC private keys are only generated as part of an EC key *pair*, and the EC domain parameters for the pair are specified in the template for the EC public key.

The following is a sample template for creating an EC (ECDSA) private key object:

```
CK_OBJECT_CLASS class = CKO_PRIVATE_KEY;
CK_KEY_TYPE keyType = CKK_EC;
CK_UTF8CHAR label[] = "An EC private key object";
CK_BYTE subject[] = {...};
CK_BYTE id[] = {123};
CK_BYTE ecParams[] = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = 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_SUBJECT, subject, sizeof(subject)},
  {CKA_ID, id, sizeof(id)},
  {CKA_SENSITIVE, &true, sizeof(true)},
  {CKA_DERIVE, &true, sizeof(true)},
  {CKA_EC_PARAMS, ecParams, sizeof(ecParams)},
  {CKA_VALUE, value, sizeof(value)}
};
```

7. Changes to Section 11, "Functions"

7.1 Changes to Section 11.1.6, "All other Cryptoki function return values"

[Add the following new return value:]

- CKR_KEY_PARAMS_INVALID: Invalid or unsupported domain parameters were supplied to the function. Which representation methods of domain parameters are supported by a given mechanism can vary from token to token.

7.2 Changes to Section 11.7, "Object management functions"

[Replace the return values for the C_CreateObject function with:]

Return values: CKR_ATTRIBUTE_READ_ONLY,
CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID,
CKR_CRYPTOKI_NOT_INITIALIZED, CKR_DEVICE_ERROR,
CKR_DEVICE_MEMORY, CKR_DEVICE_REMOVED, CKR_FUNCTION_FAILED,
CKR_GENERAL_ERROR, CKR_HOST_MEMORY,
CKR_KEY_PARAMS_INVALID, CKR_OK, CKR_SESSION_CLOSED,
CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY,
CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT,
CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN,
CKR_ARGUMENTS_BAD.

7.3 Changes to Section 11.14, "Key management functions"

[Replace the return values for the C_GenerateKeyPair function with:]

Return values: CKR_ATTRIBUTE_READ_ONLY,
CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID,
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_PARAMS_INVALID, CKR_MECHANISM_INVALID,
CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE,
CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID,
CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE,
CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED,
CKR_USER_NOT_LOGGED_IN, CKR_ARGUMENTS_BAD.

[Replace the fourth bullet under the C_WrapKey function with:]

- To wrap an RSA, Diffie-Hellman, EC (also related to ECDSA) or DSA private key with any secret key other than a SKIPJACK, BATON, or JUNIPER key.

[Replace the return values for the C_UnwrapKey function with:]

Return values: CKR_ATTRIBUTE_READ_ONLY,
CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID,
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_PARAMS_INVALID, CKR_MECHANISM_INVALID,
CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE,

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, CKR_ARGUMENTS_BAD.

[Replace the return values for the C_DeriveKey function with:]

Return values: CKR_ATTRIBUTE_READ_ONLY, CKR_ATTRIBUTE_TYPE_INVALID, CKR_ATTRIBUTE_VALUE_INVALID, 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_PARAMS_INVALID, CKR_KEY_SIZE_RANGE, CKR_KEY_TYPE_INCONSISTENT, CKR_MECHANISM_INVALID, CKR_MECHANISM_PARAM_INVALID, CKR_OK, CKR_OPERATION_ACTIVE, CKR_SESSION_CLOSED, CKR_SESSION_HANDLE_INVALID, CKR_SESSION_READ_ONLY, CKR_TEMPLATE_INCOMPLETE, CKR_TEMPLATE_INCONSISTENT, CKR_TOKEN_WRITE_PROTECTED, CKR_USER_NOT_LOGGED_IN, CKR_ARGUMENTS_BAD.

8. Changes to Section 12, "Mechanisms"

[Replace the line about CKM_ECDSA_KRY_PAIR_GEN in Table 55 with:]

Mechanism	Functions						
	Encrypt & Decrypt	**Sign & Verify**	**SR & VR**[1]	**Digest**	**Gen. Key/ Key Pair**	**Wrap & Unwrap**	**Derive**
CKM_EC_KEY_PAIR_GEN (CKM_ECDSA_KEY_PAIR_GEN)					✓		

[Add the following new mechanisms in Table 55:]

Mechanism	Functions						
	Encrypt & Decrypt	**Sign & Verify**	**SR & VR**[1]	**Digest**	**Gen. Key/ Key Pair**	**Wrap & Unwrap**	**Derive**
CKM_ECDH1_DERIVE							✓
CKM_ECDH1_COFACTOR_DERIVE							✓
CKM_ECMQV_DERIVE							✓

Editor's note: Other EC mechanisms could also be added once it is determined which other one of the 21 schemes defined under ANSI X9.63 [3] should also be defined under Cryptoki, especially the key transport schemes, since none are currently defined in this amendment.

8.1 Changes to Section 12.3, "About ECDSA"

[Replace Section 12.3 with the following:]

12.3 About elliptic curve

The EC cryptosystem (also related to ECDSA) in this document is the one described in the ANSI X9.62 standard and the ANSI X9.63 draft developed by the ANSI X9F1 working group.

In these standards, there are two different varieties of EC defined:

1. EC using a field with an odd prime number of elements (i.e. the finite field F_p).

2. EC using a field of characteristic two (i.e. the finite field F_{2^m}).

An EC key in Cryptoki contains information about which variety of EC it is suited for. It is preferable that a Cryptoki library, which can perform EC mechanisms, be capable of performing operations with the two varieties of EC, however this is not required. The **CK_MECHANISM_INFO** structure **CKF_EC_F_P** flag identifies a Cryptoki library supporting EC keys over F_p whereas the **CKF_EC_F_2M** flag identifies a Cryptoki library supporting EC keys over F_{2^m}.

In these specifications there are also three representation methods to define the domain parameters for an EC key. Only the **ecParameters** and the **namedCurve** choices are supported in Cryptoki. The **CK_MECHANISM_INFO** structure **CKF_EC_NAMEDCURVE** flag identifies a Cryptoki library supporting the **namedCurve** choice.

In these specifications, an EC public key (i.e. EC point Q) or the base point G when the **ecParameters** choice is used can be represented as an octet string of the compressed form or the uncompressed form. The **CK_MECHANISM_INFO** structure **CKF_EC_COMPRESS** flag identifies a Cryptoki library supporting the compress form.

If an attempt to create, generate, derive, or unwrap an EC key of an unsupported variety (or of an unsupported size of a supported variety) is made, that attempt should fail with the error code CKR_TEMPLATE_INCONSISTENT. If an attempt to create, generate, derive, or unwrap an EC key with invalid or of an unsupported representation of domain parameters is made, that attempt should fail with the error code CKR_KEY_PARAMS_INVALID. If an attempt to create, generate, derive, or unwrap an EC key of an unsupported form is made, that attempt should fail with the error code CKR_TEMPLATE_INCONSISTENT.

8.2 Changes to Section 12.4.1, "ECDSA key pair generation"

[Replace Section 12.4.1 with the following:]

12.4.1 Elliptic curve key pair generation

The EC (also related to ECDSA) key pair generation mechanism, denoted **CKM_EC_KEY_PAIR_GEN** or **CKM_ECDSA_KEY_PAIR_GEN**, is a key pair generation mechanism for EC.

This mechanism does not have a parameter.

The mechanism generates EC public/private key pairs with particular EC domain parameters, as specified in the **CKA_EC_PARAMS** or **CKA_ECDSA_PARAMS** attribute of the template for the public key. Note that this version of Cryptoki does not include a mechanism for generating these EC domain parameters.

The mechanism contributes the **CKA_CLASS**, **CKA_KEY_TYPE**, and **CKA_EC_POINT** attributes to the new public key and the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_EC_PARAMS** or **CKA_ECDSA_PARAMS** and **CKA_CKA_VALUE** attributes to the new private key. Other attributes supported by the EC public and private key types (specifically, the flags indicating which functions the keys support) may also be specified in the templates for the keys, or else are assigned default initial values.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the minimum and maximum supported number of bits in the field sizes, respectively. For example, if a Cryptoki library supports only ECDSA using a field of characteristic 2 which has between 2^{200} and 2^{300} elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly, 2^{300} is a 301-bit number).

8.3 Changes to Section 12.4.2, "ECDSA without hashing"

[Replace Section 12.4.2 with the following:]

12.4.2 ECDSA without hashing

The ECDSA without hashing mechanism, denoted **CKM_ECDSA**, is a mechanism for single-part signatures and verification for ECDSA. (This mechanism corresponds only to the part of ECDSA that processes the 20-byte hash value; it does not compute the hash value.)

For the purposes of this mechanism, an ECDSA signature is an octet string of length two times *nLen*, where *nLen* is the length in octets of the base point order *n*, and corresponds to the concatenation of the ECDSA values *r* and *s*, each represented as an octet string of length *nLen* most-significant byte first.

This mechanism does not have a parameter.

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

Table 66, ECDSA: Key And Data Length

Function	Key type	Input length	Output length
C_Sign[1]	ECDSA private key	20	2*nLen*
C_Verify[1]	ECDSA public key	20, 2*nLen* [2]	N/A

[1] Single-part operations only.

[2] Data length, signature length.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the minimum and maximum supported number of bits in the field sizes, respectively. For example, if a Cryptoki library supports only ECDSA using a field of characteristic 2 which has between 2^{200} and 2^{300} elements (inclusive), then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly, 2^{300} is a 301-bit number).

8.4 Changes to Section 12.4.3, "ECDSA with SHA-1"

[Replace Section 12.4.3 with the following:]

12.4.3 ECDSA with SHA-1

The ECDSA with SHA-1 mechanism, denoted **CKM_ECDSA_SHA1**, is a mechanism for single- and multiple-part signatures and verification for ECDSA. This mechanism computes the entire ECDSA specification, including the hashing with SHA-1.

For the purposes of this mechanism, an ECDSA signature is an octet string of length two times *nLen*, where *nLen* is the length in octets of the base point order *n*, and corresponds to the concatenation of the ECDSA values *r* and *s*, each represented as an octet string of length *nLen* most-significant byte first.

This mechanism does not have a parameter.

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

Table 67, ECDSA with SHA-1: Key And Data Length

Function	Key type	Input length	Output length
C_Sign	ECDSA private key	any	2*nLen*
C_Verify	ECDSA public key	any, 2*nLen* [2]	N/A

[2] Data length, signature length.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the minimum and maximum supported number of bits in the field sizes, respectively. For example, if a Cryptoki library supports only ECDSA using a field of characteristic 2 which has between 2^{200} and 2^{300} elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly, 2^{300} is a 301-bit number).

8.5 New Sections 12.4.5 through 12.4.8

[Insert new sections 12.4.5 through 12.4.8:]

12.4.5 EC mechanism parameters

♦ **CK_EC_KDF_TYPE, CK_EC_KDF_TYPE_PTR**

CK_EC_KDF_TYPE is used to indicate the Key Derivation Function (KDF) applied to derive keying data from a shared secret. The key derivation function will be used by the EC key agreement schemes. It is defined as follows:

```
typedef CK_ULONG CK_EC_KDF_TYPE;
```

The following table lists the defined functions.

Table ??, EC: Key Derivation Functions

Source Identifier	Value
CKD_NULL	0x00000001
CKD_ SHA1_KDF	0x00000002

The key derivation function **CKD_NULL** produces a raw shared secret value without applying any key derivation function whereas the key derivation function **CKD_SHA1_KDF**, which is based on SHA-1, derives keying data from the shared secret value as defined in the ANSI X9.63 draft.

CK_EC_KDF_TYPE_PTR is a pointer to a **CK_EC_KDF_TYPE**.

- **CK_ECDH1_DERIVE_PARAMS, CK_ECDH1_DERIVE_PARAMS_PTR**

CK_ECDH1_DERIVE_PARAMS is a structure that provides the parameters for the **CKM_ECDH1_DERIVE** and **CKM_ECDH1_COFACTOR_DERIVE** key derivation mechanisms, where each party contributes one key pair. The structure is defined as follows:

```
typedef struct CK_ECDH1_DERIVE_PARAMS {
   CK_EC_KDF_TYPE kdf;
   CK_ULONG ulSharedDataLen;
   CK_BYTE_PTR pSharedData;
   CK_ULONG ulPublicDataLen;
   CK_BYTE_PTR pPublicData;
} CK_ECDH1_DERIVE_PARAMS;
```

The fields of the structure have the following meanings:

kdf	key derivation function used on the shared secret value
ulSharedDataLen	the length in bytes of the shared info
pSharedData	some data shared between the two parties
ulPublicDataLen	the length in bytes of the other party's EC public key
pPublicData	pointer to other party's EC public key value

With the key derivation function **CKD_NULL**, *pSharedData* must be NULL and *ulSharedDataLen* must be zero. With the key derivation function **CKD_SHA1_KDF**, an optional *pSharedData* may be supplied, which consists of some data shared by the two

parties intending to share the shared secret. Otherwise, *pSharedData* must be NULL and *ulSharedDataLen* must be zero.

CK_ECDH1_DERIVE_PARAMS_PTR is a pointer to a **CK_ECDH1_DERIVE_PARAMS**.

♦ **CK_ECDH2_DERIVE_PARAMS, CK_ECDH2_DERIVE_PARAMS_PTR**

CK_ECDH2_DERIVE_PARAMS is a structure that provides the parameters to the **CKM_ECMQV_DERIVE** key derivation mechanism, where each party contributes two key pairs. The structure is defined as follows:

```
typedef struct CK_ECDH2_DERIVE_PARAMS {
   CK_EC_KDF_TYPE kdf;
   CK_ULONG ulSharedDataLen;
   CK_BYTE_PTR pSharedData;
   CK_ULONG ulPublicDataLen;
   CK_BYTE_PTR pPublicData;
   CK_ULONG ulPrivateDataLen;
   CK_OBJECT_HANDLE hPrivateData;
   CK_ULONG ulPublicDataLen2;
   CK_BYTE_PTR pPublicData2;
} CK_ECDH2_DERIVE_PARAMS;
```

The fields of the structure have the following meanings:

kdf	key derivation function used on the shared secret value
ulSharedDataLen	the length in bytes of the shared info
pSharedData	some data shared between the two parties
ulPublicDataLen	the length in bytes of the other party's first EC public key
pPublicData	pointer to other party's first EC public key value
ulPrivateDataLen	the length in bytes of the second EC private key
hPrivateData	key handle for second EC private key value
ulPublicDataLen2	the length in bytes of the other party's second EC public key
pPublicData2	pointer to other party's second EC public key value

With the key derivation function **CKD_NULL**, *pSharedData* must be NULL and *ulSharedDataLen* must be zero. With the key derivation function **CKD_SHA1_KDF**, an

optional *pSharedData* may be supplied, which consists of some data shared by the two parties intending to share the shared secret. Otherwise, *pSharedData* must be NULL and *ulSharedDataLen* must be zero.

CK_ECDH2_DERIVE_PARAMS_PTR is a pointer to a **CK_ECDH2_DERIVE_PARAMS**.

12.4.6 Elliptic curve Diffie-Hellman key derivation

The elliptic curve Diffie-Hellman (ECDH) key derivation mechanism, denoted **CKM_ECDH1_DERIVE**, is a mechanism for key derivation based on the Diffie-Hellman version of the elliptic curve key agreement scheme, as defined in the ANSI X9.63 draft, where each party contributes one key pair all using the same EC domain parameters.

It has a parameter, a **CK_ECDH1_DERIVE_PARAMS** structure.

This mechanism derives a secret value, 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 truncation removes bytes from the leading end of the secret value.) 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.

The derived key inherits the values of the **CKA_SENSITIVE**, **CKA_ALWAYS_SENSITIVE**, **CKA_EXTRACTABLE**, and **CKA_NEVER_EXTRACTABLE** attributes from the base key. The values of the **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes may be overridden in the template for the derived key, however. Of course, if the base key has the **CKA_ALWAYS_SENSITIVE** attribute set to TRUE, then the template may not specify that the derived key should have the **CKA_SENSITIVE** attribute set to FALSE; similarly, if the base key has the **CKA_NEVER_EXTRACTABLE** attribute set to TRUE, then the template may not specify that the derived key should have the **CKA_EXTRACTABLE** attribute set to TRUE.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the minimum and maximum supported number of bits in the field sizes, respectively. For example, if a Cryptoki library supports only EC using a field of characteristic 2 which has between 2^{200} and 2^{300} elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly, 2^{300} is a 301-bit number).

12.4.7 Elliptic curve Diffie-Hellman with cofactor key derivation

The elliptic curve Diffie-Hellman (ECDH) with cofactor key derivation mechanism, denoted **CKM_ECDH1_COFACTOR_DERIVE**, is a mechanism for key derivation based on the cofactor Diffie-Hellman version of the elliptic curve key agreement scheme,

as defined in the ANSI X9.63 draft, where each party contributes one key pair all using the same EC domain parameters. Cofactor multiplication is computationally efficient and helps to prevent security problems like small group attacks.

It has a parameter, a **CK_ECDH1_DERIVE_PARAMS** structure.

This mechanism derives a secret value, 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 truncation removes bytes from the leading end of the secret value.) 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.

The derived key inherits the values of the **CKA_SENSITIVE**, **CKA_ALWAYS_SENSITIVE**, **CKA_EXTRACTABLE**, and **CKA_NEVER_EXTRACTABLE** attributes from the base key. The values of the **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes may be overridden in the template for the derived key, however. Of course, if the base key has the **CKA_ALWAYS_SENSITIVE** attribute set to TRUE, then the template may not specify that the derived key should have the **CKA_SENSITIVE** attribute set to FALSE; similarly, if the base key has the **CKA_NEVER_EXTRACTABLE** attribute set to TRUE, then the template may not specify that the derived key should have the **CKA_EXTRACTABLE** attribute set to TRUE.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the minimum and maximum supported number of bits in the field sizes, respectively. For example, if a Cryptoki library supports only EC using a field of characteristic 2 which has between 2^{200} and 2^{300} elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly, 2^{300} is a 301-bit number).

12.4.8 Elliptic curve Menezes-Qu-Vanstone key derivation

The elliptic curve Menezes-Qu-Vanstone (ECMQV) key derivation mechanism, denoted **CKM_ECMQV_DERIVE**, is a mechanism for key derivation based the MQV version of the elliptic curve key agreement scheme, as defined in the ANSI X9.63 draft, where each party contributes two key pairs all using the same EC domain parameters.

It has a parameter, a **CK_ECDH2_DERIVE_PARAMS** structure.

This mechanism derives a secret value, 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 truncation removes bytes from the leading end of the secret value.) 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.

The derived key inherits the values of the **CKA_SENSITIVE**, **CKA_ALWAYS_SENSITIVE**, **CKA_EXTRACTABLE**, and **CKA_NEVER_EXTRACTABLE** attributes from the base key. The values of the **CKA_SENSITIVE** and **CKA_EXTRACTABLE** attributes may be overridden in the template for the derived key, however. Of course, if the base key has the **CKA_ALWAYS_SENSITIVE** attribute set to TRUE, then the template may not specify that the derived key should have the **CKA_SENSITIVE** attribute set to FALSE; similarly, if the base key has the **CKA_NEVER_EXTRACTABLE** attribute set to TRUE, then the template may not specify that the derived key should have the **CKA_EXTRACTABLE** attribute set to TRUE.

For this mechanism, the *ulMinKeySize* and *ulMaxKeySize* fields of the **CK_MECHANISM_INFO** structure specify the minimum and maximum supported number of bits in the field sizes, respectively. For example, if a Cryptoki library supports only EC using a field of characteristic 2 which has between 2^{200} and 2^{300} elements, then *ulMinKeySize* = 201 and *ulMaxKeySize* = 301 (when written in binary notation, the number 2^{200} consists of a 1 bit followed by 200 0 bits. It is therefore a 201-bit number. Similarly, 2^{300} is a 301-bit number).

Editor's note: Other EC mechanisms could also be added once it is determined which other one of the 21 schemes defined under ANSI X9.63 [3] should also be defined under Cryptoki, especially the key transport schemes, since none are currently defined in this amendment.

8.6 Changes to Section 12.9, "Wrapping/unwrapping private keys (RSA, Diffie-Hellman, and DSA)"

[Replace the first few paragraphs of Section 12.9 until the bullets with:]

12.9 Wrapping/unwrapping private keys

Cryptoki Versions 2.01 and up allow the use of secret keys for wrapping and unwrapping RSA private keys, Diffie-Hellman private keys, EC (also related to ECDSA) private keys and DSA private keys. For wrapping, a private key is BER-encoded according to PKCS #8's PrivateKeyInfo ASN.1 type. PKCS #8 requires an algorithm identifier for the type of the secret key. The object identifiers for the required algorithm identifiers are as follows:

```
rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1 }

dhKeyAgreement OBJECT IDENTIFIER ::= { pkcs-3 1 }

id-ecPublicKey OBJECT IDENTIFIER ::= { iso(1) member-
  body(2) us(840) ansi-x9-62(10045) publicKeyType(2) 1 }

id-dsa OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) us(840) x9-57(10040) x9cm(4) 1 }
```

where

```
pkcs-1 OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 1
        }

pkcs-3 OBJECT IDENTIFIER ::= {
  iso(1) member-body(2) US(840) rsadsi(113549) pkcs(1) 3
        }
```

These parameters for the algorithm identifiers have the following types, respectively:

```
NULL

DHParameter ::= SEQUENCE {
  prime               INTEGER,  -- p
  base                INTEGER,  -- g
  privateValueLength  INTEGER OPTIONAL
}

Parameters ::= CHOICE {
  ecParameters   ECParameters,
  namedCurve     CURVES.&id({CurveNames}),
  implicitlyCA   NULL
}

Dss-Parms ::= SEQUENCE {
  p INTEGER,
  q INTEGER,
  g INTEGER
}
```

For the EC domain parameters, the use of **namedCurve** is recommended over the choice **ecParameters**. The choice **implicitlyCA** must not be used in Cryptoki.

Within the PrivateKeyInfo type:

- RSA private keys are BER-encoded according to PKCS #1's RSAPrivateKey ASN.1 type. This type requires values to be present for *all* the attributes specific to Cryptoki's RSA private key objects. In other words, if a Cryptoki library does not have values for an RSA private key's **CKA_MODULUS**, **CKA_PUBLIC_EXPONENT**, **CKA_PRIVATE_EXPONENT**, **CKA_PRIME_1**, **CKA_PRIME_2**, **CKA_EXPONENT_1**, **CKA_EXPONENT2**, and **CKA_COEFFICIENT** values, it cannot create an RSAPrivateKey BER-encoding of the key, and so it cannot prepare it for wrapping.

- Diffie-Hellman private keys are represented as BER-encoded ASN.1 type INTEGER.

- EC (also related with ECDSA) private keys are BER-encoded according to SECG SEC 1 ECPrivateKey ASN.1 type:

```
ECPrivateKey ::= SEQUENCE {
   Version       INTEGER { ecPrivkeyVer1(1) }
                 (ecPrivkeyVer1),
   privateKey    OCTET STRING,
   parameters    [0] Parameters OPTIONAL,
   publicKey     [1] BIT STRING OPTIONAL
}
```

  Since the EC domain parameters are placed in the PKCS #8's privateKeyAlgorithm field, the optional **parameters** field in an ECPrivateKey must be omitted. The optional **publicKey** field could be omitted, however it may be useful to send the public key along with the private key, especially in mechanisms that involve calculations with the public key.

- DSA private keys are represented as BER-encoded ASN.1 type INTEGER.

8.7 Changes to Section 12.12.4, "RC2-CBC with PKCS padding"

[Replace the first sentence of the fourth paragraph with:]

In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA, Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section 12.9 for details).

8.8 Changes to Section 12.16.4, "RC5-CBC with PKCS padding"

[Replace the first sentence of the fourth paragraph with:]

In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA, Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section 12.9 for details).

8.9 Changes to Section 12.18.4, "General block cipher CBC with PKCS padding"

[Replace the first sentence of the fourth paragraph with:]

In addition to being able to wrap and unwrap secret keys, this mechanism can wrap and unwrap RSA, Diffie-Hellman, EC (also related to ECDSA) and DSA private keys (see Section 12.9 for details).

A. Intellectual property considerations

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

License to copy this document is granted provided that it is identified as "RSA Security Inc. Public-Key Cryptography Standards (PKCS)" in all material mentioning or referencing this document.

RSA Security Inc. makes no other representations regarding intellectual property claims by other parties. Such determination is the responsibility of the user.

B. References

[1] RSA Laboratories. PKCS #11: *Cryptographic Token Interface Standard*. Version 2.10, December 1999.

[2] ANSI X9.62. Accredited Standards Committee X9. *Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)*. 1999.

[3] ANSI X9.63. Accredited Standards Committee X9. *Public Key Cryptography for the Financial Services Industry: Key Agreement and Key Transport Using Elliptic Curve Cryptography*. Working draft, October 10, 2000

C. About PKCS

The *Public-Key Cryptography Standards* are specifications produced by RSA Laboratories in cooperation with secure systems developers worldwide for the purpose of accelerating the deployment of public-key cryptography. First published in 1991 as a result of meetings with a small group of early adopters of public-key technology, the PKCS documents have become widely referenced and implemented. Contributions from the PKCS series have become part of many formal and *de facto* standards, including ANSI X9 documents, PKIX, SET, S/MIME, and SSL.

Further development of PKCS occurs through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. For more information, contact:

PKCS Editor
RSA Laboratories
20 Crosby Drive
Bedford, MA 01730 USA
pkcs-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/pkcs