PKCS #11 v2.20 Amendment 2

PKCS #11 Mechanisms for the Cryptographic Token Key Initialization Protocol

RSA Laboratories

December 27, 2005

TABLE OF CONTENTS

1 INTRODUCTION 2
1.1 SCOPE 2
1.2 BACKGROUND 2
1.3 DOCUMENT ORGANIZATION 2
2 ACRONYMS AND NOTATION 2
2.1 ACRONYMS 2
2.2 NOTATION 3
3 PRINCIPLES OF OPERATION 3
4 MECHANISMS 4
4.1 CT-KIP 4
4.1.1 Definitions 4
4.1.2 CT-KIP Mechanism parameters 4
♦ *CK_KIP_ PARAMS; CK_KIP_ PARAMS_PTR 4*
4.1.3 CT-KIP key derivation 5
4.1.4 CT-KIP key wrap and key unwrap 5
4.1.5 CT-KIP signature generation 5
A. MANIFEST CONSTANTS 7
A.1 NOTICE REGARDING HEADER FILES 7
A.2 MECHANISMS 7
B. USING PKCS #11 WITH CT-KIP 7
C. INTELLECTUAL PROPERTY CONSIDERATIONS 10
D. REFERENCES 10
E. ABOUT OTPS 10

1 Introduction

1.1 Scope

This document is an amendment to PKCS #11 v2.20 [1] and describes extensions to PKCS #11 to support the Cryptographic Token Key Initialization Protocol described in [2].

The mechanisms defined herein are intended for general use within computer and communications systems employing connected cryptographic tokens (or software emulations thereof).

1.2 Background

A cryptographic token may be a handheld hardware device, a hardware device connected to a personal computer through an electronic interface such as USB, or a software module resident on a personal computer, which offers some cryptographic functionality that may be used e.g., to authenticate a user towards some service. Increasingly, these tokens work in a connected fashion, enabling their programmatic initialization as well as programmatic retrieval of their output values. This document intends to meet the need for an open and interoperable mechanism to programmatically initialize and configure connected cryptographic tokens with a secret key shared by an external party. A companion document entitled "Cryptographic Token Key Initialization Protocol" [2] describes the protocol that is intended for use with the mechanisms defined here.

1.3 Document organization

The organization of this document is as follows:

- Section 1 is an introduction.
- Section 2 defines acronyms and notation used in this document.
- Section 3 describes the operational principles for the key initialization.
- Section 4 defines the mechanisms in detail.
- Appendix A collects the PKCS #11 constants defined herein.
- Appendix B describes how the mechanisms defined in this document may be used during a CT-KIP protocol run.
- Appendices C, D, and E cover intellectual property issues, give references to other publications and standards, and provide general information about the One-Time Password Specifications.

2 Acronyms and notation

2.1 Acronyms

CT-KIP	Cryptographic Token Key Initialization Protocol (as defined in [2])
MAC	Message Authentication Code

PDU Protocol Data Unit

2.2 Notation

C structure declarations are made in the `Courier` typeface. PKCS #11 functions and structure names are written in **boldface**. Function parameter names and structure components are written in *italic*. XML elements are written in brackets and bold Helvetica: **<element>**.

3 Principles of Operation

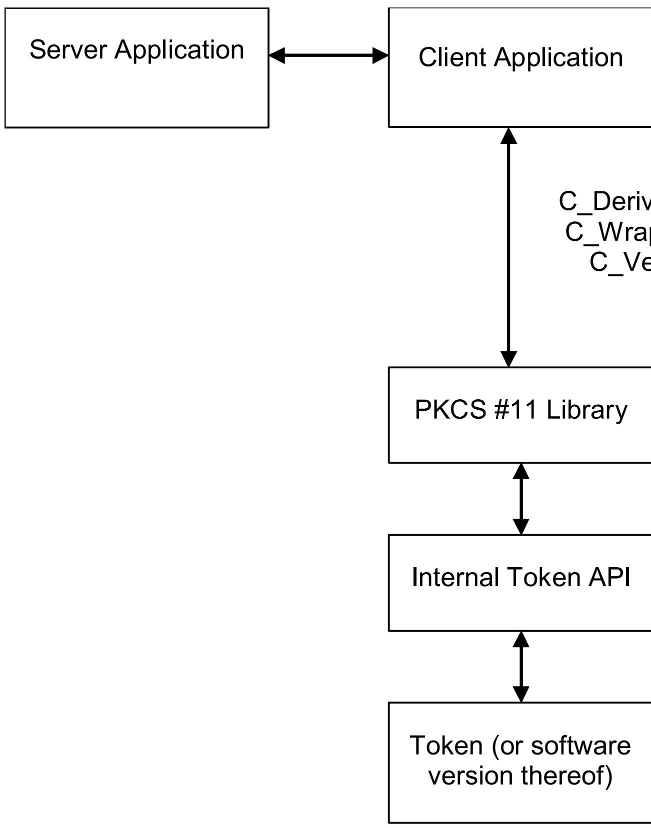

Figure 1: PKCS #11 and CT-KIP integration

Figure 1 shows an integration of PKCS #11 into an application that generates cryptographic keys through the use of CT-KIP. The application invokes **C_DeriveKey** to derive a key of a particular type on the token. The key may subsequently be used as a basis to e.g., generate one-time password values. The application communicates with a CT-KIP server that participates in the key derivation and stores a copy of the key in its database. The key is transferred to the server in wrapped form, after a call to **C_WrapKey**. The server authenticates itself to the client and the client verifies the authentication by calls to **C_Verify**.

4 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. applicable functions

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR[1]	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_KIP_DERIVE							✓
CKM_KIP_WRAP						✓	
CKM_KIP_MAC		✓					

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

4.1 CT-KIP

4.1.1 Definitions

Mechanisms:

```
CKM_KIP_DERIVE
CKM_KIP_WRAP
CKM_KIP_MAC
```

4.1.2 CT-KIP Mechanism parameters

- **CK_KIP_ PARAMS; CK_KIP_ PARAMS_PTR**

CK_KIP_PARAMS is a structure that provides the parameters to all the CT-KIP related mechanisms: The **CKM_KIP_DERIVE** key derivation mechanism, the **CKM_KIP_WRAP** key wrap and key unwrap mechanism, and the **CKM_KIP_MAC** signature mechanism. The structure is defined as follows:

```
typedef struct CK_KIP_PARAMS {
   CK_MECHANISM_PTR  pMechanism;
   CK_OBJECT_HANDLE  hKey;
   CK_BYTE_PTR       pSeed;
   CK_ULONG          ulSeedLen;
} CK_KIP_PARAMS;
```

The fields of the structure have the following meanings:

pMechanism	pointer to the underlying cryptographic mechanism (e.g. AES, SHA-256), see further [2], Appendix D
hKey	handle to a key that will contribute to the entropy of the derived key (CKM_KIP_DERIVE) or will be used in the MAC operation (CKM_KIP_MAC)
pSeed	pointer to an input seed
ulSeedLen	length in bytes of the input seed

CK_KIP_PARAMS_PTR is a pointer to a **CK_KIP_PARAMS** structure.

4.1.3 CT-KIP key derivation

The CT-KIP key derivation mechanism, denoted **CKM_KIP_DERIVE**, is a key derivation mechanism that is capable of generating secret keys of potentially any type, subject to token limitations.

It takes a parameter of type **CK_KIP_PARAMS** which allows for the passing of the desired underlying cryptographic mechanism as well as some other data. In particular, when the *hKey* parameter is a handle to an existing key, that key will be used in the key derivation in addition to the *hBaseKey* of **C_DeriveKey**. The *pSeed* parameter may be used to seed the key derivation operation.

The mechanism derives a secret key with a particular set of attributes as specified in the attributes of the template for the key.

The mechanism contributes the **CKA_CLASS** and **CKA_VALUE** attributes to the new key. Other attributes supported by the key type may be specified in the template for the key, or else will be assigned default initial values. Since the mechanism is generic, the **CKA_KEY_TYPE** attribute should be set in the template, if the key is to be used with a particular mechanism.

4.1.4 CT-KIP key wrap and key unwrap

The CT-KIP key wrap and unwrap mechanism, denoted **CKM_KIP_WRAP**, is a key wrap mechanism that is capable of wrapping and unwrapping generic secret keys.

It takes a parameter of type **CK_KIP_PARAMS**, which allows for the passing of the desired underlying cryptographic mechanism as well as some other data. It does not make use of the *hKey* parameter of **CK_KIP_PARAMS**.

4.1.5 CT-KIP signature generation

The CT-KIP signature (MAC) mechanism, denoted **CKM_KIP_MAC**, is a mechanism used to produce a message authentication code of arbitrary length. The keys it uses are secret keys.

It takes a parameter of type **CK_KIP_PARAMS**, which allows for the passing of the desired underlying cryptographic mechanism as well as some other data. The mechanism does not make use of the *pSeed* and the *ulSeedLen* parameters of **CT_KIP_PARAMS**.

This mechanism produces a MAC of the length specified by *pulSignatureLen* parameter in calls to **C_Sign**.

If a call to **C_Sign** with this mechanism fails, then no output will be generated.

A. Manifest constants

A.1 Notice regarding header files

A C or C++ source file in a Cryptoki application or library can define all the types, and mechanisms described here by including the header file ct-kip.h. The inclusion of the ct-kip.h header file should be preceded by an inclusion of the top-level Cryptoki header file pkcs11.h, and the source file must also specify the preprocessor directives indicated in Section 8 of [1].

A.2 Mechanisms

```
#define CKM_KIP_DERIVE          0x00000510
#define CKM_KIP_WRAP            0x00000511
#define CKM_KIP_MAC             0x00000512
```

B. Using PKCS #11 with CT-KIP

A suggested procedure to perform CT-KIP with a cryptographic token through the PKCS #11 interface using the mechanisms defined herein is as follows (see also [1]):

a. On the client side,

   I. The client selects a suitable slot and token (e.g. through use of the <TokenID> or the <PlatformInfo> element of the CT-KIP trigger message).

   II. Optionally, a nonce *R* is generated, e.g. by calling **C_SeedRandom** and **C_GenerateRandom**.

   III. The client sends its first message to the server, potentially including the nonce *R*.

b. On the server side,

   I. A nonce R_S is generated, e.g. by calling **C_SeedRandom** and **C_GenerateRandom**.

   II. If the server needs to authenticate its first CT-KIP message, and use of **CKM_KIP_MAC** has been negotiated, it calls **C_SignInit** with **CKM_KIP_MAC** as the mechanism followed by a call to **C_Sign**. In the call to **C_SignInit**, K_{AUTH} (see [2]) shall be the signature key, the *hKey* parameter in the **CK_KIP_PARAMS** structure shall be set to NULL_PTR, the *pSeed* parameter of the **CT_KIP_PARAMS** structure shall also be set to NULL_PTR and the *ulSeedLen* parameter shall be set to zero. In the call to **C_Sign**, the *pData* parameter shall be set to point to (the concatenation of the nonce *R*, if received, and) the nonce R_S (see [2] for a definition of the variables), and the *ulDataLen* parameter shall hold the length of the (concatenated) string. The desired length of the MAC shall be specified through the *pulSignatureLen* parameter as usual.

III. The server sends its first message to the client, including R_S, the server's public key K (or an identifier for a shared secret key K), and optionally the MAC.

c. On the client side,

I. If a MAC was received, it is verified. If the MAC does not verify, or was required but not received, the protocol session ends with a failure.

II. If the MAC verified, or was not required and not present, a generic secret key, R_C, is generated by calling **C_GenerateKey** with the **CKM_GENERIC_SECRET_KEY_GEN** mechanism. The *pTemplate* attribute shall have **CKA_EXTRACTABLE** and **CKA_SENSITIVE** set to **CK_TRUE**, and should have **CKA_ALLOWED_MECHANISMS** set to **CKM_KIP_DERIVE** only.

III. The generic secret key R_C is wrapped by calling **C_WrapKey**. If the server's public key is used to wrap R_C, and that key is temporary only, then the **CKA_EXTRACTABLE** attribute of R_C shall be set to CK_FALSE once R_C has been wrapped and the server's public key is to be destroyed. If a shared secret key is used to wrap R_C, and use of the CT-KIP key wrapping algorithm was negotiated, then the **CKM_KIP_WRAP** mechanism shall be used. The *hKey* handle in the **CK_KIP_PARAMS** structure shall be set to NULL_PTR. The *pSeed* parameter in the **CK_KIP_PARAMS** structure shall point to the nonce R_S provided by the CT-KIP server, and the *ulSeedLen* parameter shall indicate the length of R_S. The *hWrappingKey* parameter in the call to **C_WrapKey** shall be set to refer to the wrapping key.

IV. The client sends its second message to the server, including the wrapped generic secret key R_C.

d. On the server side,

I. Once the wrapped generic secret key R_C has been received, the server calls **C_UnwrapKey**. If use of the CT-KIP key wrapping algorithm was negotiated, then **CKM_KIP_WRAP** shall be used to unwrap R_C. When calling **C_UnwrapKey**, the **CK_KIP_PARAMS** structure shall be set as described in c.III above. The *hUnwrappingKey* function parameter shall refer to the shared secret key and the *pTemplate* function parameter shall have **CKA_SENSITIVE** set to **CK_TRUE**, **CKA_KEY_TYPE** set to **CKK_GENERIC_SECRET** and should have **CKA_ALLOWED_MECHANISMS** set to **CKM_KIP_DERIVE** only. This will return a handle to the generic secret key R_C.

II. A token key, K_{TOKEN}, is derived from R_C by calling **C_DeriveKey** with the **CKM_KIP_DERIVE** mechanism, using R_C as *hBaseKey*. The *hKey* handle in the **CK_KIP_PARAMS** structure shall refer either to the public key supplied by the CT-KIP server, or alternatively, the shared secret key indicated by the server. The *pSeed* parameter shall point to the nonce R_S provided by the CT-KIP server, and the *ulSeedLen* parameter shall indicate the length of R_S. The *pTemplate* attribute shall be set in accordance with local policy and as negotiated in the protocol. This will return a handle to the token key, K_{TOKEN}.

III. For the server's last CT-KIP message to the client, if use of the CT-KIP MAC algorithm has been negotiated, then the MAC is calculated by calling **C_SignInit** with the **CKM_KIP_MAC** mechanism followed by a call to **C_Sign**. In the call to **C_SignInit**, K_{AUTH} (see [2]) shall be the signature key, the *hKey* parameter in the **CK_KIP_PARAMS** structure shall be a handle to the generic secret key R_C, the *pSeed* parameter of the **CT_KIP_PARAMS** structure shall be set to NULL_PTR, and the *ulSeedLen* parameter shall be set to zero. In the call to **C_Sign**, the *pData* parameter shall be set to NULL_PTR and the *ulDataLen* parameter shall be set to 0. The desired length of the MAC shall be specified through the *pulSignatureLen* parameter as usual.

IV. The server sends its second message to the client, including the MAC.

e. On the client side,

I. The MAC is verified in a reciprocal fashion as it was generated by the server. If use of the **CKM_KIP_MAC** mechanism was negotiated, then in the call to **C_VerifyInit,** the *hKey* parameter in the **CK_KIP_PARAMS** structure shall refer to R_C, the *pSeed* parameter shall be set to NULL_PTR, and *ulSeedLen* shall be set to 0. The *hKey* parameter of **C_VerifyInit** shall refer to K_{AUTH}. In the call to **C_Verify**, *pData* shall be set to NULL_PTR, *ulDataLen* to 0, *pSignature* to the MAC value received from the server, and *ulSignatureLen* to the length of the MAC. If the MAC does not verify the protocol session ends with a failure.

II. A token key, K_{TOKEN}, is derived from R_C by calling **C_DeriveKey** with the **CKM_KIP_DERIVE** mechanism, using R_C as *hBaseKey*. The *hKey* handle in the **CK_KIP_PARAMS** structure shall be set to NULL_PTR as token policy must dictate use of the same key as was used to wrap R_C. The *pSeed* parameter shall point to the nonce R_S provided by the CT-KIP server, and the *ulSeedLen* parameter shall indicate the length of R_S. The *pTemplate* attribute shall be set in accordance with local policy and as negotiated and expressed in the protocol. In particular, the value of the **<KeyID>** element in the server's response message may be used as **CKA_ID.** The call to **C_DeriveKey** will, if successful, return a handle to K_{TOKEN}. [1]

[1] When K_{AUTH} is the newly generated K_{TOKEN}, the client will need to call **C_DeriveKey** before calling **C_VerifyInit** and **C_Verify** (since the *hKey* parameter of **C_VerifyInit** shall refer to K_{TOKEN}). In this case, the token should not allow K_{TOKEN} to be used for any other operation than the verification of the MAC value until the MAC has successfully been verified.

C. Intellectual property considerations

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

D. 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] RSA Laboratories. *Cryptographic Token Key Initialization Protocol.* Version 1.0, December 2005. URL: ftp://ftp.rsasecurity.com/pub/otps/ct-kip/ct-kip-v1-0.pdf.

E. About OTPS

The *One-Time Password Specifications* are documents produced by RSA Laboratories in cooperation with secure systems developers for the purpose of simplifying integration and management of strong authentication technology into secure applications, and to enhance the user experience of this technology.

Further development of the OTPS series will occur through mailing list discussions and occasional workshops, and suggestions for improvement are welcome. As four our PKCS documents, results may also be submitted to standards forums. For more information, contact:

OTPS Editor
RSA Laboratories
174 Middlesex Turnpike
Bedford, MA 01730 USA
otps-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/