

## **A Proposal for Using XKMS with Card Management Systems**

Author: Mike Just, Entrust, Inc.

Date: June 27, 2001

The concepts presented in this paper have been reviewed and approved by Gemplus, Oberthur Card Systems and Schlumberger. All three card manufacturer support Entrust's initiative to extend XKMS to allow registration capability at the smart card production level on behalf of a number of users.

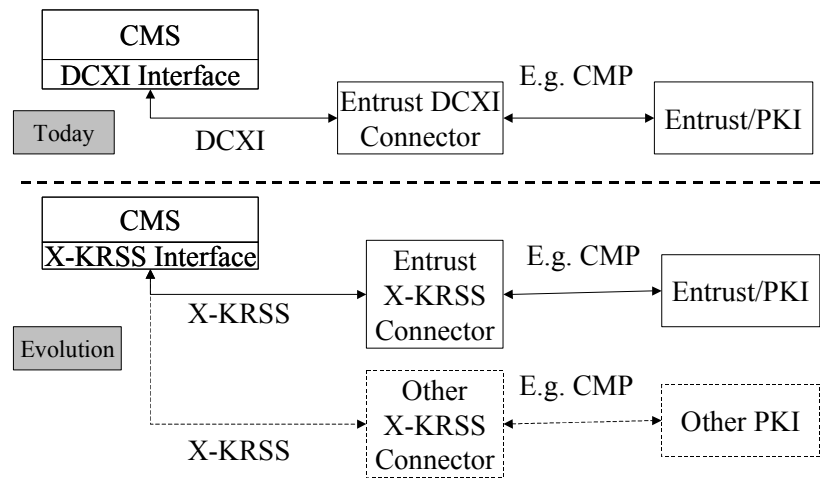
**Table of Contents**

1 Introduction ..... 3  
1.1 Influencing the X-KRSS standard..... 3  
1.2 References ..... 3  
1.3 Acknowledgements ..... 4  
2 Supporting suspension assertion ..... 4  
3 Support for bulk requests ..... 5  
3.1 Supporting bulk requests ..... 5  
3.2 Supporting asynchronicity..... 6  
4 Additional transaction management data ..... 6  
4.1 Unprocessed data..... 7  
Appendix A - Modified X-KRSS Schema for Supporting Bulk Registration Requests..... 8

# 1 Introduction

This document specifies how the X-KRSS (XML Key Registration Service Specification) portion of XKMS (XML Key Management Specification) may be modified so as to support the key management interface to a Card Management System (CMS). The suggested modifications are constructed so as to minimize the modifications to the current XKMS schema. Alternative constructions may be effective as well. In addition, some of the proposed changes may also support the requirements of other interfaces.

Entrust currently provides an XML-based interface known as DCXI (Device Connector XML Interface). The prime motivation for moving towards use of X-KRSS is that X-KRSS will eventually become a standardized protocol, offering an opportunity for interoperability for any client of the X-KRSS interface. This transition is depicted below.



## 1.1 Influencing the X-KRSS standard

As of June 2001, the current XKMS specification (version 1.1, draft 4) does not support all CMS requirements, including:

1. support for suspension and resumption,
2. support for bulk requests,
3. support for additional transaction processing data.

The remainder of this document examines these requirements and provides ways in which the X-KRSS schema may be modified in order to provide support. For the remainder of this paper, it is assumed that the reader is familiar with the XKMS specification. Throughout this document, proposed changes to the XKMS schema are highlighted in **bold**.

## 1.2 References

[1] XML Key Management Specification, W3C Technical Note, March 30, 2001, <http://www.w3.org/TR/xkms/>.

- [2] SOAP Security Extensions: Digital Signature, W3C Technical Note, February 6, 2001, <http://www.w3.org/TR/SOAP-dsig/>.
- [3] Web Services Description Language (WSDL) 1.1, W3C Note 15 March 2001 <http://www.w3.org/TR/2001/NOTE-wsdl-20010315>
- [4] XML Digital Signature Specification, W3C Candidate Recommendation/IETF Proposed Standard, <http://www.ietf.org/rfc/rfc3075.txt>.

### 1.3 Acknowledgements

Many thanks to the following for their time invested both in contributing to the ideas contained in this proposal, and their review of previous drafts:

- Patrick George (Gemplus), Bruno Leroux (Oberthur Card Systems), Philippe Mezger (Schlumberger).

## 2 Supporting suspension assertion

The current X-KRSS specification does not support requests for the suspension or resumption of an assertion. The motivation for such requests is to avoid, if possible, the requirement to reissue a token in case an action that would typically require a revocation, can be managed with a temporary revocation, or suspension.

Currently, the X-KRSS Registration request only supports a request for registration, revocation and recovery. At a high level, the X-KRSS schema for the Register message can be defined as follows:

```
<Register>
  <Prototype>
  <AuthInfo>
  <Respond>
```

A `<Status>` element included as part of the `<Prototype>` distinguishes the type of registration request: a *Valid* status indicates a Registration request; an *Invalid* status indicates a Revocation request; an Indeterminate status indicates a *Recovery* request. The schema for the status type is given below:

```
<simpleType name="AssertionStatus">
  <enumeration value="Valid"/>
  <enumeration value="Invalid"/>
  <enumeration value="Indeterminate"/>
</simpleType>
```

We propose here a secondary status type, defined as

```
<element name="SubStatus" type="AssertionSubStatus"/>
```

and to be included within the complex type "KeyBindingType". For the purpose of supporting suspension and recovery requests, we define the following

```
<simpleType name="AssertionSubStatus">
  <enumeration value="AssertionOnHold"/>
  <enumeration value="AssertionOffHold"/>
</simpleType>
```

where an "Invalid" assertion status and "AssertionOnHold" sub status together form a request for suspension of an assertion, while a "Valid" assertion status and "AssertionOffHold" sub status form a request for the resumption of an assertion.

It is anticipated that additional requirements (e.g. outside the scope of a CMS) may be satisfied by the inclusion of a more granular sub status element.

### 3 Support for bulk requests

A bulk registration request permits the registration of more than one <Prototype> element where only 1 authentication (e.g. signature) needs to be computed over the request. X-KRSS currently only supports the inclusion of a single <Prototype> element in a request. An implication of supporting a bulk request relates to the asynchronicity of the request. In particular, note that a very large bulk request may take a considerable time to process. Thus, it would be advantageous to support an asynchronous request-response.

Bulk requests would typically be submitted by an RA (Registration Authority), on behalf of a number of users. Notice that a request may be signed by an RA as specified by the KeyBindingAuth element.

```
<element name="KeyBindingAuth" minOccurs="0">
  <complexType>
    <sequence>
      <element ref="ds:Signature" minOccurs="0"/>
    </sequence>
  </complexType>
</element>
```

"<element ref="ds:Signature" minOccurs="0"/>" is defined in the XML Digital Signature specification.

#### 3.1 Supporting bulk requests

Bulk assertion requests may be supported by allowing for a multi-valued <Prototype> element as part of a <Register> element. The schema would change as follows, with modifications to the current XKMS schema in **bold**:

```
<element name="Prototype" type="s0:KeyBindingType" maxOccurs=unbounded
/>
<!-- All prototypes must be of the same Status. -->
<!-- More than 1 "Prototype" must not be support for requests of
"Indeterminate" status. -->
```

The complete schema change is shown in Appendix A. The accompanying comments have been made to restrict the complexity of a bulk registration request. As described in Appendix A, the request may be made with or without Proof of Possession (POP) corresponding to each <Prototype> element.

The request may be authenticated via the <KeyBindingAuth> element. Alternatively, in the case that SOAP enveloping is used, the request may be signed using SOAP digital signatures. In this case, with the <Register> element included as part of the SOAP body,

the entire SOAP body may be signed, with the signature included in the SOAP header. There is some additional overhead with this alternative since a greater amount of information will be signed, compared to what was signed using the XKMS <KeyBindingAuth> element.

The <RegisterResult> element defined in X-KRSS already supports the return of more than one <Prototype> element as part of a response. We assume that client-side key generation is performed so that the server will not be returning private key information. (If private key information for multiple <Prototype>s was to be returned, then the schema for the private key element would have to be modified.)

### 3.2 Supporting asynchronicity

Especially in situations in which a large number of <Prototype> elements are included in a registration request, an extended amount of time might be required to process the entire request. In this situation, it should be possible for the client to drop their connection, obtaining the result to their registration request at a later time.

In this case, the initial <RegisterResult> would include "ResultCode=Pending". Since no additional information need be returned in this case, it is recommended that the XKMS schema be modified as follows:

```
<element name="Answer" minOccurs=0>
<!-- An answer must be provided in case the "ResultCode=Success". -->
```

In most cases it may be assumed that bulk requests are submitted by Web services like CMS, which can act as client and as servers.

In principal there are two possible models for obtaining subsequent results:

- Model 1 RPC like call, where the CMS sends a request batch to the CA and the CA returns the response batch to the CMS. As already indicated, because of the long processing time, this RPC call needs to be split into two SOAP HTTP requests, i) to send the request batch, which might return some control information and ii) a second request to download the response batch and the fault message. If the CA has not yet finished processing the batch additional polling with the second request is required.
- Model 2 "messaging like" call, where the CMS sends the request batch using a message system or a SOAP HTTP request to the CA. When the CA has processed the request batch it sends the response batch using a message system or a SOAP HTTP request to the CMS.

Both the CMS and the CA Web service will have to define what transport protocols it implements using WSDL [3].

## 4 Additional transaction management data

Additional information that needs to be included in an X-KRSS request/response is described in this section. This includes information such as

- a count of the number of <Prototype> elements included as part of a request and returned as part of a response, acting as a "sanity check" during the construction of a message with a large number of <Prototype> elements;
- certificate template information, and user-specific certificate extensions which indicate additional information to be included for each assertion.

Additional information such as this is currently accommodated in the X-KRSS schema through the <ProcessInfo> element. Specific requirements such as these will be more fully specified using the <ProcessInfo> syntax at a later time.

Unprocessed data, having unique processing requirements, is explicitly defined (without using the <ProcessInfo> element) on its own below.

#### **4.1 Unprocessed data**

A particular requirement within a CMS is to have certain information submitted as part of a registration request, and have this information returned, unprocessed, as part of the response. (Specifically for the purposes of a CMS, it is Enterprise Resource Planning data.) This is particularly useful for the management of bulk requests. It is likely that such an element would be useful in other applications as well. For this reason, we define the following element, whose semantics for the server are that the data simply be returned (unprocessed) in the <RegistrationResult>.

```
<element name="ReturnProcessInfo" minOccurs="0">
  <complexType>
    <sequence>
      <element name="returnInfo" type="string" minOccurs="0",
maxOccurs="unbounded">
    </sequence>
  </complexType>
</element>
```

This element would be included in the schema for both the registration request and registration response, in X-KRSS.

## Appendix A - Modified X-KRSS Schema for Supporting Bulk Registration Requests

Proposed changes to X-KRSS schema in support of bulk requests were described earlier. This appendix provides more detailed information on the proposed changes.

At a high level, the X-KRSS schema for the Register message can be defined as follows:

```
<Register>
  <Prototype>
  <AuthInfo>
  <Respond>
```

where <Prototype> contains the public key and name information for the requesting entity, <AuthInfo> includes the POP as well as information for authenticating the request (more accurately, it is authentication over the <Prototype> element), and <Respond> contains a list of attributes that the requestor would like returned.

In a nutshell, support for bulk requests would allow for more than one <Prototype> element. There are many ways in which the current X-KRSS schema may be altered to allow support for bulk requests. We propose the following changes, where a primary goal was to limit the number of changes required to the schema.

```
<s:element name="Register">
  <s:complexType>
    <s:sequence>
```

```
      <s:element name="Prototype" type="s0:KeyBindingType"
maxOccurs=unbounded />
      <!-- All prototypes must be of the same Status. -->
      <!-- More than 1 "Prototype" must not be support for requests of
"Indeterminate" status. -->
```

```
      <s:element name="AuthInfo" minOccurs="0">
        <s:complexType>
          <s:choice>
            <s:element name="AuthUserInfo"
type="s0:AuthUserInfoType"/>
            <s:element name="AuthServerInfo"
type="s0:AuthServerInfoType"/>
          </s:choice>
        </s:complexType>
      </s:element> <!-- "AuthInfo" -->
```

```
<!-- "AuthUserInfo" is used for bulk requests. The key binding will
still apply to the prototype element. -->
```

```
      <s:element name="Respond" minOccurs="0" >
        <s:complexType>
          <s:sequence>
            <s:element name="string" type="string" minOccurs="0"
maxOccurs="unbounded"/>
          </s:sequence>
        </s:complexType>
      </s:element> <!-- "Respond" -->
```

```
<!-- "Respond" applies similarly to each and every <Prototype> element.
-->
```

```
  </s:sequence>
```

```

    </s:complexType>
  </s:element>

```

The "AuthUserInfo" element is included below:

```

<s:complexType name="AuthUserInfoType">
  <s:sequence>
    <s:element name="ProofOfPossession" minOccurs="0"
      maxOccurs=unbounded >
      <s:complexType>
        <s:sequence>
          <s:element ref="ds:Signature" minOccurs="0"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <!-- Each "ProofOfPossession" element in the sequence of multiple
    elements corresponds 1-1 to each "Prototype" element. -->
    <s:element name="KeyBindingAuth" minOccurs="0">
      <s:complexType>
        <s:sequence>
          <s:element ref="ds:Signature" minOccurs="0"/>
        </s:sequence>
      </s:complexType>
    </s:element>
    <!-- The "KeyBindingAuth" element applies to the sequence of 1 or
    more "Prototype" elements. -->
    <s:element name="PassPhraseAuth" type="string" minOccurs="0"
      maxOccurs=unbounded />
    <!-- Each "PassPhraseAuth" element in the sequence of multiple
    elements corresponds 1-1 to each "Prototype" element. -->
  </s:sequence>
</s:complexType>

```