Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 27, 2007

A. Doherty
M. Nystroem
RSA, The Security Division of EMC

February 23, 2007

Cryptographic Token Key Initialization Protocol (CT-KIP) Web Service Version 1.0 Revision 0 draft-doherty-keyprov-ct-kip-ws-00

Status of this Memo

Copyright Notice

Abstract

This document defines a SOAP-based Web Service interface intended for use by implementor's of the Cryptographic Token Key Initialization Protocol (CT-KIP) to support four-pass (i.e., two round-trip) cryptographic token key initialization. Reader familiarity with CT-KIP is required.

Table of Contents

1. Introduction

1.1. Scope

This document describes a cryptographic token key initialization Web service interface. This interface provides an open and interoperable method of implementing the Cryptographic Token Key Initialization Protocol (CT-KIP) using Simple Object Access Protocol (SOAP) messages. Reader familiarity with [1] is required.

The objectives of this Web service interface are:

- To provide a universal means for CT-KIP clients and Web services to interoperate.
- To encapsulate the CT-KIP protocol using a standard messaging system.
- To extend the utilization of CT-KIP within highly available and distributed services-oriented environments.
- To sustain the security assurances of systems in operation.
- To provide a solution that is easy to administer and scales well.

The Web service interface is intended for general application within computer and communications systems employing connected cryptographic tokens (or software emulations thereof). A Web service built on this interface can be deployed in a highly available and distributed services oriented environment.

1.2. Background

This document is a companion document to [1], which was submitted to the IETF in 2005. CT-KIP is a client-server protocol for initialization and configuration of cryptographic tokens. 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 or wireless device (e.g., a mobile phone or PDA). Such a token offers cryptographic functionality that may be used, for example, to authenticate a user towards some service. The CT-KIP protocol requires neither private-key capabilities in the cryptographic tokens, nor an established public-key infrastructure. Provisioned (or generated) secrets are only available to the server and the cryptographic token itself.

Since submission of [1] to the IETF, implementations have emerged in

the form of software toolkits that create and process PDU messages required for cryptographic token key generation and configuration. These CT-KIP toolkits became core components in certain services offering inline token provisioning. In the context of this document "inline token provisioning" refers to cryptographic token initialization and configuration via CT-KIP. "Initialization" refers to the process of generating and storing a secret, such as a seed for one-time password generation, into the token. "Configuration" refers to metadata associated with the provisioned secret; supplying this information to the token is a critical step in ensuring that it is prepared to perform the activities required by the system.

As interest grows in CT-KIP so does the need for a common interface for transporting PDU messages to Web services that can dynamically provision tokens to many types of cryptographic tokens located anywhere on the Internet. This document intends to meet this need by defining a simple service aimed at providing Web service implementors and cryptographic token vendors with an open and interoperable mechanism for dynamically initializing and configuring connected cryptographic tokens.

1.3. Document organization

The organization of this document is as follows:

- Section 1 is an introduction.
- Section 2 defines some notation used in this document.
- Section 3 defines the Web service interface in detail.
- Section 4 defines a binding of the protocol to transports.
- Section 5 provides security considerations.
- Appendix A defines the Web Services Description Language (WSDL) schema.
- Appendix B gives example messages.
- Appendix C provides general information about the One-Time Password Specifications (OTPS).

2. Acronyms and notation

2.1. Acronyms

CT-KIP	Cryptographic Token Key Initialization Protocol [1]
CT-PA	Cryptographic Token Provisioning Application
OTPS	One-time Password Specifications (Appendix C)
PDU	Protocol Data Unit
SOAP	Simple Object Access Protocol [4], [5]
CT-KIP-WS	CT-KIP Web Service
WS-Security	Web Services Security: SOAP Message Security [6]
WSDL	Web Services Description Language [7]

2.2. Notation

The following typographical convention is used in the body of the text: <XMLElement>.

3. Service overview

3.1. Usage domain

The primary function of the CT-KIP Web service (CT-KIP-WS) is to serve requests from distributed applications for initialization and configuration of cryptographic tokens, using CT-KIP as the basis. The goal of the CT-KIP-WS is to provide a generic framework that allows a broad range of cryptographic tokens and their providers to be supported. Put in context, experience shows that CT-KIP is only one step in the process of provisioning a token to an end-user. The token provisioning process typically involves:

a. A user enrolling for a token with a cryptographic token provisioning application (CT-PA);

b. The CT-PA fulfilling a token to the end-user (a hardware token would normally be shipped or hand-delivered to the end-user, a software token would be downloaded and installed on whatever platform hosts the token, e.g., a PC or mobile device);

c. A user requesting a new credential from the CT-PA, triggering the initialization and configuration of a key on the token; and

d. The CT-PA activating the token credential for a particular use.

Note that steps (c) and (d) can be repeated multiple times depending on how many credentials the token can contain. However, as shown in Figure 1, Step (c) alone engages CT-KIP.

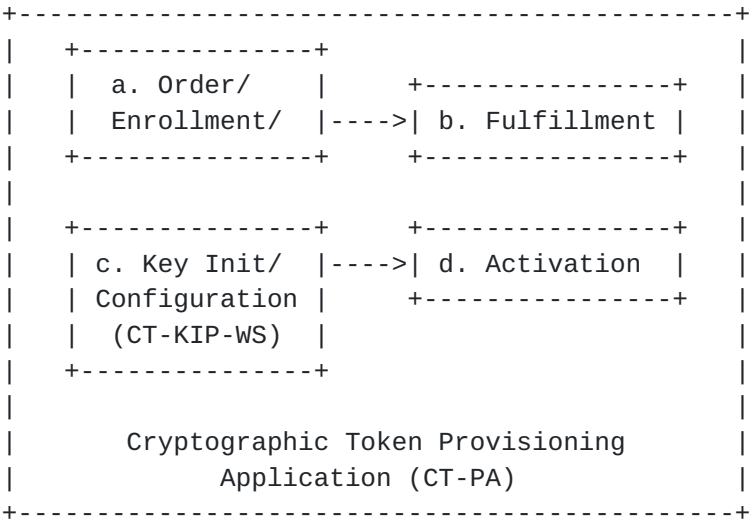

Figure 1. Logical Relationship Between CT-KIP-WS and Sample Components of a CT-PA

Although CT-KIP-WS is logically segregated from other components in a CT-PA, it may be deployed within the same environment, sharing the same data storage as shown in Figure 2.

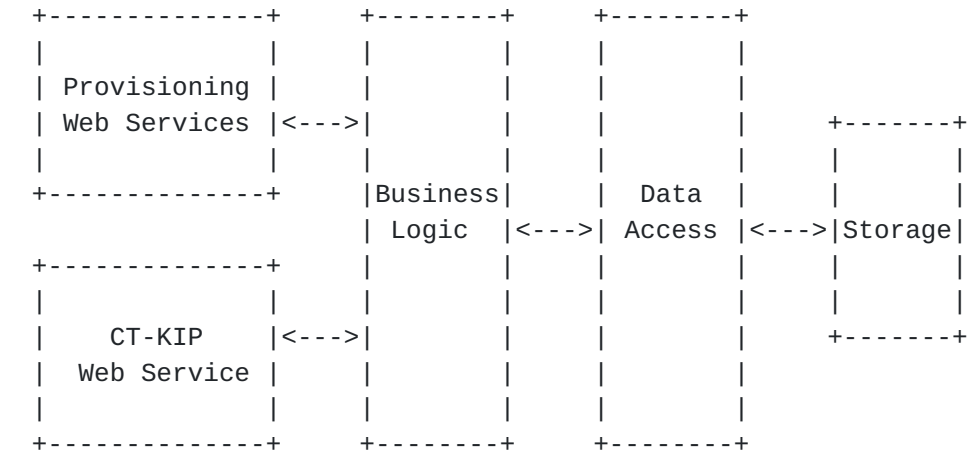

Figure 2. Illustration of CT-KIP-WS Deployed As Part of a CT-PA

In order for CT-KIP-WS to work within the context of a larger application, such as a CT-PA, the token provider may require that the user be authorized before initializing the key, and/or require some token-specific provisioning data to enforce policy and business rules to properly configure the key. To this end, CT-KIP-WS messages carry authorization data and provisioning data in addition to the PDU itself. Authorization data and provisioning data are used by the CT-PA and are opaque to CT-KIP-WS.

This document describes a CT-KIP service in the context of SOAP-based messages, whose definition and binding details are specified by a WSDL schema. This schema and example messages based on it are described in subsequent sections of this document. Besides SOAP, protocol bindings, e.g., HTTP/1.1, are also possible as specified in Section 4.2 of [1].

3.2. Entities

In principle, the CT-KIP-WS API involves a client application that manages a cryptographic token, and a CT-KIP Web service that resides on a Web application server running in a clustered or distributed multi-node environment.

It is assumed that a desktop/laptop or a wireless device (e.g. a mobile phone or a PDA) will host the client application that communicates with the CT-KIP Web service and manages the cryptographic token. Collectively, the cryptographic token and the communicating application form the CT-KIP client.

It is expected that a Web application server will host the Web

service that communicates with the CT-KIP client. As such, the following considerations, which are particular to a service-oriented environment, constrain the Web API design:

- While CT-KIP allows for an optional trigger message from the server to get the protocol started, such a trigger, if required by the system hosting the CT-KIP-WS, should be handled by the Web server that receives a browser-based request from the user for a new (or renewal of a) token. The Web server would POST the trigger message in accordance with the HTTP/1.1 binding as defined in [1]. The browser resident on the device on which the cryptographic token is hosted will receive the trigger and request that the token application send the first CT-KIP-WS message. Note: The rationale for not defining a trigger message in CT-KIP-WS is that cryptographic tokens are ill-suited for hosting Web APIs capable of serving requests.

- In the absence of a trigger message, the CT-KIP protocol is initiated by a CT-KIP-WS ClientHello message. To prevent a Denial of Service attack that could be launched from an attacker's application, which could flood the system with ClientHello requests, an authorization code is defined as a mandatory field in the request schema. It is up to the CT-PA to generate an authorization code before the CT-KIP-WS is first invoked, such as when a user orders a token via an online service. The specification of the code is application-specific and opaque to CT-KIP-WS.

- Cryptographic token providers that implement the CT-KIP-WS may require a contractual relationship to be pre-established with the maker of devices (e.g., PDA, USB memory device, or mobile phone carrier) that house cryptographic tokens. For example, a token provider may charge to provision tokens to certain devices on a subscription basis. Additionally, token providers may need a way to generate activity reports for gathering such things as market analytics. In support of these type of business scenarios, the CT-KIP-WS includes an optional ProvisioningData parameter in each of the SOAP messages, which is opaque to the protocol itself. What information is passed in this parameter is application-specific, and could include information such as the identifier of the token container (e.g., PDA model or mobile phone carrier) hosting the CT-KIP client.

- Since CT-KIP is intended to support provisioning of cryptographic tokens of any type, it was designed to not require HTTPS. In particular, some legacy or just technologically simple mobile phone devices do not have native HTTPS capability. Therefore, it is assumed that messages will be transported over HTTP not HTTPS.

- Although the CT-KIP protocol is stateful, it is expected that a Web service that encapsulates it will be stateless. This is because it is typical for multiple instances of a Web service to be deployed in a clustered and/or multiple node environment. The implication is that each CT-KIP-WS request could be routed by a network load balancer to the currently available instance of the service.

3.3. Principles of operation

The following list provides a high-level description of each message involved in a 4-step CT-KIP session.

1. <CT-KIP-WS-Client-Hello> To initiate the first call to CT-KIP-WS, a user performs some action that is recognized by the client application as a request for a new or renewed token credential. This could be triggered by a co-resident browser that receives a CT-KIP trigger message from a Web server, or by a user interface layered on top of the client application. In any event, the client application will form a CT-KIP request message that includes the following information:

AuthData	Authorization Code that a provisioning application may require in order for CT-KIP to proceed. Specification of AuthData is negotiated between a CT-PA and the client application. If included in the request, CT-KIP-WS forwards AuthData to the CT-PA for processing; AuthData is opaque to CT-KIP-WS.
ProvisioningData	Token data required by the CT-PA for policy and/or business reasons. Specification of ProvisioningData is negotiated between a CT-PA and the client application. If included in the request, CT-KIP-WS forwards ProvisioningData to the CT-PA for processing; ProvisioningData is opaque to CT-KIP-WS.
Request	<ClientHello> PDU in accordance with [1].

2. <CT-KIP-WS-Server-Hello> The CT-KIP-WS forwards the authorization code to the CT-PA, which verifies the code and, if valid, allows the CT-KIP-WS to continue. The CT-KIP-WS then forwards the ProvisioningData to the CT-PA and generates a ServerNonce, returning it in a response message of the following form:

ProvisioningData Application-specific data required for the client application to initialize the token credential.

Response <ServerHello> PDU in accordance with [1]. Note: Although a goal of the Web API is to ensure the PDU is opaque, experience shows that a stateless service that is deployed as multiple instances load balanced across multiple machines, should include a signed ServerNonce in the ServerInfoType protocol extension as a way of ensuring that the CT-KIP client will return it in the next request. For more information on this extension, refer to Section 3.9.2 of [1].

3. <CT-KIP-WS-Client-Nonce> The CT-KIP client generates a random nonce, encrypts it, and sends it to the CT-KIP Web service in a request message that includes the following:

AuthData Authorization Code that a provisioning application may require in order for CT-KIP to proceed. Specification of AuthData is negotiated between a CT-PA and the client application. If included in the request, CT-KIP-WS forwards AuthData to the CT-PA for processing; AuthData is opaque to CT-KIP-WS.

ProvisioningData Token data required by the CT-PA for policy and/or business reasons. Specification of ProvisioningData is negotiated between a CT-PA and the client application. If included in the request, CT-KIP-WS forwards ProvisioningData to the CT-PA for processing; ProvisioningData is opaque to CT-KIP-WS.

Request <ClientNonce> PDU in accordance with [1]. Note that if data were provided in the ServerInfoType protocol extension of the <ServerHello> PDU, the CT-KIP client will return this information in the ServerInfoType protocol extension of the ClientNonce request.

The CT-KIP client uses the ServerNonce and ClientNonce to generate a secret key and loads it into the cryptographic token.

4. <CT-KIP-WS-Server-Finished> The CT-KIP-WS forwards the authorization code to the CT-PA, which verifies the code and, if valid, allows the CT-KIP-WS to continue. The CT-KIP-WS then forwards the provisioning data to the CT-PA, decrypts the ClientNonce, and generates a secret key using the ServerNonce and ClientNonce. Finally, CT-KIP-WS requests token credential configuration (e.g., expiry date) from the CT-PA . The service then returns this information in a response message with the following information:

ProvisioningData	Application-specific data required for the client application to configure the token credential.
Response	<ServerFinished> PDU in accordance with [1]. A protocol extension, such as OTPConfigurationDataType may contain token configuration information. Alternatively, the CT-KIP Web service may pass this information in ProvisioningData.

The CT-KIP client uses the information contained in the ServerFinished response to configure the token credential. Simultaneously, the CT-KIP Web service may call the CT-PA with the credential, requesting the CT-PA to update its data store.

3.4. WSDL schema basics

3.4.1. Introduction

Core parts of the WSDL for CT-KIP-WS, found in Appendix A, are explained in this section. Specific messages are defined in Section 3.5. Examples are found in Appendix B. In cases of disagreement between the WSDL in Appendix A and element snippets in this section, the WSDL takes precedence.

The WSDL format for CT-KIP-WS messages has been designed to facilitate integration with CT-PAs. However, it is possible that the use of application-specific attributes could harm interoperability and therefore use of these attributes should be carefully considered.

XML types defined in this sub-section are not CT-KIP-WS messages; rather they provide building blocks that are used by CT-KIP-WS messages.

3.4.2. Attributes, typing, and referencing

CT-KIP-WS elements rely on parties being able to compare received values with stored values. Unless otherwise noted, all attributes in this document that have the XML Schema "xs:string" type, or a type derived from it, must be compared using an exact binary comparison. In particular, CT-KIP-WS implementations must not depend on case-insensitive string comparisons, normalization or trimming of white space, or conversion of locale-specific formats such as numbers.

Implementations that compare values that are represented using different character encodings MUST use a comparison method that returns the same result as converting both values to the Unicode character encoding, Normalization Form C [2], and then performing an exact binary comparison.

No collation or sorting order for attributes or element values is defined. CT-KIP-WS implementations must therefore not depend on specific sorting orders for values.

3.4.3. Namespaces

Namespaces are defined for the CT-KIP WSDL, for the XML schema used, for SOAP, and for WSDL [7]. CT-KIP-WS namespaces are listed below. The target namespace of the CT-KIP WSDL is implementation-specific, and defined as type "uri" (for a sample implementation, refer to Appendix A). The CT-KIP WSDL creates a schema for XML/SOAP interfaces using [8], and relies on the XML Schema [3] based attribute typing model.

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:impl="uri"
  targetNamespace="uri">
```

3.5. Message elements

3.5.1. Request/response model

The general model adopted by CT-KIP-WS is one of clients performing protocol operations against servers. In this model, a client issues a CT-KIP-WS request describing the operation to be performed at the service end-point. A complete CT-KIP-WS transaction involves transfer of one message between the client and server. A transaction is initiated with a request message sent by the client, and all accepted requests receive corresponding responses from the CT-KIP-WS. In all cases, request and response messages will contain a PDU

representing one of the passes in the CT-KIP protocol. The PDU format defines an XML-based representation that carries identification of the protocol message and corresponding data in accordance with Section 3.8 of [1].

Request and response messages are defined within an XML schema. Based on experience, implementation should treat messages as XML streams. Note that use of these messages in the context of SOAP-based web services is contemplated, and SOAP binding specifics are discussed in Section 4. CT-KIP-WS request/response message elements are described in the sub-sections below.

3.5.2. Element <ClientRequest>

The <ClientRequest> element is used to make a CT-KIP request.

```
<element name="ClientRequest">
  <complexType>
    <sequence>
      <element name="AuthData" type="xs:string"/>
      <element name="ProvisioningData" type="xs:string"/>
      <element name="Request" type="xs:string"/>
    </sequence>
  </complexType>
</element>
```

The components of this message have the following meaning:

- <AuthData>: A mandatory element of type "xs:string". This string is opaque to the protocol layer. Its format specification is determined by the CT-PA and is outside the scope of CT-KIP-WS. See the description of AuthData in Section 3.3, points 1 and 3.

- <ProvisioningData>: A mandatory element of type "xs:string". This string is opaque to the protocol layer. Its format specification is determined by the CT-PA and is outside the scope of CT-KIP-WS. See the description of ProvisioningData in Section 3.3, points 1 and 3.

- <Request>: A mandatory element of type "xs:string" that represents the PDU. Its format specification is defined in Section 3.8 of [1]. See the description of PDU in Section 3.3, points 1 and 3.

3.5.3. Element <ServerResponse>

The <ServerResponse> element is used to convey the results of a request specified in a <ClientRequest> element.

```
<element name="ServerResponse">
  <complexType>
    <sequence>
      <element name="AuthData" type="xs:string"/>
      <element name="ProvisioningData" type="xs:string"/>
      <element name="Response" type="xs:string"/>
    </sequence>
  </complexType>
</element>
```

The components of this message have the following meaning:

- <AuthData>: A mandatory element of type "xs:string". This string is opaque to the protocol layer. Its format specification is determined by the CT-PA and is outside the scope of CT-KIP-WS. See the description of AuthData in Section 3.3, points 2 and 4.

- <ProvisioningData>: A mandatory element of type "xs:string". This string is opaque to the protocol layer. Its format specification is determined by the CT-PA and is outside the scope of CT-KIP-WS. See the description of ProvisioningData in Section 3.3, points 2 and 4.

- <Response>: A mandatory element of type "xs:string" that represents the PDU. ts format specification is defined in Section 3.8 of [1]. See the description of PDU in Section 3.3, points 2 and 4.

3.6. Operation

A single operation is supported by CT-KIP-WS that takes a CT-KIP-WS request message as an inbound XML stream and returns a CT-KIP response message as an outbound XML stream. The Operations section of the CT-KIP WSDL is described below. Messages are grouped as a pair of request and response elements. A port type identifies to whom the message is sent and how, and defines a single operation for handling the requests and responses.

3.6.1. Message <processClientRequest>

The <processClientRequest> message is used to transfer <ClientRequest> elements from the client to the CT-KIP-WS.

```
<message name="processClientRequest">
  <part element="impl:ClientRequest" name="ClientRequest"/>
</message>
```

3.6.2. Message <processServerResponse>

The <processServerResponse> message is used to transfer <ServerResponse> elements from CT-KIP-WS to the client.

```
<message name="processServerResponse">
  <part element="impl:ServerResponse" name="ServerResponse"/>
</message>
```

3.6.3. PortType <CtkipMessageProcessor>

Port type <CtkipMessageProcessor> identifies itself as the destination for CT-KIP-WS messages, and contains a single operation representing how <processClientRequest> and <processServerResponse> messages are to be handled.

```
<portType name="CtkipMessageProcessor">
  <operation name="processClientMessage">
    <documentation> Processes a ct-kip client hello or
      nonce message and returns a ct-kip response message.
    </documentation>
    <input name="request" message="impl:processClientRequest"/>
    <output name="response" message="impl:processServerResponse"/>
  </operation>
</portType>
```

3.7. Protocol binding specification

The binding section of the CT-KIP WSDL describes how the operations are performed by binding together the <CtkipMessageProcessor> port, SOAP/HTTP transport, and the <processClientMessage> operation. Note that the protocol binding is defined as SOAP with transport over HTTP. Refer to Section 4.1 for more discussion. Also note that, as specified in Section 3.4.3, the target namespace of the CT-KIP WSDL is implementation-specific, and defined below as type "uri" (for a sample implementation, refer to Appendix A).

```
   <binding name="CtkipMessageProcessorBinding"
     type="impl:CtkipMessageProcessor">
     <soap:binding style="document"
       transport="http://schemas.xmlsoap.org/soap/http"/>
       <operation name="processClientMessage">
         <soap:operation soapAction="processClientMessage"
           style="document"/>
           <input>
             <soap:body use="literal"
               namespace="uri"/>
           </input>
           <output>
             <soap:body use="literal"
               namespace="uri"/>
           </output>
       </operation>
   </binding>
```

4. Service bindings

4.1. SOAP binding

This section discusses aspects of carrying CT-KIP-WS messages within SOAP. Use of SOAP 1.2 [4], [5] SHOULD be agreed between client and CT-KIP-WS parties. Similarly, client and server parties may agree on various choices of bindings from SOAP to protocol carriers; however, use with HTTP/1.1 [9] is contemplated.

4.1.1. Operational model

CT-KIP-WS requests and responses MUST be carried with the bodies of SOAP messages, not within the SOAP message envelopes or as header data. This document does not specify other constraints on the use and processing of SOAP envelopes and headers by CT-KIP-WS requesters and responders. Requesters and responders MUST include no more than one CT-KIP-WS request or response within a single SOAP message body, and must not include other data within a SOAP message body that carries a CT-KIP-WS request or response. Within the SOAP message body, a CT-KIP-WS message must be represented as unencoded content.

In normal operation, CT-KIP-WS transactions are represented as request-response pairs; all transactions require only a single message pair. If the recipient of a CT-KIP-WS message is unable to process the message at the CT-KIP-WS level, that recipient MUST generate a SOAP fault to report this error. On the other hand, failures within the scope of CT-KIP processing MUST be reported using facilities within the CT-KIP protocol and MUST NOT generate SOAP faults.

4.1.2. Use of SOAP over HTTP

CT-KIP-WS requesters and responders SHOULD set Cache-Control and Pragma header fields to direct HTTP intermediaries not to cache CT-KIP-WS messages. In particular,

- Requesters SHOULD:
  - Include a Cache-Control header field set to "no-cache", "no-store".
  - Include a Pragma header set to "no-cache".
- Responders SHOULD:
  - Include a Cache-Control header field set to "no-cache, no-must-revalidate, private".

* Include a Pragma header set to "no-cache".

* NOT include a validator, such as Last-Modified or eTag header.

If a CT-KIP-WS responder employing the SOAP/HTTP1.1 binding refuses to engage in a transaction initiated by a requester, the responder SHOULD return a "403 (Forbidden)" HTTP response. If SOAP-level processing errors occur, below the CT-KIP-WS layer, the responder MUST return a "500 (Internal Server Error)" response, including a SOAP message with a SOAP fault element. Errors at the CT-KIP-WS layer are accompanied by "200 (OK)" at the HTTP layer, with the specifics reflected within the CT-KIP-WS response.

4.2. Security bindings

4.2.1. Security service requirements

This section describes candidate requirements that may arise for security services to protect CT-KIP-WS messages. Specific requirements may vary for different operational environments. Further, required services may be achieved using different methods in different environments. In general, the following types of approaches can be applicable:

o Implicit protection, where the path between the CT-KIP-WS requester and responder is confined within a secure environment.

o External protection, where protocol methods such as SSL/TLS or WS-Security are applied to protect CT-KIP-WS messages.

o Native protection, where security facilities defined within CT-KIP-WS are applied to protect CT-KIP-WS data.

4.2.2. Implicit protection of the CT-KIP-WS

If the endpoints of the CT-KIP-WS exchange is confined within a secure environment, then external protection is not required. However, native protection should be relied upon to protect the CT-KIP PDUs from an attacker operating on the inside of the environment Section 4.2.4. Additional protections, such as XML digital signature, SHOULD be applied to the AuthData and ProvisioningData to offer non-repudiation.

4.2.3. External protection of the CT-KIP-WS

If a CT-KIP-WS exchange will be transmitted over an unsecure network, e.g., between two endpoints that are connected via the Internet, then external protections such as those described below MUST be used.

4.2.3.1. SSL/TLS

SSL or TLS [10] can be used to provide authentication, confidentiality, and integrity for CT-KIP-WS transactions, though cannot offer non-repudiation. This is a convenient and effective approach for environments where the span of the CT-KIP-WS protocol between requester and responder can correspond with a single SSL/TLS hop. Client and server systems supporting CT-KIP-WS SHOULD support this method, in order to provide a common basis for secure interoperability.

- Servers MUST authenticate themselves to clients with server-side certificates, and client-side certificates should also be used in order to achieve bidirectional authentication between client and server. Client-side certificates MUST be supported for operation in environments where authentication of CT-KIP-WS requesters is a security policy requirement. Version 3 X.509 certificates MUST be used.

- TLS implementations MUST implement the TLS_RSA_WITH_3DES_EDE_CBC_SHA ciphersuite. TLS implementations SHOULD implement the TLS_RSA_WITH_AES_128_CBC_SHA ciphersuite.

- FIPS TLS implementations MUST implement the TLS_RSA_FIPS_WITH_3DES_EDE_CBC_SHA ciphersuite. FIPS TLS implementations SHOULD implement the TLS_RSA_FIPS_AES_128_CBC_SHA ciphersuite.

- SSL implementations MUST implement the SSL_RSA_WITH_3DES_EDE_CBC_SHA ciphersuite.

- FIPS SSL implementations MUST implement the SSL_RSA_FIPS WITH_3DES_EDE_CBC_SHA ciphersuite.

4.2.3.2. WS-Security

WS-Security [6] can be applied to provide authentication, confidentiality, and integrity services for CT-KIP-WS transactions, and to perform and process signatures in support of non-repudiation. This is an effective approach for environments where the CT-KIP-WS is implemented as a multi-hop Web service, or where message integrity is required. This approach can be applied independently of whether or not active intermediaries intervene on the path between requesters and responders.

4.2.4. Native protection of the CT-KIP-WS

CT-KIP-WS relies on built-in (i.e., native) security features of the CT-KIP protocol to protect the CT-KIP PDUs while in transit. For more information, see Section 5 of [1].

Note that the native CT-KIP security features do not protect what is transmitted outside of the CT-KIP PDU itself, e.g., the AuthData and ProvisioningData parameters. As discussed in previous Section 4.2.3, it is possible for authentication and/or confidentiality services to be applied selectively; the policies that CT-KIP-WS clients and servers use in determining when security services are to be applied are not defined in this document.

5. Security considerations

As discussed in Section 4.2 of this document, CT-KIP-WS deployers MUST identify the security requirements and operational constraints of their environments, so that appropriate security can be ensured for the CT-KIP-WS protocol. As that section notes, protection may be obtained implicitly from the environment, through external protocol mechansims, and/or through native facilities defined in CT-KIP; appropriate choices will vary for different operational contexts.

In order to protect CT-KIP-WS transactions through cryptographic means, some form of trusted infrastructure (e.g., a PKI, Kerberos, and/or local configuration) must be applied to establish keys among communicating parties. Provision and management of such an infrastructure is outside the scope of CT-KIP-WS.

CT-KIP-WS implementations MUST maintain the confidentiality and integrity of sensitive data items that they generate and process. CT-KIP-WS implementations MUST also determine the authenticity and integrity of received messages before taking security-relevant actions based on those messages. When signature-based protection is used, this implies that signatures MUST be successfully verified before sensitive actions are performed. Following the authentication process, it is also necessary to determine whether an authenticated peer identity represents an entity that is authorized (per local policy) to perform CT-KIP-WS.

In addition to considerations identified in this section, Security Considerations discussed in [1] also apply to CT-KIP-WS.

6. IANA considerations

IANA has no action with respect to this document.

7. Intellectual property considerations

RSA and RSA Security are registered trademarks or trademarks of RSA Security Inc. in the United States and/or other countries. The names of other products and services mentioned may be the trademarks of their respective owners.

8. References

8.1. Normative references

[1] Nystroem, M., "Cryptographic Token Key Initialization Protocol", RFC 4758, November 2006, <http://www.ietf.org/rfc/rfc4758.txt>.

[2] Davis, M. and M. Drst, "Unicode Normalization Forms", March 2001, <http://www.unicode.org/unicode/reports/tr15/tr15-21.html>.

[3] Thompson, Beech, D., Maloney, M., and N. Mendelsohn, "W3C XML Schema", W3C Recommendation xmlschema-1, October 2004, <http://www.w3.org/TR/xmlschema-1/>.

8.2. Informative references

[4] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., and H. Nielsen, "SOAP Version 1.2 Part 1: Messaging Framework", W3C Recommendation, June 2003, <http://www.w3.org/TR/soap12-part1/>.

[5] Gudgin, M., Hadley, M., Mendelsohn, N., Moreau, J., and H. Nielsen, "SOAP Version 1.2 Part 2: Adjuncts", W3C Recommendation, June 2003, <http://www.w3.org/TR/soap12-part2/>.

[6] OASIS, "Web Services Security: SOAP Message Security 1.0", January 2004.

[7] IBM and Microsoft Corporation, "Web Service Description Language Schema", 2005, <http://www.schemas.xmlsoap.org/wsdl/>.

[8] IBM and Microsoft Corporation, "XML SOAP WSDL", 2003, <http://schemas.xmlsoap.org/wsdl/soap/2003-02-11.xsd >.

[9] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[10] Dierks, T., "The TLS Protocol Version 1.0", RFC 2246, January 1999, <http://www.ietf.org/rfc/rfc2246.txt>.

Appendix A. Example CT-KIP-WS WSDL schema

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:impl="http://ctkipservice.rsasecurity.com"
  targetNamespace="http://ctkipservice.rsasecurity.com">
  <types>
    <schema elementFormDefault="qualified"
      targetNamespace="http://ctkipservice.rsasecurity.com"
      xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="ClientRequest">
        <complexType>
          <sequence>
            <element name="AuthData" type="xs:string"/>
            <element name="ProvisioningData"
              type="xs:string"/>
            <element name="Request" type="xs:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="ServerResponse">
        <complexType>
          <sequence>
            <element name="AuthData" type="xs:string"/>
            <element name="ProvisioningData"
              type="xs:string"/>
            <element name="Response" type="xs:string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </types>
  <message name="processClientRequest">
    <part element="impl:ClientRequest" name="ClientRequest"/>
  </message>
  <message name="processServerResponse">
    <part element="impl:ServerResponse" name="ServerResponse"/>
  </message>
  <portType name="CtkipMessageProcessor">
    <operation name="processClientMessage">
      <documentation> Processes a ct-kip client hello or
        nonce message and returns a ct-kip response message.
      </documentation>
      <input name="request" message="impl:processClientRequest"/>
      <output name="response"
        message="impl:processServerResponse"/>
```

```
      </operation>
    </portType>
    <binding name="CtkipMessageProcessorBinding"
      type="impl:CtkipMessageProcessor">
      <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http"/>
      <operation name="processClientMessage">
        <soap:operation soapAction="processClientMessage"
          style="document"/>
        <input>
          <soap:body use="literal"
            namespace="http://ctkipservice.rsasecurity.com"/>
        </input>
        <output>
          <soap:body use="literal"
            namespace="http://ctkipservice.rsasecurity.com"/>
        </output>
      </operation>
    </binding>
    <service name="CtkipService">
      <port name="CtkipService"
        binding="impl:CtkipMessageProcessorBinding">
        <soap:address
          location=
          "http://localhost:8080/axis/services/CtkipService"/>
      </port>
    </service>
  </definitions>
```

Appendix B. Examples of CT-KIP SOAP messages

All examples are syntactically correct. Input and output parameters are fictitious however. The examples illustrate a complete CT-KIP exchange, starting with the ClientHello message from the client.

B.1. Example of a <CT-KIP-WS-Client-Hello> message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  Version="1.0">
  <soapenv:Body>
    <ClientRequest xmlns="http://ctkipservice.rsasecurity.com">
      <AuthData>pleaseGiveThisGuyAToken</AuthData>
      <ProvisioningData>HamAndEggs</ProvisioningData>
      <Request>PENsaWVudEhlbGxvIHhtbG5zPSJodHRwOi8vd3d3LnJzYXNlY3
      VyaXR5LmNvbS9yc2FsYWJzL290cHMvc2NoZW1hcy8yMDA1LzExL2N0LWtpc
      CMiIHhtbG5zOnhzZD0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hl
      bWEiIHhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2h
      lbWEtaW5zdGFuY2UiIFZlcnNpb249IjEuMCI+IDxTdXBwb3J0ZWRLZXlUeX
      BlcyB4bWxucz0iIj4gPEFsZ29yaXRobSB4c2k6dHlwZT0ieHNkOmFueVVSS
      SI+aHR0cDovL3d3dy5yc2FzZWN1cml0eS5jb20vcnNhbGFicy9vdHBzL3Nj
      aGVtYXMvMjAwNS8wOS9vdHBzLXdzdCNTZWN1cklELUFFUzwvQWxnb3JpdGh
      tPiA8L1N1cHBvcnRlZEtleVR5cGVzPiA8U3VwcG9ydGVkRW5jcnlwdGlvbk
      FsZ29yaXRobXMgeG1sbnM9IiI+IDxBbGdvcml0aG0geHNpOnR5cGU9InhzZ
      DphbnlVUkkiPmh0dHA6Ly93d3cudzMub3JnLzIwMDEvMDQveG1sZW5jI3Jz
      YS0xXzU8L0FsZ29yaXRobT4gPC9TdXBwb3J0ZWRFbmNyeXB0aW9uQWxnb3J
      pdGhtcz4gPFN1cHBvcnRlZE1BQ0FsZ29yaXRobXMgeG1sbnM9IiI+IDxBbG
      dvcml0aG0geHNpOnR5cGU9InhzZDphbnlVUkkiPmh0dHA6Ly93d3cucnNhc
      2VjdXJpdHkuY29tL3JzYWxhYnMvb3Rwcy9zY2hlbWFzLzIwMDUvMTEvY3Qt
      a2lwI2N0LWtpcC1wcmYtYWVzPC9BbGdvcml0aG0+IDwvU3VwcG9ydGVkTUF
      DQWxnb3JpdGhtcz4gPC9DbGllbnRIZWxsbz4g</Request>
    </ClientRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

B.2. Example of a <CT-KIP-Server-Hello> message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ServerResponse xmlns="http://ctkipservice.rsasecurity.com">
```

```
<AuthData>phase2authorized</AuthData>
<ProvisioningData>HamAndEggs</ProvisioningData>
<Response>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiP
z4NCjxTZXJ2ZXJIZWxsbyBTZXNzaW9uSUQ9IlVZYWlzbVQ4bUNMcEpaM0J6
UGhXMFE9PSIgU3RhdHVzPSJDb250aW51ZSIgVmVyc2lvbj0iMS4wIiB4bWx
ucz0iaHR0cDovL3d3dy5yc2FzZWN1cml0eS5jb20vcnNhbGFicy9vdHBzL3
NjaGVtYXMvMjAwNS8xMi9jdC1raXAjIiB4bWxuczpkcz0iaHR0cDovL3d3d
y53My5vcmcvMjAwMC8wOS94bWxkc2lnIyIgeG1sbnM6eHNpPSJodHRwOi8v
d3d3LnczLm9yZy8yMDAxL1hNTFNjaGVtYS1pbnN0YW5jZSI+PEtleVR5cGU
geG1sbnM9IiI+aHR0cDovL3d3dy5yc2FzZWN1cml0eS5jb20vcnNhbGFicy
9vdHBzL3NjaGVtYXMvMjAwNS8wOS9vdHBzLXdzdCNTZWN1cklELUFFUzwvS
2V5VHlwZT48RW5jcnlwdGlvbkFsZ29yaXRobSB4bWxucz0iIj5odHRwOi8v
d3d3LnczLm9yZy8yMDAxLzA0L3htbGVuYyNyc2EtMV81PC9FbmNyeXB0aW9
uQWxnb3JpdGhtPjxFbmNyeXB0aW9uS2V5IHhtbG5zPSIiIHhtbG5zOmRzPS
JodHRwOi8vd3d3LnczLm9yZy8yMDAwLzA5L3htbGRzaWcjIiB4bWxuczp4c
2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3RhbmNl
Ij48dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3RhbmNlIj
48ZHM6WDUwOURhdGEgeG1sbnM9Imh0dHA6Ly93d3cucnNhc2VjdXJpdHk
uY29tL3JzYWxhYnMvb3Rwcy9zY2hlbWFzLzIwMDUvMTIvY3Qta2lwIyIgeG
1sbnM6ZHM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHNpZyMiI
HhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hlbWEt
aW5zdGFuY2UiPjxkczpYNTA5Q2VydGlmaWNhdGU+TUlJQ2VUQ0NBV0dnQXd
JQkFnSVFNME5GTlRVNE5UTXpNRGxHT0RFek1EQU5CZ2txaGtpRzl3MEJBUV
FGQURCQk1UOHdQUVlEVlFRRA0KRXpaVFpXTjFjbWwwZVNCRWVXNWhiV2xqY
3lCVVpXTm9ibTlzYjJkcFpYTXNJRWx1WXk0Z1VISnBiV0Z5ZVNCRFFTQlNi
MjkwSURFdw0KSGhjTk1ESXdOVEUzTVRreU1UVTFXaGNOTWpJd05URXlNVGt
5TVRVMVdqQTBNVEl3TUFZRFZRUURFeWxUWldOMWNtDBlU0JFZVc1aA0KYl
dsamN5QlVaV05vYm05c2IyZHBaWE1nUVVORkwxTmxjblpsY2pDQm56QU5CZ
2txaGtpRzl3MEJBUUVGQUFPQmpRQXdnWWtDZ1lFQQ0KMW5wMURJZjNIT0hB
SzJhaGNSelpDSnNxSUMxUU1FcXRzZGFuS1NFbjVDR3RMQ2RMdjlMYkxVWW8
2Y1F4S1NKdHd2aWdwZURnQkFiLw0KVVljVU5YeS83ZFk3ckE1V3BZbHNhQT
loNUM5cXpQTUJIeFZHU0llNWs2MXVVYkF3ZEZoQ01mTGg3NzZ3Ui8vVlo3Y
3V5cG81ZDNjQw0KYnZnSEd3cXc0WnVFQ2JLdk9OTUNBd0VBQVRBTkJna3Fo
a2lHOXcwQkFRUUZBQU9DQVFFQXE4TU1KczFTY3p3cGZjWnFuOWxvTSsyUg0
KaEZtTjFJWmlYeWV2ejFWdkdEOUdVcmxMYWxtL0V0OTg5elIvZFZoY2lHWG
1BQXhZblYvTW9abXNoalhvem1KZ2pSbWZxcUhMUzQ2VQ0KSjluTFoyQnVFV
mNySG5uNmY5bWVJamVNV20rRHZoKzhWaTlLSk9Mb3pZYkRvYVVNbSs1Rjd5
d0tzVXVCUFJTSjF5a0dKRzZkT0NCWg0KbEpHbU0za2JaNTRsUkFLMlRZY3U
ySk0yMWk3QktkZUU5eEl0eWFiSnpFazNRQ3NYMGVyWTdoM1YvL29rSWZLV0
xoOExpZW9XYlY0Kw0KVnRyUUVvaVV3eXFkWXN3d2dNT3lSaUt1R1RrazNEa
Ehkb3FoRzhTcUhTeGtQdG80MmhFbnBPeDlqMnJxY3NPV29zdk55Zm05bndr
cQ0KSmZodUpDbEx3T3p3NVE9PTwvZHM6WDUwOUNlcnRpZmljYXRlPjwvZHM
6WDUwOURhdGE+PC9FbmNyeXB0aW9uS2V5PjxQYXlsb2FkIHhtbG5zPSIiIH
htbG5zOmRzPSJodHRwOi8vd3d3LnczLm9yZy8yMDAwLzA5L3htbGRzaWcjI
iB4bWxuczp4c2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1h
LWluc3RhbmNlIj48Tm9uY2UgeG1sbnM9IiI+M1hremlRWHlsdHd2YVB5bm5
CMkFqQT09PC9Ob25jZT48L1BheWxvYWQ+PEV4dGVuc2lvbnMgeG1sbnM9Ii
IgeG1sbnM6ZHM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHNpZ
yMiIHhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hl
```

bWEtaW5zdGFuY2UiPjxFeHRlbnNpb24geG1sbnM9IiIgeG1sbnM6Y3Qta2l
wPSJodHRwOi8vd3d3LnJzYXNlY3VyaXR5LmNvbS9yc2FsYWJzL290cHMvc2
NoZW1hcy8yMDA1LzEyL2N0LWtpcCMiIHhtbG5zOmRzPSJodHRwOi8vd3d3L
nczLm9yZy8yMDAwLzA5L3htbGRzaWcjIiB4bWxuczp4c2k9Imh0dHA6Ly93
d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3RhbmNlIj48RGF0YT4zWGt
6aVFYeWx0d3ZhUHlubkIyQWpBPT08L0RhdGE+PC9FeHRlbnNpb24+PC9FeH
RlbnNpb25zPjxNYWNBbGdvcml0aG0geG1sbnM9IiI+aHR0cDovL3d3dy5yc
2FzZWN1cml0eS5jb20vcnNhbGFicy9vdHBzL3NjaGVtYXMvMjAwNS8xMi9j
dC1raXAjY3Qta2lwLXByZi1hZXM8L01hY0FsZ29yaXRobT48L1Nlc
nZlckhlbGxvPg==</Response>
      </ServerResponse>
    </soapenv:Body>
  </soapenv:Envelope>

B.3. Example of a <CT-KIP-WS-Client-Nonce> message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  Version="1.0">
  <soapenv:Body>
    <ClientRequest xmlns="http://ctkipservice.rsasecurity.com">
      <AuthData>phase2authorized</AuthData>
      <ProvisioningData>HamAndEggs</ProvisioningData>
      <Request>PENsaWVudE5vbmNlIHhtbG5zPSJodHRwOi8vd3d3LnJzYXNlY3
      VyaXR5LmNvbS9yc2FsYWJzL290cHMvc2NoZW1hcy8yMDA1LzExL2N0LWtpc
      CMiIHhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hl
      bWEtaW5zdGFuY2UiIFZlcnNpb249IjEuMCIgU2Vzc2lvbklEPSJVWWFpc21
      UOG1DTHBKWjNCelBoVzBRPT0iPjxFbmNyeXB0ZWROb25jZSB4bWxucz0iIj
      5LNlBNWDEreUVmc1FncklIR0N4ZEZmSGRqbVhvc1FWRmxucXJ3VU8ycjJuN
      mMyNXovU2pwMDYrRTB2SFBITXRzclBFUkhyd0VkQmRrUGpSbzlYM2o5TmQ0
      ZWRhOW5mNG5jakI1L2Q2L0xWWEFvVUd3ZTE3UDhSWFRGWXZxdnI5cW9ESlh
      ydkJHKzZTbUxMeEk4d1QzZEJScSszbUJUdXRzcS9USGFHb1FZUkE9PC9Fbm
      NyeXB0ZWROb25jZT48RXh0ZW5zaW9ucyB4bWxucz0iIiB4bWxuczpkcz0ia
      HR0cDovL3d3dy53My5vcmcvMjAwMC8wOS94bWxkc2lnIyIgeG1sbnM6eHNp
      PSJodHRwOi8vd3d3LnczLm9yZy8yMDAxL1hNTFNjaGVtYS1pbnN0YW5jZSI
      gPjxFeHRlbnNpb24geG1sbnM9IiIgeG1sbnM6Y3Qta2lwPSJodHRwOi8vd3
      d3LnJzYXNlY3VyaXR5LmNvbS9yc2FsYWJzL290cHMvc2NoZW1hcy8yMDA1L
      zEyL2N0LWtpcCMiIHhtbG5zOmRzPSJodHRwOi8vd3d3LnczLm9yZy8yMDAw
      LzA5L3htbGRzaWcjIiB4bWxuczp4c2k9Imh0dHA6Ly93d3cudzMub3JnLzI
      wMDEvWE1MU2NoZW1hLWluc3RhbmNlIj48RGF0YT4zWGt6aVFYeWx0d3ZhUH
      lubkIyQWpBPT08L0RhdGE+PC9FeHRlbnNpb24+PC9FeHRlbnNpb25zPjwvQ
      2xpZW50Tm9uY2U+</Request>
    </ClientRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

B.4. Example of a <CT-KIP-WS-Server-Finished> message

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <soapenv:Body>
    <ServerResponse xmlns="http://ctkipservice.rsasecurity.com">
      <AuthData>phase2authorized</AuthData>
      <ProvisioningData>HamAndEggs</ProvisioningData>
      <Response>PD94bWwgdmVyc2lvbj0iMS4wIiBlbmNvZGluZz0iVVRGLTgiP
      z4NCjxTZXJ2ZXJGaW5pc2hlZCBTZXNzaW9uSUQ9IlVZYWlzbVQ4bUNMcEpa
      M0J6UGhXMFE9PSIgU3RhdHVzPSJTdWNjZXNzIiB4bWxucz0iaHR0cDovL3d
      3dy5yc2FzZWN1cml0eS5jb20vcnNhbGFicy9vdHBzL3NjaGVtYXMvMjAwNS
      8xMi9jdC1raXAjIiB4bWxuczpkcz0iaHR0cDovL3d3dy53My5vcmcvMjAwM
      C8wOS94bWxkc2lnIyIgeG1sbnM6eHNpPSJodHRwOi8vd3d3LnczLm9yZy8y
      MDAxL1hNTFNjaGVtYS1pbnN0YW5jZSI+PFRva2VuSUQgeG1sbnM9IiI+QUE
      9PTwvVG9rZW5JRD48S2V5SUQgeG1sbnM9IiI+WmtKbDBaRWJicW9RdmNVT1
      MvV0Jtdz09PC9LZXlJRD48S2V5RXhwaXJ5RGF0ZSB4bWxucz0iIj4yMDA3L
      TAzLTMxVDEyOjE3OjQzLjE1MS0wNTowMDwvS2V5RXhwaXJ5RGF0ZT48U2Vy
      dmljZUlEIHhtbG5zPSIiPlRlc3Rfc2VydmVyPC9TZXJ2aWNlSUQ+PFVzZXJ
      JRCB4bWxucz0iIi+PEV4dGVuc2lvbnMgeG1sbnM9IiIgeG1sbnM6ZHM9Imh
      0dHA6Ly93d3cudzMub3JnLzIwMDAvMDkveG1sZHNpZyMiIHhtbG5zOnhzaT
      0iaHR0cDovL3d3dy53My5vcmcvMjAwMS9YTUxTY2hlbWEtaW5zdGFuY2UiP
      jxFeHRlbnNpb24gQ3JpdGljYWw9InRydWUiIHhtbG5zPSIiIHhtbG5zOmRz
      PSJodHRwOi8vd3d3LnczLm9yZy8yMDAwLzA5L3htbGRzaWcjIiB4bWxuczp
      4c2k9Imh0dHA6Ly93d3cudzMub3JnLzIwMDEvWE1MU2NoZW1hLWluc3Rhbm
      NlIj48T1RQRm9ybWF0PkRlY2ltYWw8L09UUEZvcm1hdD48T1RQTGVuZ3RoP
      jg8L09UUExlbmd0aD48T1RQTW9kZSB4bWxucz0iaHR0cDovL3d3dy5yc2Fz
      ZWN1cml0eS5jb20vcnNhbGFicy9vdHBzL3NjaGVtYXMvMjAwNS8xMi9jdC1
      raXAjIiB4bWxuczpkcz0iaHR0cDovL3d3dy53My5vcmcvMjAwMC8wOS94bW
      xkc2lnIyIgeG1sbnM6eHNpPSJodHRwOi8vd3d3LnczLm9yZy8yMDAxL1hNT
      FNjaGVtYS1pbnN0YW5jZSI+PFRpbWUgVGltZUludGVydmFsPSI2MCIgeG1s
      bnM9Imh0dHA6Ly93d3cucnNhc2VjdXJpdHkuY29tL3JzYWxhYnMvb3Rwcy9
      zY2hlbWFzLzIwMDUvMTIvY3Qta2lwIyIgeG1sbnM6ZHM9Imh0dHA6Ly93d3
      cudzMub3JnLzIwMDAvMDkveG1sZHNpZyMiIHhtbG5zOnhzaT0iaHR0cDovL
      3d3dy53My5vcmcvMjAwMS9YTUxTY2hlbWEtaW5zdGFuY2UiLz48L09UUE1v
      ZGU+PC9FeHRlbnNpb24+PC9FeHRlbnNpb25zPjxNYWMgTWFjQWxnb3JpdGh
      tPSJjb20ucnNhLmN0a2lwLnRvb2xraXQudHlwZXMuTWFjVHlwZUA2Mjc5ZC
      IgeG1sbnM9IiIgeG1sbnM6ZHM9Imh0dHA6Ly93d3cudzMub3JnLzIwMDAvM
      DkveG1sZHNpZyMiIHhtbG5zOnhzaT0iaHR0cDovL3d3dy53My5vcmcvMjAw
      MS9YTUxTY2hlbWEtaW5zdGFuY2UiPk9uV0pwVXp2M2d5T3ZBeXQ4MHJWN1E
      9PTwvTWFjPjwvU2VydmVyRmluaXNoZWQ+</Response>
    </ServerResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Appendix C. 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 for our PKCS documents, results may also be submitted to standards forums. For more information, contact:

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

Authors' Addresses

Andrea Doherty
RSA, The Security Division of EMC

Email: adoherty@rsasecurity.com

Magnus Nystroem
RSA, The Security Division of EMC

Email: magnus@rsasecurity.com

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).