RSA LABORATORIES

PKCS #11 Mechanisms for One-Time Password Tokens

V1.0 DRAFT 1

RSA Security

February 15, 2005

TABLE OF CONTENTS

1 INTRODUCTION 2

1.1 SCOPE 2

1.2 BACKGROUND 2

1.3 DOCUMENT ORGANIZATION 2

2 USAGE OVERVIEW 3

2.1 CASE 1: GENERATION OF OTP VALUES 3

2.2 CASE 2: VERIFICATION OF PROVIDED OTP VALUES 4

2.3 CASE 3: GENERATION OF OTP KEYS 5

3 OTP KEYS 5

4 OTP MECHANISMS 7

4.1 RSA SECURID 7

4.1.1 RSA SecurID secret key objects 7

4.1.2 RSA SecurID mechanism parameters 8

- *CK_SECURID_PARAMS; CK_SECURID_PARAMS_PTR 8*

4.1.3 RSA SecurID key generation 10

4.1.4 RSA SecurID OTP generation and validation 11

4.1.5 Return values 11

A. MANIFEST CONSTANTS 12

A.1 KEY TYPES 12

A.2 MECHANISMS 12

A.3 ATTRIBUTES 12

A.4 RETURN VALUES 12

B. EXAMPLES OF OTP RETRIEVAL AND VERIFICATION 12

B.1 OTP RETRIEVAL 12

B.2 OTP VERIFICATION 14

C. INTELLECTUAL PROPERTY CONSIDERATIONS 15

D. REFERENCES 16

E. ABOUT OTPS 16

1 Introduction

1.1 Scope

This document describes general PKCS #11 [1] objects, procedures and mechanisms that can be used to retrieve and verify one-time passwords (OTPs) generated by OTP tokens.

1.2 Background

A One-Time Password (OTP) 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 generates one-time passwords that may be used to authenticate a user towards some service. Increasingly, these tokens work in a connected fashion, enabling programmatic retrieval of their OTP values. To meet the needs of applications wishing to access these connected OTP tokens in an interoperable manner, this document extends PKCS #11 [1] to better support these tokens, easing the task for vendors of OTP-consuming applications, and enabling a better user experience.

This document adds basic support of One-Time Password (OTP) tokens to PKCS #11 by defining a common OTP key type with an extensible set of attributes and by describing how PKCS #11 functions can be used to retrieve and verify OTP values generated by an OTP token. It also describes an OTP key generation mechanism that may be used to execute on-token key generation.

Building on the OTP framework, the document specifies the PKCS #11 RSA SecurID™ OTP mechanisms[1]. Additional mechanisms may be defined separately to support other types of OTP tokens.

1.3 Document organization

The organization of this document is as follows:

- Section 1 is an introduction.
- Section 2 provides an overview description of the support for OTP tokens in PKCS #11 defined herein.
- Section 3 defines the new OTP key object type and its attributes.
- Section 4 defines specific OTP mechanisms.
- Appendix A collects defined PKCS #11 constants.
- Appendix B provides example usages of the OTP mechanisms.

[1] RSA SecurID® two-factor authentication is a symmetric authentication method which is patented by RSA Security. A user authenticates by submitting a one-time password (OTP), or PASSCODE value generated by an RSA SecurID token. The RSA SecurID 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 the personal computer

- 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 Usage overview

OTP tokens represented as PKCS #11 mechanisms may be used in a variety of ways. The usage cases can be categorized according to the type of sought functionality.

2.1 Case 1: Generation of OTP values

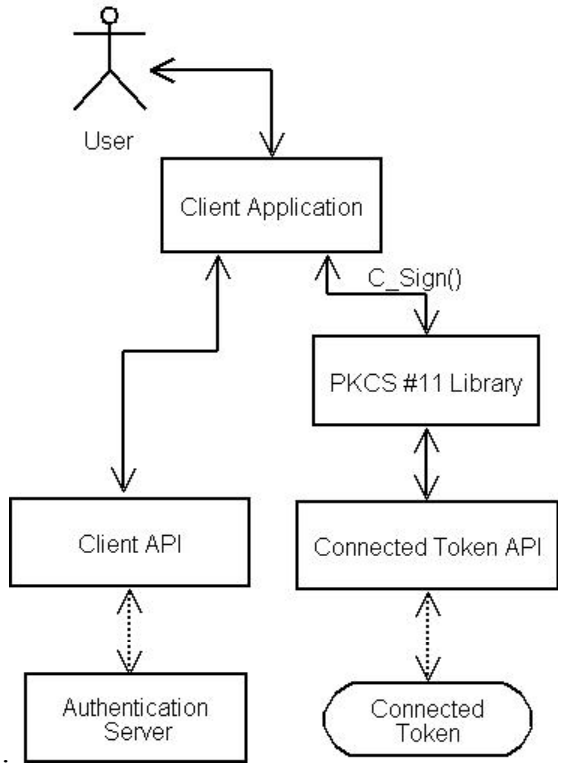

Figure 1: Retrieving OTP values through C_Sign

Figure 1 shows an integration of PKCS #11 into an application that needs to authenticate users holding OTP tokens. In this particular example, a connected hardware token is used, but a software token is equally possible. The application invokes **C_Sign** to retrieve the OTP value from the token. In the example, the application then passes the retrieved OTP value to a client API that sends it via the network to an authentication server. The client API may implement a standard authentication protocol such as RADIUS [2] or EAP [3], or a proprietary protocol such as that used by RSA Security's ACE/Agent® software.

2.2 Case 2: Verification of provided OTP values

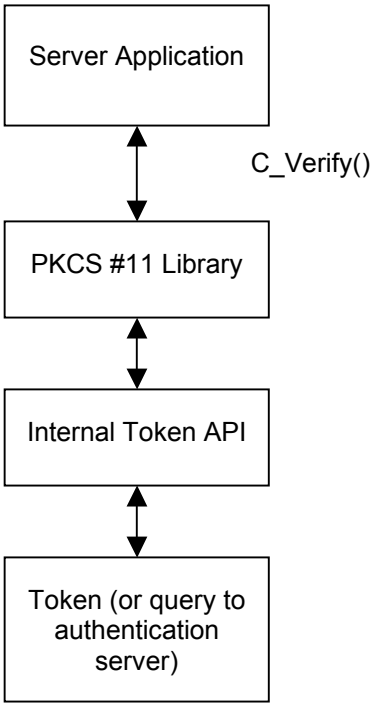

Figure 2: Server-side verification of OTP values

Figure 2 illustrates the server-side equivalent of the scenario depicted in Figure 1. In this case, a server application invokes **C_Verify** with the received OTP value as the signature value to be verified.

2.3 Case 3: Generation of OTP keys

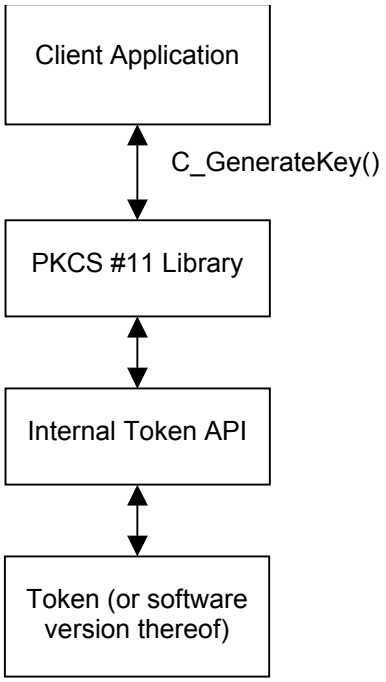

Figure 3: Generation of an OTP key

Figure 3 shows an integration of PKCS #11 into an application that generates OTP keys. The application invokes **C_GenerateKey** to generate an OTP key of a particular type on the token. The key may subsequently be used as a basis to generate OTP values.

3 OTP keys

OTP key objects (object class **CKO_SECRET_KEY)** hold secret keys used by OTP tokens. The following table defines the attributes common to all OTP keys, in addition to the common attributes defined for this object class:

Table 1: Common OTP key attributes

Attribute	Data type	Meaning
CKA_SERIAL_NUMBER[1, 3]	Byte array	Serial number of key
CKA_OTP_FORMAT	CK_ULONG	Format of OTP values produced with this key: 0 = Decimal (default) 1 = Hexadecimal 2 = Alphanumeric
CKA_OTP_LENGTH[1, 9]	CK_ULONG	Length of OTP values (in the CKA_OTP_FORMAT) produced with this key.

Attribute	Data type	Meaning
CKA_OTP_ PINPAD[9]	CK_BBOOL	CK_TRUE if the token will mix in a user-supplied PIN into OTP computations made with this key.
CKA_ OTP_APP_BASED	CK_BBOOL	CK_TRUE if the token will mix in application information into OTP computations made with this key (see below). Default is CK_FALSE.
CKA_OTP CHALLENGE_MODE	CK_BBOOL	CK_TRUE if the token supports calculation of OTPs based on a provided challenge for this key. Default is CK_FALSE.
CKA_OTP_TIME_MODE	CK_BBOOL	True if the token supports calculation of OTPs based on the current time for this key. Default is CK_TRUE.
CKA_OTP_COUNTER_MODE	CK_BBOOL	True if the token supports calculation of OTPs based on a counter value for this key. Default is CK_FALSE.
CKA_OTP_ACCEPT_TIME	CK_BBOOL	True if the token supports verification of OTPs based on a provided timestamp rather than the internal, current time for this key. Default is CK_FALSE. This attribute carries no meaning when CKA_OTP_TIME_MODE is CK_FALSE.
CKA_ OTP_DEFAULT_PIN_ALLOWED	CK_BBOOL	CK_TRUE if successful login to the token will allow a token-resident PIN to be included in OTP computations made with this key (i.e. user will not have to supply a separate PIN when requesting OTP computations). Default is CK_FALSE.
CKA_OTP_DEFAULT_PIN_SET	CK_BBOOL	CK_TRUE it the default PIN is set for this key. Default is CK_FALSE.
CKA_OTP_DEFAULT_PIN[6, 7]	RFC 2279 string	Default PIN for this key. Default value is empty (i.e. *ulValueLen* = 0).
CKA_OTP_SERVICE_IDENTIFIER	RFC 2279 string	Identifies a service that may validate OTPs generated by this key. Default value is empty (i.e. *ulValueLen* = 0).
CKA_VALUE[1, 4, 6, 7]	Byte array	Value of the key.
CKA_VALUE_LEN[2, 3]	CK_ULONG	Length in bytes of key value.

Refer to Table 15 in [1] for table footnotes

An application may find out if a default PIN is supported for a given OTP key by calling **C_GetAttributeValue** and querying for **CKA_DEFAULT_PIN_ALLOWED**. An application may find out if a default PIN has been set for a given OTP key by calling

C_GetAttributeValue and querying for **CKA_DEFAULT_PIN_SET**. An application may set the default PIN by calling **C_SetAttributeValue** with **CKA_DEFAULT_PIN**. The value of the **CKA_DEFAULT_PIN_ALLOWED** attribute cannot be set by an application. The value of the **CKA_DEFAULT_PIN_SET** attribute can only be set indirectly by an application (by setting the PIN).

For OTP tokens with multiple keys, the keys may be enumerated using **C_FindObjects**. The CKA_OTP_SERVICE_IDENTIFIER attribute may be used to distinguish between keys. The actual choice of key for a particular operation is however application-specific and beyond the scope of this document.

4 OTP mechanisms

The following table shows, for the OTP 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 2: OTP mechanisms vs. applicable functions

Mechanism	Functions						
	Encrypt & Decrypt	Sign & Verify	SR & VR[1]	Digest	Gen. Key/ Key Pair	Wrap & Unwrap	Derive
CKM_SECURID_KEY_GEN					✓		
CKM_SECURID_TRADITIONAL		✓					
CKM_SECURID		✓					

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

4.1 RSA SecurID

4.1.1 RSA SecurID secret key objects

RSA SecurID key objects (object class **CKO_SECRET_KEY,** key type **CKK_SECURID**) hold RSA SecurID secret keys. The following table defines the RSA SecurID secret key object attributes, in addition to the common attributes defined for this object class, and in addition to the common OTP key attributes defined above:

Table 1, RSA SecurID secret key object attributes

Attribute	Data type	Meaning
CKA_OTP_TIME_INTERVAL[1]	CK_ULONG	Interval between OTP values produced with this key, in seconds. Default is 60.

Refer to Table 15 in [1] for table footnotes

The following is a sample template for creating an RSA SecurID secret key object:

```
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SECURID;
CK_UTF8CHAR label[] = "RSA SecurID secret key object";
CK_BYTE serialNumber[]= {...};
CK_ULONG outputFormat = 0;
CK_ULONG outputLength = 6;
CK_ULONG timeInterval = 60;
CK_DATE endDate = {...};
CK_BYTE value[] = {...};
CK_BBOOL true = TRUE;
CK_ATTRIBUTE template[] = {
   {CKA_CLASS, &class, sizeof(class)},
   {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
   {CKA_END_DATE, &endDate, sizeof(endDate)},
   {CKA_TOKEN, &true, sizeof(true)},
   {CKA_SENSITIVE, &true, sizeof(true)},
   {CKA_LABEL, label, sizeof(label)-1},
   {CKA_SIGN, &true, sizeof(true)},
   {CKA_VERIFY, &true, sizeof(true)},
   {CKA_SERIAL_NUMBER, serialNumber},
   {CKA_OTP_FORMAT, &outputFormat,
        sizeof(outputFormat)},
   {CKA_OTP_LENGTH, &outputLength,
        sizeof(outputLength)},
   {CKA_OTP_TIME_INTERVAL, &timeInterval,
        sizeof(timeInterval)},
   {CKA_VALUE, value, sizeof(value)}
};
```

4.1.2 RSA SecurID mechanism parameters

♦ **CK_SECURID_PARAMS; CK_SECURID_PARAMS_PTR**

CK_SECURID_PARAMS is a structure that provides the parameters to the RSA SecurID OTP retrieval mechanisms.

```
typedef struct CK_SECURID_PARAMS {
   CK_CHAR         utcTime[14];
   CK_FLAGS        flags;
   CK_UTF8CHAR_PTR pPIN;
   CK_UTF8CHAR_PTR pApplicationID;
```



```
    CK_VOID_PTR     pReserved;
} CK_SECURID_PARAMS;
```

The fields of the structure have the following meanings:

utcTime	UTC time value in the form YYYYMMDDhhmmss used as the input for computing or verifying time-based OTP values. This value will be ignored for OTP values based only on a challenge. It will also be ignored if the token has its own clock, unless the CKA_OTP_ACCEPT_TIME attribute is set to CK_TRUE for the key in question and the operation is an OTP value verification.
flags	bit flags indicating characteristics of the sought OTP value as defined below.
pPIN	a NULL-terminated UTF8 string containing a UTF8-encoded RSA SecurID PIN. The parameter can be set to NULL_PTR when the key's CKA_ DEFAULT_PIN_SET attribute is CK_TRUE, if use of the default PIN is desired.
pApplicationID	a NULL-terminated UTF8 string containing a UTF8-encoded application-specific identifier, which for some keys and tokens may be used to influence the OTP calculation on a per application basis. Any supplied parameter value will be ignored unless the key's CKA_OTP_ APP_BASED attribute is set to CK_TRUE. The value may be provided to the calling application through configuration, external protocol interaction, or by other means. For a client-server based application, this would typically be the IP address (in "numerical dot-notation" and as seen by the client) of the server requesting user authentication. This parameter carries no meaning and shall be set to NULL when the mechanism is CKM_SECURID_TRADITIONAL.
pReserved	reserved for future use. Should be NULL_PTR for this version. Cryptoki libraries shall ignore this parameter.

The following table defines the *flags* field:

Table 2, RSA SecurID OTP Mechanism Flags

Bit Flag	Mask	Meaning
CKF_NEXT_OTP	0x00000001	True if the token shall return the next OTP, rather than the current one. Does not carry any meaning if CKF_CHALLENGE_ONLY (see below) is set.
CKF_CHALLENGE_ONLY	0x00000002	True if the OTP value shall be computed on a provided challenge without using the current time (assuming that both CKA_OTP_CHALLENGE_MODE and CKA_OTP_TIME_MODE are CK_TRUE for the key in question, if a challenge is provided, the default behavior is to include both the challenge and the current time in the OTP computation. This flag carries no meaning and shall not be set when the mechanism is CKM_SECURID_TRADITIONAL.
CKF_TOKEN_CODE	0x00000004	True if the RSA SecurID OTP returned by **C_Sign** shall be a tokencode, rather than a PASSCODE™. When this flag is set, any provided PIN information will be ignored (whether provided by default or not).

CK_SECURID_PARAMS_PTR is a pointer to a **CK_SECURID_PARAMS**.

4.1.3 RSA SecurID key generation

The RSA SecurID key generation mechanism, denoted **CKM_SECURID_KEY_GEN**, is a key generation mechanism for the RSA SecurID algorithm.

It does not have a parameter.

The mechanism generates RSA SecurID keys with a particular set of attributes as specified in the template for the key.

The mechanism contributes at least the **CKA_CLASS**, **CKA_KEY_TYPE**, **CKA_VALUE_LEN**, and **CKA_VALUE** attributes to the new key. Other attributes supported by the RSA SecurID key type 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 SecurID key sizes, in bytes.

4.1.4 RSA SecurID OTP generation and validation

CKM_SECURID_TRADITIONAL and CKM_SECURID are the mechanisms for the retrieval and verification of RSA SecurID OTP values based on the current time, a provided challenge, or the current time and a provided challenge. As indicated above, other information may also be included in the OTP computation. CKM_SECURID_TRADITIONAL is the mechanism used by most existing RSA SecurID tokens, while CKM_SECURID is the mechanism supported by newer tokens.

These mechanisms take a pointer to a CK_SECURID_PARAMS structure as a parameter. The *utcTime* value in the structure may be NULL if the token has an internal clock. For tokens with an internal clock, any provided *utcTime* values will be ignored, unless the CKA_OTP_ACCEPT_TIME attribute for the key in question is CK_TRUE and the operation is a verification of an OTP value (this allows for verification of old OTP values). Likewise, any provided *utcTime* values will be ignored if CKF_CHALLENGE_ONLY is set to true in the CK_SECURID_PARAMS structure.

When signing using the CKM_SECURID mechanism, the data to be signed, *pData*, is the challenge. Note that providing a challenge when the mechanism is CKM_SECURID_TRADITIONAL or the attribute CKA_OTP_CHALLENGE_MODE is CK_FALSE for the key in question will result in an error (CKR_ARGUMENTS_BAD). Similarly, it is an error (CKR_ARGUMENTS_BAD) to set the CKF_CHALLENGE_ONLY flag and not provide a challenge. When no challenge is provided, *pData* shall be set to NULL_PTR.

4.1.5 Return values

Support for any of the CKM_SECURID_TRADITIONAL and CKM_SECURID mechanisms extends the set of return values for **C_Verify** with the following values:

- CKR_SECURID_NEW_PIN_MODE: The supplied OTP was not accepted and the library requests a new OTP computed using a new PIN. The new PIN is set through means out of scope for this document.
- CKR_SECURID_NEXT_TOKEN_CODE: The supplied OTP was correct but indicated a larger than normal drift in the token's internal clock. To ensure this was not due to a temporary problem, the application should provide the next one-time *tokencode* (not one-time *PASSCODE*, see above) to the library for verification.

A. Manifest constants

A.1 Key types

```
#define CKK_SECURID                     0x00000022
```

A.2 Mechanisms

```
#define CKM_SECURID_KEY_GEN             0x00000280
#define CKM_SECURID_TRADITIONAL         0x00000281
#define CKM_SECURID                     0x00000282
```

A.3 Attributes

```
#define CKA_OTP_FORMAT                  0x00000220
#define CKA_OTP_LENGTH                  0x00000221
#define CKA_OTP_TIME_INTERVAL           0x00000222
#define CKA_OTP_PINPAD                  0x00000223
#define CKA_OTP_APP_BASED               0x00000224
#define CKA_OTP_CHALLENGE_MODE          0x00000225
#define CKA_OTP_TIME_MODE               0x00000226
#define CKA_OTP_ACCEPT_TIME             0x00000227
#define CKA_OTP_DEFAULT_PIN_ALLOWED     0x00000228
#define CKA_OTP_DEFAULT_PIN_SET         0x00000229
#define CKA_OTP_DEFAULT_PIN             0x0000022A
#define CKA_OTP_SERVICE_IDENTIFIER      0x0000022B
#define CKA_OTP_COUNTER_MODE            0x0000022C
```

A.4 Return values

```
#define CKR_SECURID_NEW_PIN_MODE        0x000001B0
#define CKR_SECURID_NEXT_TOKEN_CODE     0x000001B1
```

B. Examples of OTP retrieval and verification

B.1 OTP retrieval

The following sample code snippet illustrates the retrieval of an OTP value from an RSA SecurID token using the **C_Sign** function. The data to be signed is in the example NULL_PTR, since an OTP value only based on the current time is sought. The current UTC time, if a time is specified, is supplied in the CK_SECURID_PARAMS structure.

```
CK_SESSION_HANDLE hSession;
CK_OBJECT_HANDLE hKey;
CK_CHAR szTime = {. . .};
/* UTC time value for OTP, or NULL */
CK_UTF8CHAR szPIN[] = "...";
CK_SECURID_PARAMS params = {szTime, 0, szPIN, NULL_PTR,
        NULL_PTR};
```



```
CK_MECHANISM mechanism = {CKM_SECURID, &params,
        sizeof(params)};
CK_ULONG ulOTPLen, ulKeyCount;
CK_BYTE *pOTP;   /* Storage for OTP result */
CK_RV rv;
CK_OBJECT_CLASS class = CKO_SECRET_KEY;
CK_KEY_TYPE keyType = CKK_SECURID;
CK_ATTRIBUTE template[] = {
  {CKA_CLASS, &class, sizeof(class)},
  {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
};
/* Find the RSA SecurID key on the token. */
rv = C_FindObjectsInit(hSession, template, 2);
if (rv == CKR_OK) {
   rv = C_FindObjects(hSession, &hKey, 1, &ulKeyCount);
   rv = C_FindObjectsFinal(hSession);
}

if ((rv != CKR_OK) || (ulKeyCount == 0)) {
   printf("\nError: unable to find RSA SecurID key on
        token.\n");
   return(rv);
}

/* Initialize the signing mechanism. */
rv = C_SignInit(hSession, &mechanism, hKey);
if (rv == CKR_OK) {
    ulOTPLen = sizeof(OTP);
    /* Get the buffer length needed for the OTP Value.
        */
    rv = C_Sign(hSession, NULL_PTR, 0, NULL_PTR,
        &ulOTPLen);
    if (rv == CKR_OK) {
        pOTP = malloc(ulOTPLen);
        if (pOTP != NULL_PTR) {
            /* Get the actual OTP value. */
            rv = C_Sign(hSession, NULL_PTR, 0, pOTP,
            &ulOTPLen);
        }
    }
}

if ((rv != CKR_OK) || (pOTP == NULL_PTR)) {
   printf("\nError retrieving OTP Value.\n");
   return(rv);
}
```



```
   printf("\nReturned OTP Value is %s.\n", pOTP);

   return(rv);
```

B.2 OTP verification

The following sample code snippet illustrates the verification of an OTP value from an RSA SecurID token, using the **C_Verify** function. The data to be verified is in the example NULL_PTR, since the OTP value was computed on the current time and not a challenge. The current UTC time, if a time is specified, is supplied in the CK_SECURID_PARAMS structure.

```
   CK_SESSION_HANDLE hSession;
   CK_OBJECT_HANDLE hKey;
   CK_CHAR szTime = {. . .};
   /* UTC time value for OTP, or NULL */
   CK_UTF8CHAR szPIN[] = "...";    /* PIN, from app. */
   CK_SECURID_PARAMS params = {szTime, 0, szPIN, NULL_PTR,
           NULL_PTR};
   CK_MECHANISM mechanism = {CKM_SECURID, &params,
           sizeof(params)};
   CK_ULONG ulKeyCount;
   CK_RV rv;
   CK_BYTE OTP = {. . .};     /* Supplied OTP value. */
   CK_ULONG ulOTPLen = strlen((CK_CHAR_PTR)OTP);
   CK_OBJECT_CLASS class = CKO_SECRET_KEY;
   CK_KEY_TYPE keyType = CKK_SECURID;

   CK_ATTRIBUTE template[] = {
     {CKA_CLASS, &class, sizeof(class)},
     {CKA_KEY_TYPE, &keyType, sizeof(keyType)},
   };

   /* Find the RSA SecurID key on the token. */
   rv = C_FindObjectsInit(hSession, template, 2);
   if (rv == CKR_OK) {
      rv = C_FindObjects(hSession, &hKey, 1, &ulKeyCount);
      rv = C_FindObjectsFinal(hSession);
   }

   if ((rv != CKR_OK) || (ulKeyCount == 0)) {
      printf("\nError: unable to find RSA SecurID key on
           token.\n");
      return(rv);
}

   rv = C_VerifyInit(hSession, &mechanism, hKey);
```



```
if (rv == CKR_OK) {
  ulOTPLen = sizeof(OTP);
  rv = C_Verify(hSession, NULL_PTR, 0, OTP, ulOTPLen);
}

switch(rv) {
    case CKR_OK:
        printf("\nSupplied OTP value is correct.\n");
        break;

    case CKR_SIGNATURE_INVALID:
        printf("\nSupplied OTP value is incorrect.\n");
        break;

    default:
        printf("\nError:Unable to verify OTP value.\n");
        break;
}

return(rv);
```

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. The RSA SecurID technology is covered by a number of US patents (and foreign counterparts), in particular US patent nos. 4,720,860, 4,856,062, 4,885,778, 5,097,505, 5,168,520, and 5,657,388. Additional patents are pending.

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] Rigney et al, "Remote Authentication Dial In User Service (RADIUS)", IETF RFC2865, June 2000. URL: http://ietf.org/rfc/rfc2865.txt.

[3] Aboba et al, "Extensible Authentication Protocol (EAP)", IETF RFC 3748, June 2004. URL: http://ietf.org/rfc/rfc3748.txt.

E. About OTPS

The *One-Time Password Specifications* are documents produced by RSA Security 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.

RSA Security plans further development of the OTPS series 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 Security
174 Middlesex Turnpike
Bedford, MA 01730 USA
otps-editor@rsasecurity.com
http://www.rsasecurity.com/rsalabs/

