RSA LABORATORIES

Extensions to CT-KIP to support one- and two-pass key initialization

Draft 2

RSA Laboratories

March 27, 2006

Editor's note: This is the second draft of this document. Please send comments and suggestions to: otps@rsasecurity.com or otps-editor@rsasecurity.com.

TABLE OF CONTENTS

1 Introduction

1.1 Scope

This document describes extensions to the Cryptographic Token Key Initialization Protocol (CT-KIP) [1] to support one-pass (i.e. one *message*) and two-pass (i.e. one *round-trip*) cryptographic token key initialization.

1.2 Background

There are several deployment scenarios where it would be beneficial to have one- or two-pass versions of CT-KIP. These include situations where a direct communication between the OTP token and the CT-KIP server is not possible, where work-flow constraints otherwise would limit real-time communications (e.g. needs for administrators to authorize processes), or where network latency or other design constraints makes a four-pass version less suitable.

This document tries to meet the needs of these scenarios by describing extensions to CT-KIP for the initialization of cryptographic tokens in one round-trip or less.

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 defines extensions to CT-KIP.
- Section 4 discusses security considerations.
- Appendix A contains two profiles of CT-KIP.
- Appendix B provides example messages.
- Appendix C discusses integration with PKCS #11 [5].
- Appendices D, E, and F 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

2.2 Notation

||	String concatenation
ID_C	Identifier for CT-KIP client

ID_S	Identifier for CT-KIP server
K_{AUTH}	Secret key used for authentication purposes
K_{TOKEN}	Secret key used for token computations, generated in CT-KIP
K_{CLIENT}	Public key of CT-KIP client (token)
K_{SHARED}	Secret key shared between the cryptographic token and the CT-KIP server
$K_{DERIVED}$	Secret key derived from a passphrase that is known to both the cryptographic token or user and the CT-KIP server
R	Pseudorandom value chosen by the cryptographic token and used for MAC computations

The following typographical conventions are used in the body of the text: **<XML Element>**, ***XMLAttribute***, **XMLType**.

3 One- and two-pass CT-KIP

3.1 Protocol flow

3.1.1 One-pass CT-KIP

In one-pass CT-KIP, the server simply sends a **<ServerFinished>** message to the CT-KIP client. In this case, there is no exchange of the **<ClientHello>**, **<ServerHello>**, and **<ClientNonce>** CT-KIP messages, and hence there is no way for the client to express supported algorithms etc. Before attempting one-pass CT-KIP, the server must therefore have prior knowledge not only that the client is able and willing to accept this variant of CT-KIP, but also of algorithms and key types supported by the client.

Outside the specific cases where one-pass CT-KIP is desired, clients should be constructed and configured to only accept CT-KIP server messages in response to client-initiated transactions.

3.1.2 Two-pass CT-KIP

In two-pass CT-KIP, the client's initial **<ClientHello>** message is directly followed by a **<ServerFinished>** message. There is no exchange of the **<ServerHello>** message or the **<ClientNonce>** message. Essentially, two-pass CT-KIP is a transport of key material from the CT-KIP server to the CT-KIP client. However, as the two-pass variant of CT-KIP consists of one round trip to the server, the client is still able to specify algorithm preferences, etc. in the **<ClientHello>** message.

3.2 Extensions to the <ClientHello> message

A new extension is defined for this message. The extension signals client support for the two-pass version of CT-KIP, informs the server of supported two-pass variants, and provides optional payload data to the CT-KIP server. The payload is sent in an opportunistic fashion, and may be discarded by the CT-KIP server if the server does not support the two-pass variant the payload is associated with. Depending on the client's policy, this extension may or may not have the ***Critical*** attribute set to ***true***. If ***Critical*** ≠

"***true***" then a CT-KIP server may ignore the extension and proceed with ordinary 4-pass CT-KIP. Otherwise, the server must find a suitable two-pass variant or else the protocol run will fail.

```
<xs:complexType name="TwoPassSupportType">
 <xs:annotation>
  <xs:documentation xml:lang="en">
   This extension is only valid in ClientHello PDUs. It informs the server of the client's support for
   the two-pass version of CT-KIP, and carries optional payload data associated with each
   supported two-pass key initialization method
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="AbstractExtensionType">
   <xs:sequence maxOccurs="unbounded">
    <xs:element name="SupportedKeyInitializationMethod" type="xs:anyURI"/>
    <xs:element name="Payload" minOccurs="0"/>
   </xs:sequence>
  </xs:extension>
 </xs:complexContent>
</xs:complexType>
```

The elements of this type have the following meaning:

- **<SupportedKeyInitializationMethod>**: A two-pass key initialization method supported by the CT-KIP client. Multiple supported methods may be present, in which case they shall be listed in order of precedence.
- **<Payload>**: An optional payload associated with each supported key initialization method.

A CT-KIP client that indicates support for two-pass CT-KIP must also include the nonce *R* in its **<ClientHello>** message (this will enable the client to verify that the CT-KIP server it is communicating with is alive).

3.3 Extensions to the <ServerFinished> message

A new extension is defined for this message. It carries an identifier for the selected key initialization method as well as key initialization method-dependent payload data.

Servers may include this extension in a **<ServerFinished>** message that is being sent in response to a received **<ClientHello>** message if and only if that **<ClientHello>** message contained a **TwoPassSupportType** extension and the client indicated support for the selected key initialization method. Servers must include this extension in a **<ServerFinished>** message that is sent as a 1-pass CT-KIP.

```
<xs:complexType name="KeyInitializationDataType">
 <xs:annotation>
  <xs:documentation xml:lang="en">
   This extension is only valid in ServerFinished PDUs. It contains key initialization data and its
   presence results in a two-pass (or one-pass, if no ClientHello was sent) CT-KIP exchange.
  </xs:documentation>
 </xs:annotation>
 <xs:complexContent>
  <xs:extension base="AbstractExtensionType">
   <xs:sequence>
    <xs:element name="KeyInitializationMethod" type="xs:anyURI"/>
```



```
      <xs:element name="Payload"/>
    </xs:sequence>
   </xs:extension>
  </xs:complexContent>
</xs:complexType>
```

3.4 MAC calculations

3.4.1 One-pass CT-KIP

In[A1] one-pass CT-KIP, the server is required to include an identifier, ID_S, for itself (in the **<ServerID>** element) in the **<ServerFinished>** message. The MAC value in the **<ServerFinished>** message shall be computed on the (ASCII) string "MAC 2 computation", the server identifier ID_S, and a monotonically increasing, unsigned integer value I guaranteed not to be used again by this server[1], using a MAC key K_{MAC}. In contrast to [1], this key shall be provided together with K_{TOKEN} to the client, and hence there is no need for a K_{AUTH} for key confirmation purposes[2].

If CT-KIP-PRF is used as the MAC algorithm, then the input parameter s shall consist of the concatenation of the (ASCII) string "MAC 2 computation" ID_S, and I. The parameter *dsLen* shall be set to at least 16 (i.e. the length of the MAC shall be at least 16 octets):

$$dsLen \geq 16$$

$$MAC = \text{CT-KIP-PRF}\ (K_{MAC}, \text{"MAC 2 computation"} \parallel ID_S \parallel I, dsLen)$$

The server shall provide I to the client in the ***Nonce*** attribute of the **<MAC>** element of the **<ServerFinished>** message using the following extension of the CT-KIP **MacType**:

```
xs:complexType name="ExtendedMacType">
        <xs:simpleContent>
            <xs:extension base="ct-kip:MacType">
                <xs:attribute name="Nonce" type="xs:nonNegativeInteger" use="required"/>
            </xs:extension>
        </xs:simpleContent>
</xs:complexType>
```

3.4.2 Two-pass CT-KIP

In two-pass CT-KIP, the client is required to include a nonce R in the **<ClientHello>** message. Further, the server is required to include an identifier, ID_S, for itself (in the **<ServerID>** element) in the **<ServerFinished>** message. The MAC value in the **<ServerFinished>** message shall be computed on the (ASCII) string "MAC 2 computation", the server identifier ID_S, and R using a MAC key K_{MAC}. Again, in contrast with [1], this key shall be provided together with K_{TOKEN} to the client, and hence there is no need for a K_{AUTH} for key confirmation purposes.

---

[1] This could be a the number of seconds since some point in time with sufficient granularity, a counter value, or a combination of the two where the counter value is reset for each new time value.

[2] An extension is planned to [1] to allow for generation of a K_{MAC} in addition to K_{TOKEN} in 4-pass CT-KIP.

If CT-KIP-PRF is used as the MAC algorithm, then the input parameter *s* shall consist of the concatenation of the (ASCII) string "MAC 2 computation" and *R*, and the parameter *dsLen* shall be set to the length of *R*:

$$dsLen = len(R)$$

$$MAC = \text{CT-KIP-PRF}\ (K_{MAC}, \text{"MAC 2 computation"} \parallel ID_S \parallel R, dsLen)$$

4 Security considerations

4.1 Applicability of 4-pass CT-KIP security considerations

This document extends [1], and the security considerations discussed in [1] therefore applies here as well, but with some exceptions:

- Message re-ordering attacks cannot occur since in the CT-KIP variants described in this document each party sends at most one message each.
- The attack described in Section 5.5 of [1] where the attacker replaces a client-provided R_C with his own R'_C does not apply as the client does not provide any entropy to K_{TOKEN} in the 1- and 2-pass variants discussed here. The attack as such (and its countermeasures) still applies, however, as it essentially is a man-in-the-middle attack.

In addition, the 1- and 2-pass CT-KIP variants described in this document warrant some further security considerations that are discussed in the following.

4.2 Security considerations specific to 1- and 2-pass CT-KIP

4.2.1 Client contributions to K_{TOKEN} entropy

In 4-pass CT-KIP, both the client and the server provide randomizing material to K_{TOKEN} , in a manner that allows both parties to verify that they did contribute to the resulting key. In the 1- and 2-pass CT-KIP versions defined herein, only the server contributes to the entropy of K_{TOKEN}. This means that a broken or compromised (pseudo-)random number generator in the server may cause more damage than it would in the 4-pass variant. Server implementations should therefore take extreme care to ensure that this situation does not occur.

4.2.2 Replay protection

A 4-pass CT-KIP client knows that a server it is communicating with is "live" since the server must create a MAC on information sent by the client. The same is true for 2-pass CT-KIP thanks to the requirement that the client sends *R* in the Client Hello message and that the server includes *R* in the MAC computation. In 1-pass CT-KIP clients .(tokens)

---

[3] In this case, implementations must protect against attacks where K_{TOKEN} is used to pre-compute MAC values, see further [1].

[4] See footnote **Error! Bookmark not defined.**.

that record the latest I used by a particular server (as identified by ID_S) will be able to detect replays.

4.2.3 Key confirmation

4-pass CT-KIP servers provide key confirmation through the MAC on R_C in the Server Finished message. In the 1- and 2-pass CT-KIP variants described herein, key confirmation is provided when the MAC is made using the newly delivered K_{TOKEN} as K_{AUTH}. It should be noted that even when the MAC is calculated using a separate K_{AUTH}, it still allows the client to verify that it is talking with a trusted and authenticated server, however.

4.2.4 Key protection in the passphrase profile

The passphrase-based key wrap profile uses the PBKDF2 function from [3] to generate an encryption key from a passphrase and salt string. The derived key, $K_{DERIVED}$ is used by the server to encrypt K_{TOKEN} and by the token to decrypt the newly delivered $K_{TOKEN.}$ It is important to note that password-based encryption is generally limited in the security that it provides and despite the use of use of salt and iteration count in PBKDF2 to increase the complexity of attack; Implementations should take additional measures to strengthen the security of the passphrase-based key wrap profile. The following measures should be considered where applicable:

- The passphrase should be selected well, and usage guidelines such as the ones in [9] should be taken into account.
- A different passphrase should be used for every key initialization wherever possible. (E.g. the use of a global passphrase for a batch of tokens should be avoided). One way to achieve this is to use randomly-generated passphrases.
- The passphrase should be protected well if stored on the server and/or on the token and should be delivered to the token's user using secure methods.
- User pre-authentication should be implemented to ensure that K_{TOKEN} is not delivered to a rogue workstation.
- The iteration count in PBKDF2 should be high to impose more work for an attacker using brute-force methods (see [3] for recommendations). However, it must be noted that the higher the count, the more work is required on the legitimate token to decrypt the newly delivered K_{TOKEN}. Servers may use relatively low iteration counts to accommodate tokens with limited processing power such as some PDA and cell phones when other security measures are taken and the security of the password-based key wrap method is not weakened.
- Transport level security (e.g. TLS) should be used where possible to protect a 2-pass or 1-pass protocol run. Transport level security provides a second layer of protection for the newly generated K_{TOKEN}.

XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Copyright (c) RSA Security Inc. 2006. All rights reserved. -->

<xs:schema
  targetNamespace="http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-kip-two-pass#"
  xmlns:ct-kip-two-pass="http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-kip-two-
pass#"
  xmlns:ct-kip="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns="http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-kip-two-pass#">

<!--
<xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
   schemaLocation="http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/xmldsig-core-
schema.xsd"/>
-->
<xs:import namespace="http://www.w3.org/2000/09/xmldsig#"
    schemaLocation="../W3C/xmldsig-core-schema.xsd"/>

<xs:import namespace="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#"
    schemaLocation="ct-kip.xsd"/>

<!—Extended core types -->
<xs:complexType name="ExtendedMacType">
  <xs:simpleContent>
    <xs:extension base="ct-kip:MacType">
      <xs:attribute name="Nonce" type="xs:nonNegativeInteger" use="required"/>
    </xs:extension>
  </xs:simpleContent>
</xs:complexType>

<!—2- and 1-pass extensions -->
<xs:complexType name="TwoPassSupportType">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      This extension is only valid in ClientHello PDUs. It informs the
      server of the client's support for the two-pass version of
      CT-KIP, and carries optional payload data associated with each
      supported two-pass key initialization method.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="ct-kip:AbstractExtensionType">
      <xs:sequence maxOccurs="unbounded">
        <xs:element name="SupportedKeyInitializationMethod" type="xs:anyURI"/>
        <xs:element name="Payload" minOccurs="0"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

<xs:complexType name="KeyInitializationDataType">
  <xs:annotation>
    <xs:documentation xml:lang="en">
      This extension is only valid in ServerFinished PDUs. It contains
```

```
      key initialization data and its presence results in a two-pass
      (or one-pass, if no ClientHello was sent) CT-KIP exchange.
    </xs:documentation>
  </xs:annotation>
  <xs:complexContent>
    <xs:extension base="ct-kip:AbstractExtensionType">
      <xs:sequence>
        <xs:element name="KeyInitializationMethod" type="xs:anyURI"/>
        <xs:element name="Payload"/>
      </xs:sequence>
    </xs:extension>
  </xs:complexContent>
</xs:complexType>

</xs:schema>
```

A. Key initialization profiles of CT-KIP

A.1 Introduction

This appendix introduces two profiles of CT-KIP for key initialization. They may both be used for two- as well as one-pass initialization of cryptographic tokens. Further profiles may be defined by external identities or through the OTPS process.

A.2 Key transport profile

A.2.1 Introduction

This profile initializes the cryptographic token with a symmetric key, K_{TOKEN}, through key transport, using a public key, K_{CLIENT}, whose private key part resides in the token as the transport key.

A.2.2 Identification

This profile shall be identified with the following URI:

http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-kip#transport

A.2.3 Payloads

The client shall send a payload associated with this key initialization method. The payload shall be of type **ds:KeyInfoType** ([7]), and only those choices of the **ds:KeyInfoType** that identify a public key are allowed. The **ds:X509Certificate** option of the **ds:X509Data** alternative is recommended when the public key corresponding to the private key on the cryptographic token has been certified.

The server payload associated with this key initialization method shall be of type **xenc:EncryptedKeyType** ([8]), and only those encryption methods utilizing a public key that are supported by the CT-KIP client (as indicated in the **<SupportedEncryptionAlgorithms>** element of the **<ClientHello>** message) are allowed as values for the **<xenc:EncryptionMethod>** element. Further, the **<ds:KeyInfo>** element shall contain the same value (i.e. identify the same public key) as the **<Payload>** of the corresponding supported key initialization method in the **<ClientHello>** message that triggered the response. The **<xenc:CarriedKeyName>** element may be present, but shall, when present, contain the same

value as the **<KeyID>** element of the **<ServerFinished>** message. The ***Type*** attribute of the **xenc:EncryptedKeyType** shall be present and shall identify the type of the wrapped key. The type shall be one of the types supported by the CT-KIP client (as reported in the **<SupportedKeyTypes>** of the preceding **<ClientHello>** message). The transported key shall consist of two parts of equal length. The first half constitutes K_{MAC} and the second half constitutes K_{TOKEN}. The length of K_{TOKEN} (and hence also the length of K_{MAC}) is determined by the type of K_{TOKEN}.

CT-KIP servers and tokens supporting this profile must support the **http://www.w3.org/2001/04/xmlenc#rsa-1_5** key-wrapping mechanism defined in [8].

When this profile is used, the ***MacAlgorithm*** attribute of the **<Mac>** element of the **<ServerFinished>** message must be present and must identify the selected MAC algorithm. The selected MAC algorithm must be one of the MAC algorithms supported by the CT-KIP client (as indicated in the **<SupportedMACAlgorithms>** element of the **<ClientHello>** message). The MAC shall be calculated as described in Section 3.4.

A.3 Key wrap profile

A.3.1 Introduction

This profile initializes the cryptographic token with a symmetric key, K_{TOKEN}, through key wrap, using a (symmetric) key-wrapping key, K_{SHARED}, known in advance by both the token and the CT-KIP server.

A.3.2 Identification

This profile shall be identified with the following URI:

http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-kip#wrap

A.3.3 Payloads

The client shall send a payload associated with this key initialization method. The payload shall be of type **ds:KeyInfoType** ([7]), and only those choices of the **ds:KeyInfoType** that identify a symmetric key are allowed. The **ds:KeyName** alternative is recommended.

The server payload associated with this key initialization method shall be of type **xenc:EncryptedKeyType** ([8]), and only those encryption methods utilizing a symmetric key that are supported by the CT-KIP client (as indicated in the **<SupportedEncryptionAlgorithms>** element of the **<ClientHello>** message) are allowed as values for the **<xenc:EncryptionMethod>** element. Further, the **<ds:KeyInfo>** element shall contain the same value (i.e. identify the same symmetric key) as the **<Payload>** of the corresponding supported key initialization method in the **<ClientHello>** message that triggered the response. The **<xenc:CarriedKeyName>** element may be present, and shall, when present, contain the same value as the **<KeyID>** element of the **<ServerFinished>** message. The ***Type*** attribute of the **xenc:EncryptedKeyType** shall be present and shall identify the type of the wrapped key. The type shall be one of the types supported by the CT-KIP client (as reported in the **<SupportedKeyTypes>** of the preceding **<ClientHello>** message). The wrapped key shall consist of two parts of equal length. The first half

constitutes K_{MAC} and the second half constitutes K_{TOKEN}. The length of K_{TOKEN} (and hence also the length of K_{MAC}) is determined by the type of K_{TOKEN}.

CT-KIP servers and tokens supporting this profile must support the **http://www.w3.org/2001/04/xmlenc#kw-aes128** key-wrapping mechanism defined in [8].

When this profile is used, the ***MacAlgorithm*** attribute of the **<Mac>** element of the **<ServerFinished>** message must be present and must identify the selected MAC algorithm. The selected MAC algorithm must be one of the MAC algorithms supported by the CT-KIP client (as indicated in the **<SupportedMACAlgorithms>** element of the **<ClientHello>** message). The MAC shall be calculated as described in Section 3.4.

A.4 Passphrase-based key wrap profile

A.4.1 Introduction

This profile is a variation of the key wrap profile. It initializes the cryptographic token with a symmetric key, K_{TOKEN}, through key wrap, using a passphrase-derived key-wrapping key, $K_{DERIVED}$. The passphrase is known in advance by both the token user and the CT-KIP server. To preserve the property of not exposing K_{TOKEN} to any other entity than the CT_KIP server and the token itself, the method should be employed only when the token contains facilities (e.g. a keypad) for direct entry of the passphrase.

A.4.2 Identification

This profile shall be identified with the following URI:

http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-kip#passphrase-wrap

A.4.3 Payloads

The client shall send a payload associated with this key initialization method. The payload shall be of type **ds:KeyInfoType** ([7]). The **ds:KeyName** option shall be used and the key name shall identify the passphrase that will be used by the server to generate the key-wrapping key. As an example, the identifier could be a user identifier or a registration identifier issued by the server to the user during a session preceding the CT-KIP protocol run.

The server payload associated with this key initialization method shall be of type **xenc:EncryptedKeyType** ([8]), and only those encryption methods utilizing a passphrase to derive the key-wrapping key that are supported by the CT-KIP client (as indicated in the **<SupportedEncryptionAlgorithms>** element of the **<ClientHello>** message) are allowed as values for the **<xenc:EncryptionMethod>** element. Further, the **<ds:KeyInfo>** element shall contain the same value (i.e. identify the same passphrase) as the **<Payload>** of the corresponding supported key initialization method in the **<ClientHello>** message that triggered the response. The **<xenc:CarriedKeyName>** element may be present, and shall, when present, contain the same value as the **<KeyID>** element of the **<ServerFinished>** message. The ***Type*** attribute of the **xenc:EncryptedKeyType** shall be present and shall identify the type of the wrapped key. The type shall be one of the types supported by the CT-KIP client (as reported in the **<SupportedKeyTypes>** of the preceding **<ClientHello>** message). The wrapped key shall consist of two parts of equal length. The first half

constitutes K_{MAC} and the second half constitutes K_{TOKEN}. The length of K_{TOKEN} (and hence also the length of K_{MAC}) is determined by the type of K_{TOKEN}.

CT-KIP servers and tokens supporting this profile must support the PBES2 password based encryption scheme defined in [3] (and identified as **http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbes2** in [4]), the PBKDF2 password-based key derivation function also defined in [3] (and identified as **http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2** in [4]), and the **http://www.w3.org/2001/04/xmlenc#kw-aes128** key-wrapping mechanism defined in [8].

When this profile is used, the ***MacAlgorithm*** attribute of the **<Mac>** element of the **<ServerFinished>** message must be present and must identify the selected MAC algorithm. The selected MAC algorithm must be one of the MAC algorithms supported by the CT-KIP client (as indicated in the **<SupportedMACAlgorithms>** element of the **<ClientHello>** message). The MAC shall be calculated as described in Section 3.4.

B. Example messages

B.1 Note regarding the examples

All examples are syntactically correct. MAC and cipher values are fictitious however. The examples illustrate a complete two-pass CT-KIP exchange. The server messages may also constitute the only messages in a one-pass CT-KIP exchange.

B.2 Example of a <ClientHello> message indicating support for two-pass CT-KIP

The client indicates support both for the two-pass key transport variant as well as the two-pass key wrap variant.

```
<?xml version="1.0" encoding="UTF-8"?>
<ClientHello
  xmlns:ct-kip="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  Version="1.0">
  <TokenID>12345678</TokenID>
  <TriggerNonce>112dsdfwf312asder394jw==</TriggerNonce>
  <SupportedKeyTypes>
    <Algorithm>http://www.rsasecurity.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-
AES</Algorithm>
  </SupportedKeyTypes>
  <SupportedEncryptionAlgorithms>
    <Algorithm>http://www.w3.org/2001/04/xmlenc#rsa-1_5</Algorithm>
    <Algorithm>http://www.w3.org/2001/04/xmlenc#kw-aes128</Algorithm>
    <Algorithm>http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#ct-kip-prf-
aes</Algorithm>
  </SupportedEncryptionAlgorithms>
  <SupportedMACAlgorithms>
    <Algorithm>http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#ct-kip-prf-
aes</Algorithm>
  </SupportedMACAlgorithms>
  <Extensions>
    <Extension xsi:type="ct-kip:TwoPassSupportType">
     <SupportedKeyInitializationMethod>http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-
kip#wrap
```



```
</SupportedKeyInitializationMethod>
    <Payload xsi:type="ds:KeyInfoType">
      <ds:KeyName>Key-001</ds:KeyName>
    </Payload>
   <SupportedKeyInitializationMethod>http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-
kip#transport</SupportedKeyInitializationMethod>
    <Payload xsi:type="ds:KeyInfoType">
      <ds:X509Data>
        <ds:X509Certificate>miib</ds:X509Certificate>
      </ds:X509Data>
    </Payload>
  </Extension>
 </Extensions>
</ClientHello>
```

B.3 Example of a <ServerFinished> message using the key transport profile

In this example, the server responds to the previous request using the key transport profile.

```
<?xml version="1.0" encoding="UTF-8"?>
<ServerFinished
 xmlns:ct-kip="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 Version="1.0" SessionID="4114" Status="Success">
 <TokenID>12345678</TokenID>
 <KeyID>43212093</KeyID>
 <KeyExpiryDate>2009-09-16T03:02:00Z</KeyExpiryDate>
 <ServiceID>Example Enterprise Name</ServiceID>
 <UserID>exampleLoginName</UserID>
 <Extensions>
  <Extension xsi:type="ct-kip:KeyInitializationDataType">
   <KeyInitializationMethod>http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-
kip#transport</KeyInitializationMethod>
    <Payload xsi:type="xenc:EncryptedKeyType"
     Type="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES">
      <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
      <ds:KeyInfo>
        <ds:X509Data>
          <ds:X509Certificate>miib</ds:X509Certificate>
        </ds:X509Data>
      </ds:KeyInfo>
      <xenc:CipherData>
        <xenc:CipherValue>abcd</xenc:CipherValue>
      </xenc:CipherData>
      <xenc:CarriedKeyName>43212093</xenc:CarriedKeyName>
    </Payload>
  </Extension>
  <Extension xsi:type="ct-kip:OTPKeyConfigurationDataType">
    <OTPFormat>Decimal</OTPFormat>
    <OTPLength>6</OTPLength>
    <OTPMode><Time/></OTPMode>
  </Extension>
 </Extensions>
 <Mac MacAlgorithm="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/11/ct-kip#ct-kip-prf-
aes">miidfasde312asder394jw==</Mac>
</ServerFinished>
```


B.4 Example of a <ServerFinished> message using the key wrap profile

In this example, the server responds to the previous request using the key wrap profile.

```
<?xml version="1.0" encoding="UTF-8"?>
<ServerFinished
 xmlns:ct-kip="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc="http://www.w3.org/2001/04/xmlenc#"
 Version="1.0" SessionID="4114" Status="Success">
 <TokenID>12345678</TokenID>
 <KeyID>43212093</KeyID>
 <KeyExpiryDate>2009-09-16T03:02:00Z</KeyExpiryDate>
 <ServiceID>Example Enterprise Name</ServiceID>
 <UserID>exampleLoginName</UserID>
 <Extensions>
  <Extension xsi:type="ct-kip:KeyInitializationDataType">
   <KeyInitializationMethod>http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-
kip#wrap</KeyInitializationMethod>
   <Payload xsi:type="xenc:EncryptedKeyType"
    Type="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES">
    <xenc:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128"/>
    <ds:KeyInfo>
     <ds:KeyName>Key-001</ds:KeyName>
    </ds:KeyInfo>
    <xenc:CipherData>
     <xenc:CipherValue>abcd</xenc:CipherValue>
    </xenc:CipherData>
    <xenc:CarriedKeyName>43212093</xenc:CarriedKeyName>
   </Payload>
  </Extension>
  <Extension xsi:type="ct-kip:OTPKeyConfigurationDataType">
   <OTPFormat>Decimal</OTPFormat>
   <OTPLength>6</OTPLength>
   <OTPMode><Time/></OTPMode>
  </Extension>
 </Extensions>
 <Mac MacAlgorithm="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#ct-kip-prf-
aes">miidfasde312asder394jw==</Mac>
</ServerFinished>
```

B.5 Example of a <ServerFinished> message using the passphrase-based key wrap profile

In this example, the server responds to the previous request using the passphrase-based key wrap profile.

```
<?xml version="1.0" encoding="UTF-8"?>
<ServerFinished
 xmlns:ct-kip="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#"
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
 xmlns:xenc=http://www.w3.org/2001/04/xmlenc#
xmlns:pkcs-5="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#"
 Version="1.0" SessionID="4114" Status="Success">
 <TokenID>12345678</TokenID>
 <KeyID>43212093</KeyID>
 <KeyExpiryDate>2009-09-16T03:02:00Z</KeyExpiryDate>
```



```
 <ServiceID>Example Enterprise Name</ServiceID>
 <UserID>exampleLoginName</UserID>
 <Extensions>
  <Extension xsi:type="ct-kip-two-pass:KeyInitializationDataType">
   <KeyInitializationMethod>http://www.rsasecurity.com/rsalabs/otps/schemas/2006/03/ct-
kip#passphrase-wrap</KeyInitializationMethod>
    <Payload xsi:type="xenc:EncryptedKeyType"
     Type="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/09/otps-wst#SecurID-AES">
      <xenc:EncryptionMethod
       Algorithm="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbes2">
       <pkcs-5:PBES2-params>
        <KeyDerivationFunc
         Algorithm="http://www.rsasecurity.com/rsalabs/pkcs/schemas/pkcs-5#pbkdf2">
         <pkcs-5:PBKDF2-params>
          <Salt>
                    <Specified>32113435</Specified>
          </Salt>
                        <IterationCount>1024</IterationCount>
                    <PRF/>
                        </pkcs-5:PBKDF2-params>
                    </KeyDerivationFunc>
                        <EncryptionScheme
                          Algorithm="http://www.w3.org/2001/04/xmlenc#kw-aes128-cbc">
                        </EncryptionScheme>
                    </pkcs-5:PBES2-params>
      </xenc:EncryptionMethod>
      <ds:KeyInfo>
                    <ds:KeyName>Passphrase1</ds:KeyName>
      </ds:KeyInfo>
      <xenc:CipherData>
                    <xenc:CipherValue>ZWcLvpFoXNHAG+lx3+Rww/Sic+o</xenc:CipherValue>
      </xenc:CipherData>
      <xenc:CarriedKeyName>43212093</xenc:CarriedKeyName>
    </Payload>
   </Extension>
   <Extension xsi:type="ct-kip:OTPKeyConfigurationDataType">
    <OTPFormat>Decimal</OTPFormat>
    <OTPLength>6</OTPLength>
    <OTPMode><Time/></OTPMode>
   </Extension>
  </Extensions>
  <Mac MacAlgorithm="http://www.rsasecurity.com/rsalabs/otps/schemas/2005/12/ct-kip#ct-kip-prf-
aes">miidfasde312asder394jw==</Mac>
</ServerFinished>
```

C. Integration with PKCS #11

C.1 The 2-pass variant

A suggested procedure to perform 2-pass CT-KIP with a cryptographic token through the PKCS #11 interface using the mechanisms defined in [6] 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. A nonce R is generated, e.g. by calling **C_SeedRandom** and **C_GenerateRandom**.

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

b. On the server side,

I. A token key K_{TOKEN} of suitable type is generated, e.g. by calling **C_GenerateKey**. Likewise, a MAC key K_{MAC} is generated, e.g. by calling **C_GenerateKey**. The template for these keys shall allow the keys to be exported but only in wrapped form (i.e. **CKA_SENSITIVE** shall be set to CK_TRUE and **CKA_EXTRACTABLE** shall also be set to CK_TRUE).

II. The server wraps K_{MAC} and K_{TOKEN} with either the client's public key or the shared secret key by calling **C_WrapKey**. If use of the CT-KIP key wrapping algorithm has been negotiated, then the **CKM_KIP_TWO_WRAP [ISSUE: This mechanism not yet defined. An alternative could be to wrap K = K_{MAC} | K_{TOKEN}]** mechanism shall be used to wrap K_{MAC} and K_{TOKEN}. When calling **C_WrapKey**, 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 provided by the CT-KIP client, and the *ulSeedLen* parameter shall indicate the length of R. The *hWrappingKey* parameter in the call to **C_WrapKey** shall be set to refer to the wrapping key.

III. Next, the server needs to calculate a MAC using K_{MAC}. 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_{MAC} shall be the signature key, the *hKey* parameter in the **CK_KIP_PARAMS** structure shall be a handle to the key K_{TOKEN}, 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 the concatenation of the string ID_S and the nonce R, and the *ulDataLen* parameter shall be set to the length of the concatenated string. The desired length of the MAC shall be specified through the *pulSignatureLen* parameter and shall be set to the length of R.

IV. If the server also needs to authenticate its message (due to an existing K_{TOKEN} being replaced), the server calculates a second MAC [**ISSUE: How to transport this second MAC?**]. Again, 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 this call to **C_SignInit**, K_{AUTH} (or the K_{MAC} existing before the protocol run) shall be the signature key, the *hKey* parameter in the **CK_KIP_PARAMS** structure shall be a handle to the key K_{TOKEN}, the *pSeed* parameter of the **CT_KIP_PARAMS** structure shall be set to NULL_PTR, and the *ulSeeidLen* parameter shall be set to zero. In the call to **C_Sign**, the *pData* parameter shall be set to the concatenation of the string ID_S and the nonce R, and the *ulDataLen* parameter shall be set to the length

of concatenated string. The desired length of the MAC shall be specified through the *pulSignatureLen* parameter and shall be set to the length of *R*.

V. The server sends its second message to the client, including the two wrapped keys, the MAC and possibly also the authenticating MAC.

c. On the client side,

I. The client calls **C_UnwrapKey** to receive a handle to K_{TOKEN} and K_{MAC}. [**ISSUE: Maybe call C_DeriveKey twice if K = K_{MAC} | K_{TOKEN} ?]** When calling **C_UnwrapKey**, the *pTemplate* parameter shall be used to set additional key attributes 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** for K_{TOKEN}**.**

II. The MAC is verified in a reciprocal fashion as it was generated by the server. If use of the **CKM_KIP_MAC** mechanism has been negotiated, then in the call to **C_VerifyInit,** the *hKey* parameter in the **CK_KIP_PARAMS** structure shall refer to K_{TOKEN}, 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_{MAC}. In the call to **C_Verify**, *pData* shall be set to the concatenation of the string ID_S and the nonce *R*, and the *ulDataLen* parameter shall be set to the length of the concatenated string, *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. The token shall be constructed to not "commit" to the new K_{TOKEN} or the new K_{MAC} unless the MAC verifies.

III. If an authenticating MAC was received (required if the new K_{TOKEN} will replace an existing key on the token), then it is verified in a similar vein. Again, if the MAC does not verify the protocol session ends with a failure, and the token must be constructed no to "commit" to the new K_{TOKEN} or the new K_{MAC} unless the MAC verifies.

C.2 The 1-pass variant

A suggested procedure to perform 1-pass CT-KIP with a cryptographic token through the PKCS #11 interface using the mechanisms defined in [6] is as follows (see also [1]):

a. On the server side,

I. A token key K_{TOKEN} of suitable type is generated, e.g. by calling **C_GenerateKey**. Likewise, a MAC key K_{MAC} is generated, e.g. by calling **C_GenerateKey**.

IV. The server wraps K_{MAC} and K_{TOKEN} with either the client's public key or the shared secret key by calling **C_WrapKey**. If use of the CT-KIP key wrapping algorithm has been negotiated, then the **CKM_KIP_TWO_WRAP** mechanism [**ISSUE: See above**] shall be used to wrap K_{TOKEN}. When calling **C_WrapKey**, 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 octet-string representation[5] of an integer *I* whose value shall be incremented before each protocol run, and the *ulSeedLen* parameter shall indicate the length of the octet-string representation of *I*. The *hWrappingKey* parameter in the call to **C_WrapKey** shall be set to refer to the wrapping key.

II. For the server's 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_{MAC} shall be the signature key, the *hKey* parameter in the **CK_KIP_PARAMS** structure shall be a handle to the key K_{TOKEN}, 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 the concatenation of the string ID_S and the octet-string representation of the integer *I*, and the *ulDataLen* parameter shall be set to the length of concatenated string. The desired length of the MAC shall be specified through the *pulSignatureLen* parameter as usual, and shall be equal to, or greater than, sixteen (16).

III. If the server also needs to authenticate its message (due to an existing K_{TOKEN} being replaced), the server calculates a second MAC [**ISSUE: How to transfer this in the protocol?**]. If the CT-KIP MAC mechanism is used, the server does this 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} (or the K_{MAC} already existing on the token) shall be the signature key, the *hKey* parameter in the **CK_KIP_PARAMS** structure shall be a handle to the key K_{TOKEN}, 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 the concatenation of the string ID_S and the octet-string representation of the integer I+1 (i.e. *I* shall be incremented before each use), and the *ulDataLen* parameter shall be set to the length of the concatenated string. The desired length of the MAC shall be specified through the *pulSignatureLen* parameter as usual, and shall be equal to, or greater than, sixteen (16).

IV. The server sends its second message to the client, including the MAC and possibly also the authenticating MAC.

b. On the client side,

I. The client calls **C_UnwrapKey** to receive a handle to K_{TOKEN} and K_{MAC}. When calling **C_UnwrapKey**, the *pTemplate* parameter shall be used to set additional key attributes 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** for K_{TOKEN}**.**

[5] The integer-to-octet string conversion shall be made using the **I2OSP** primitive from [2]. There shall be no leading zeros.

II. The MAC is verified in a reciprocal fashion as it was generated by the server. If use of the **CKM_KIP_MAC** mechanism has been negotiated, then in the call to **C_VerifyInit,** the *hKey* parameter in the **CK_KIP_PARAMS** structure shall refer to K_{TOKEN}, 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_{MAC}. In the call to **C_Verify**, *pData* shall be set to the concatenation of the string ID_S and the octet-string representation of the provided value for I[6], and the *ulDataLen* parameter shall be set to the length of the concatenated string, *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. The token shall be constructed to not "commit" to the new K_{TOKEN} or the new K_{MAC} unless the MAC verifies.

III. If an authenticating MAC was received (required if K_{TOKEN} will replace an existing key on the token), it is verified in a similar vein. Again, if the MAC does not verify the protocol session ends with a failure, and the token must be constructed no to "commit" to the new K_{TOKEN} or the new K_{MAC} unless the MAC verifies.

D. Intellectual property considerations

RSA Security has filed one or more patent applications covering inventions described in this document, including, but not limited to, International Application No. PCT/US2004/021846 which has been nationalized in the United States. RSA Security intends to make royalty-free reciprocal licenses available to implementers of the CT-KIP described in this document for RSA Security patent claims that would be necessarily infringed by implementation of such CT-KIP, solely for the purpose of implementing such CT-KIP.

[6] This shall include a check by the token that the provided value of I is larger than any stored value I_{ID_S} for the identified server ID_S . If this is not the case, the verification shall fail. If the verification succeeds, the token shall store the provided value of I as a new I_{ID_S}.

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

E. References

[1] 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.

[2] RSA Laboratories. *PKCS #1: RSA Cryptography Standard*. Version 2.1, June 2002. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf.

[3] RSA Laboratories. *PKCS #5: Password-Based Cryptography Standard*. Version 2.0, March 1999. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-5v2/pkcs5v2-0.pdf.

[4] RSA Laboratories. *XML Schema for PKCS #5 Version* 2.0. PKCS #5 Version 2.0 Amendment 1 (Draft 1), to be published.

[5] 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.

[6] RSA Laboratories. *PKCS #11 Mechanisms for the Cryptographic Token Key Initialization Protocol*. PKCS #11 v2.20 Amendment 2, December 2005. URL: ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-11/v2-20/pkcs-11v2-20a2.pdf.

[7] W3C. *XML-Signature Syntax and Processing*. W3C Recommendation February, 2002. URL: http://www.w3.org/TR/2002/REC-xmldsig-core-20020212/.

[8] W3C. *XML Encryption Syntax and Processing*. W3C Recommendation December, 2002. URL: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/.

[9] National Institute of Standards and Technology (NIST). *FIPS PUB 112: Password Usage*. May 1985. URL: http://www.itl.nist.gov/fipspubs/fip112.htm.

F. 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/

