Abstract:
This specification defines the first (1.0) version of the Symmetric Key Services Markup Language (SKSML).

Status:
This document was last revised or approved by the EKMI TC on the above date. The level of approval is also listed above. Check the “Latest Version” or “Latest Approved Version” location noted above for possible later revisions of this document.

Technical Committee members should send comments on this specification to the Technical Committee’s email list. Others should send comments to the Technical Committee by using the “Send A Comment” button on the Technical Committee’s web page at http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=ekmi

For information on whether any patents have been disclosed that may be essential to implementing this specification, and any offers of patent licensing terms, please refer to the Intellectual Property Rights section of the Technical Committee web page (http://www.oasis-open.org/committees/ekmi/ipr.php).

The non-normative errata page for this specification is located at http://www.oasis-open.org/committees/[TBD]/.
Notices

Copyright © OASIS® 2008. All Rights Reserved.

All capitalized terms in the following text have the meanings assigned to them in the OASIS Intellectual Property Rights Policy (the "OASIS IPR Policy"). The full Policy may be found at the OASIS website.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published, and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this section are included on all such copies and derivative works. However, this document itself may not be modified in any way, including by removing the copyright notice or references to OASIS, except as needed for the purpose of developing any document or deliverable produced by an OASIS Technical Committee (in which case the rules applicable to copyrights, as set forth in the OASIS IPR Policy, must be followed) or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by OASIS or its successors or assigns.

This document and the information contained herein is provided on an "AS IS" basis and OASIS DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY OWNERSHIP RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

OASIS requests that any OASIS Party or any other party that believes it has patent claims that would necessarily be infringed by implementations of this OASIS Committee Specification or OASIS Standard, to notify OASIS TC Administrator and provide an indication of its willingness to grant patent licenses to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification.

OASIS invites any party to contact the OASIS TC Administrator if it is aware of a claim of ownership of any patent claims that would necessarily be infringed by implementations of this specification by a patent holder that is not willing to provide a license to such patent claims in a manner consistent with the IPR Mode of the OASIS Technical Committee that produced this specification. OASIS may include such claims on its website, but disclaims any obligation to do so.

OASIS takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on OASIS’ procedures with respect to rights in any document or deliverable produced by an OASIS Technical Committee can be found on the OASIS website. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this OASIS Committee Specification or OASIS Standard, can be obtained from the OASIS TC Administrator. OASIS makes no representation that any information or list of intellectual property rights will at any time be complete, or that any claims in such list are, in fact, Essential Claims.

The names "OASIS", [insert specific trademarked names, abbreviations, etc. here] are trademarks of OASIS, the owner and developer of this specification, and should be used only to refer to the organization and its official outputs. OASIS welcomes reference to, and implementation and use of, specifications, while reserving the right to enforce its marks against misleading uses. Please see http://www.oasis-open.org/who/trademark.php for above guidance.
# Table of Contents

1 Introduction ................................................................. 5
   1.1 Terminology ............................................................. 5
   1.2 Glossary ................................................................. 5
   1.3 Normative References ................................................ 6
   1.4 Non-normative References ......................................... 6

2 Background (non-normative) ....................................... 7
   2.1 Requirements (non-normative) .................................... 8

3 Examples of use of SKSML (non-normative) .................. 10
   3.1 Request for a new symmetric key ............................... 10
   3.2 Response with a new symmetric key ............................ 12
   3.3 Request for an existing symmetric key ......................... 15
   3.4 Response with an existing symmetric key ..................... 16
   3.5 Request for a new symmetric key of a specific KeyClass .................. 16
   3.6 Response with a new symmetric key of a specific KeyClass ........ 17
   3.7 Request for multiple new symmetric keys ..................... 17
   3.8 Response with multiple new symmetric keys ................. 18
   3.9 Response with an SKS error ...................................... 22
   3.10 Response with symmetric keys and errors .................... 22
   3.11 Request for a symmetric key-caching policy ................. 24
   3.12 Response with a symmetric key-caching policy (1) .......... 26
   3.13 Response with a symmetric key-caching policy (2) .......... 27
   3.14 Response with multiple symmetric key-caching policies (3) ... 29
1 Introduction

This document presents the specification for the Symmetric Key Services Markup Language (SKSML), a protocol by which applications may request and receive symmetric key-management services, securely, over the network. All text is non-normative unless otherwise indicated.

1.1 Terminology

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this specification are to be interpreted as described in IETF RFC 2119.

1.2 Glossary

3DES – Triple Data Encryption Standard

AES – Advanced Encryption Standard

Base64 – An encoding scheme for representing data

Ciphertext – Encrypted data

Cryptographic module – A software library or hardware module dedicated to performing cryptographic operations

DES – Data Encryption Standard

DID or Domain ID – Domain Identifier; the unique PEN assigned to an implementation of an SKMS within an enterprise

GKID or Global Key ID – Global Key Identifier; the unique identifier assigned to every symmetric encryption key within an SKMS. It is the concatenation of the DID-SID-KID

Initialization Vector or IV – A block of bits required to encrypt/decrypt the first block of data when used with a particular mode of cryptographic operations

KeyCachePolicy – The collection of rules that defines how a symmetric encryption key may be cached by a client implementation

KID or Key ID – Key Identifier; the unique integer assigned to every symmetric encryption key generated within a specific SKS server within an SKMS

KeyUsePolicy – The collection of rules that defines how a symmetric encryption key may be used by an application

PEN – Private Enterprise Number; the unique integer assigned by IANA to any organization that requests such a number

PII – Personally Identifiable Information, such as credit card numbers, social security numbers, bank account numbers, drivers license numbers, etc.

Plaintext – Unencrypted data

SHA – Secure Hashing Algorithm

SHA-1 – Secure Hashing Algorithm with a size of 160-bits

SHA-256 – Secure Hashing Algorithm with a size of 256-bits

SHA-384 – Secure Hashing Algorithm with a size of 384-bits
SHA-512 – Secure Hashing Algorithm with a size of 512-bits

SID or Server ID – Server Identifier; the unique integer assigned to every SKS server within an enterprise's SKMS

SKCL – Symmetric Key Client Library; a software library that supports the SKSML protocol

SKMS – Symmetric Key Management System; a collection of hardware and software providing symmetric encryption key-management services

SKS – Symmetric Key Services; a server that provides symmetric key management services over the network

SKSML – Symmetric Key Services Markup Language; an XML-based protocol to request and receive symmetric encryption key-management services

SOAP – Simple Object Access Protocol

SOAP Body – The content part of a SOAP message

SOAP Envelope – The SOAP message consisting of a SOAP Header and a SOAP Body, conforming to the SOAP protocol standard.

SOAP Error – A SOAP error message response to a SOAP request

SOAP Header – The header part of a SOAP message containing meta-information about the message, including security-related objects

Symkey – A symmetric encryption key

XMLEncryption – Encrypted content represented in eXtensible Markup Language and conformant to the World Wide Web Consortium's XML Encryption standard

XMLSignature – A digital signature represented in eXtensible Markup Language and conformant to the World Wide Web Consortium's XML Signature standard

1.3 Normative References

[AES] Advanced Encryption Standard


[SOAP] Simple Object Access Protocol 1.1

[XMLEncryption] XML Encryption Syntax and Processing

[XMLSignature] XML Signature Syntax and Processing

OASIS Standard 200401, March 2004


1.4 Non-normative References

A confluence of events is causing many companies to consider encrypting sensitive data across many applications and platforms within their IT infrastructure. Some of these events include:

- "Breach Disclosure" laws in nearly 40 states of the USA, requiring companies that have suffered breaches on computers containing Personally Identifiable Information (PII) of their employees or customers, to disclose those breaches to the affected individuals
- Industry-specific regulations such as the credit card industry's Payment Card Industry Data Security Standard, requiring the encryption of credit card numbers accompanied with strong key-management controls
- National laws such as the US' Health Insurance Portability and Accountability Act (HIPAA) and the European Union Directive, requiring the securing of health-related data and PII, respectively
- A significant increase in the number of business applications and e-commerce services on the internet requiring credit card numbers for payment, which in turn becomes a target for attackers
- A significant increase in the number of users connected to the internet with inadequate protection, leading to many attack vectors becoming propagated on these unprotected PC's

In a rush to provide solutions to the market, vendors have created many device-specific, platform-specific, database-specific and application-specific encryption and key-management tools. While these tools may be capable of performing their stated tasks adequately, a typical enterprise would have to deal with many encryption and key-management solutions to adequately protect sensitive data. The following illustration shows how the same key-management tasks need to repeated across every single device, platform, database and/or application where encryption is performed:

Not only does this raise the cost of ownership for implementing companies, but it raises the possibility that with many dissimilar key-management systems, because of the typical complexity of key-management schemes, there is a greater likelihood of human error leading to a vulnerability.

To ensure that encryption policies and designs are specified and used uniformly across applications, a common key-management service capable of supporting enterprise platforms, applications and devices is needed. To enable such applications to communicate with this service, a uniform protocol is needed. The Symmetric Key Services Markup Language (SKSML) is that protocol.

Once an enterprise has implemented an SKMS, and applications have been modified to take advantage of SKSML, they can expect to see their key-management infrastructure to resemble the following diagram:
Architected much like the Domain Name Service (DNS), an SKMS becomes the focal point for all symmetric encryption key-management services.

The Symmetric Key Client Library (SKCL) on client devices is responsible for communicating with the Symmetric Key Services (SKS) server using SKSML. The SKCL handles security, caching, cryptographic operations and ensuring that the use of the key is in conformance to policies specified for the key.

The SKS server is responsible for storing all policies, keys and information about authorized clients and servers within the SKMS, and responds to client requests.

2.1 Requirements (non-normative)

The requirements of the SKSML protocol are that:

- It must be platform independent;
- It must support the request of new and previously escrowed symmetric encryption keys;
- It must support the unique identification of every symmetric encryption key on the internet;
- It must provide message authenticity, confidentiality and integrity even when used over insecure networks;
- It must support the use of encryption/decryption services by a client even when disconnected from the network;
- It must provide flexibility in defining key-usage policies;

SKSML meets the above requirements in the following manner:

- SKSML uses SOAP and XML for encapsulating its requests and responses and can thus, be used on any platform that supports these two underlying protocols;
- Using a scheme that concatenates unique Domain identifiers (Private Enterprise Numbers issued by the IANA), unique SKS Server identifiers within a domain and unique Key identifiers within an SKS server, SKSML creates Global Key Identifiers (GKID) that can uniquely identify symmetric keys across the internet;
- SKSML relies on the Web Services Security (WSS) standard 1.0, which in turn supports the use of XML Signature and XML Encryption within the SOAP Header. Relying only the on the WSS profile that uses
RSA cryptographic key-pairs and digital certificates, SKSML uses the digital signatures for authenticity and message-integrity, while using RSA-encryption for confidentiality;

- Using secure key-caching enabled through centrally-defined policies, SKSML supports the request and receipt of `KeyCachePolicy` elements by clients for the use of symmetric encryption keys even when the client is disconnected from the network and an SKS server;

- SKSML provides significant flexibility for defining policies on how symmetric encryption keys may be used by client applications. The `KeyUsePolicy` element allows Security Officers to define which applications may use a specific key, days and times of use, location of use, purpose of use, key-sizes, encryption algorithms, etc.

SKSML is the first key-management protocol that will do for encryption key-management services what DNS did for name-service protocols: provide a single, standard means of requesting and receiving key-management services from centrally defined servers.
3 Examples of use of SKSML (non-normative)

The following high-level diagram will be used to describe the use of SKSML.

1. Client Application makes a request for a symmetric key
2. SKCL makes a digitally signed request to the SKS
3. SKS verifies SKCL request, generates, encrypts, digitally signs & escrows key in DB
4. Crypto HSM provides security for RSA Signing & Encryption keys of SKS
5. SKS responds to SKCL with signed and encrypted symmetric key
6. SKCL verifies response, decrypts key and hands it to the Client Application
7. Native (non-Java) applications make requests through Java Native Interface

3.1 Request for a new symmetric key

When a client application (that has been linked to the SKCL) needs to encrypt sensitive data, it will call an API method within the SKCL for a new symmetric key. After the SKCL has ensured that the application is authorized to make such a request (by verifying that the configured/passed-in credentials can access the cryptographic key-store module on the client containing the PrivateKey used for signing SKSML requests), the SKCL assembles the following SKSML request:

```xml
<ekmi:SymkeyRequest
xmlns:ekmi="http://docs.oasis-open.org/ekmi/2008/01">
<ekmi:GlobalKeyID>10514-0-0</ekmi:GlobalKeyID>
</ekmi:SymkeyRequest>
```

[a01] is the start of the `SymkeyRequest` element.

[a02] identifies the namespace to which this XML conforms, and the location of its XML Schema Definition (XSD).

[a03] identifies the `GlobalKeyID` (GKID) being requested by the client application. The GKID is a concatenation of three distinct identifiers in the following order: the unique Domain Identifier, the unique Server Identifier within the domain and the unique Key Identifier generated on a server. Using a “zero” value for the Server ID and the Key ID indicates a request for a new symmetric key.

[a04] is the closing tag of the `SymkeyRequest` element.

While the `SymkeyRequest` element is very simple, the Web Service Security (WSS) envelope – which provides security for all SKSML messages – expands the size of the message. The same request shown above, is displayed below in its entirety, with its WSS envelope. Please note that some content – such as Base64-encoded binary content - has been reformatted for aesthetics and clarity of the XML elements. The actual elements and data-types have been preserved from actual SKSML messages.
For an interpretation of the XML elements shown below, please refer to [WSS].

For the sake of brevity, this specification will dispense with showing the SOAP envelope and the WSS elements in all other examples, when discussing SKSML.

```xml
  <SOAP-ENV:Header>
    </wsse:Security>
  </SOAP-ENV:Header>
</SOAP-ENV:Envelope>
```

Value="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-security-utility-1.0.xsd"
3.2 Response with a new symmetric key

After an SKS server has performed its operations of authenticating the request, identifying the requester, determining policies that apply to the requester, generating the symmetric encryption key in conformance to the defined policy and finally escrowing a symmetric key securely, it assembles the following response and returns it to the client. (The SOAP message, as indicated earlier, is secured using WSS, but only the actual SKSML content is displayed and discussed here).

```xml
<ekmi:SymkeyResponse
 xmlns:ekmi='http://docs.oasis-open.org/ekmi/2008/01'
 xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
<ekmi:GlobalKeyId>10514-1-235</ekmi:GlobalKeyId>
<ekmi:KeyUsePolicy>
<ekmi:KeyUsePolicyID>10514-4</ekmi:KeyUsePolicyID>
<ekmi:PolicyName>DES-EDE KeyUsePolicy</ekmi:PolicyName>
<ekmi:KeyClass>HR-Class</ekmi:KeyClass>
<ekmi:KeyAlgorithm>
http://www.w3.org/2001/04/xmlenc#tripledes-cbc
</ekmi:KeyAlgorithm>
<ekmi:KeySize>192</ekmi:KeySize>
<ekmi:Status>Active</ekmi:Status>
<ekmi:Permissions>
<ekmi:PermittedApplications>
<ekmi:PermittedApplication>
<ekmi:ApplicationID>10514-23</ekmi:ApplicationID>
<ekmi:ApplicationName>Payroll Application</ekmi:ApplicationName>
</ekmi:PermittedApplication>
</ekmi:PermittedApplications>
</ekmi:KeyUsePolicy>
</ekmi:SymkeyResponse>
```

Copyright © OASIS® 2008. All Rights Reserved. 17. June 2008
<ekmi:PermittedApplication>
</ekmi:PermittedApplication>
<ekmi:PermittedApplications>
<ekmi:PermittedDate>
<ekmi:StartDate>2008-01-01</ekmi:StartDate>
<ekmi:EndDate>2008-12-31</ekmi:EndDate>
</ekmi:PermittedDate>
</ekmi:PermittedDates>
<ekmi:PermittedTimes>
<ekmi:PermittedTime>
<ekmi:StartTime>07:00:00</ekmi:StartTime>
<ekmi:EndTime>19:00:00</ekmi:EndTime>
</ekmi:PermittedTime>
</ekmi:PermittedTimes>
</ekmi:Permissions>
</ekmi:KeyUsePolicy>
<ekmi:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
<ekmi:CipherData>
<ekmi:CipherValue>
E9zWB/y93hV5zeTliCq0DxmlNxTuXsffMNwCJmtI1dIzq0HBnpd281g60dKdkCFjJMhQyvwXqsFyjv9h5FDqUj0XG0ca8EU871zBoXBjDxjfglpU8tGFbWZcd/ATpJD/2fwUJow/qimx18+huUIYJMaGHTxULWtx27STRcRpisY=
</ekmi:CipherValue>
</ekmi:CipherData>
</ekmi:SymkeyResponse>
</ekmi:SymkeyResponse>
</ekmi:Symkey>
</ekmi:KeyUsePolicy>
</ekmi:KeyAlgorithm>
</ekmi:Permissions>
</ekmi:KeyUsePolicy>
</ekmi:KeyAlgorithm>
</ekmi:SymkeyResponse>

[b01] is the start of the **SymkeyResponse** element.

[b02] and [b03] identify the namespaces to which this XML conforms, and the location of their XML Schema Definitions (XSD).

[b04] is the start tag of the **Symkey** element which contains the symmetric encryption key and related elements.

[b05] identifies the **GlobalKeyID** (GKID) assigned by the SKS server for the new symmetric key being returned.

In this example, the concatenated values of the Domain ID, Server ID and Key ID indicate that the key belongs to the organization represented by the PEN, 10514; it was generated on an SKS server with a Server ID of 1 and is the 235th unique key generated on that SKS server.

[b06] is the start of the **KeyUsePolicy** element. This element contains details of the policy to which SKCL implementations must conform when using the symmetric key.

[b07] identifies the unique **KeyUsePolicyID** (KUPID) which identifies this policy within the SKMS.

[b08] provides a descriptive name for this key-use policy, which is helpful to human readers when identifying this policy.

[b09] identifies the **KeyClass** to which this symmetric key belongs. Key-classes are useful to applications that wish to encrypt plaintext with a key that has specific characteristics. The requesting application is expected to know what **KeyClass** it needs before it asks for a key corresponding to that class.

[b10] is the start tag of the **KeyAlgorithm** element.

[b11] identifies the cryptographic algorithm that this symmetric key must be used with for cryptographic operations.

[b12] is the closing tag of the **KeyAlgorithm** element.

[b13] specifies the size of the symmetric encryption key in bits. While it is possible for application developers to determine this programmatically from the key-object, this element provides this information as a convenience.
[b14] indicates the **Status** of this **KeyUsePolicy** and whether it is an active policy or not. This is useful in situations where an application may wish to re-use a symmetric key to encrypt related data to the data originally encrypted with the symmetric key. While it is possible for the symmetric key object to be active in the database, it is conceivable that the **KeyUsePolicy** used by the key has changed and the application technically needs to use a new symmetric key to encrypt new data.

[b15] is the start of the **Permissions** element. This element provides a sophisticated mechanism for controlling how, when and by which applications symmetric keys be used. While there are many sub-elements within a **Permissions** element, not all **KeyUsePolicy** objects might use all **Permissions** sub-elements. The example shown in this section only uses three of the possible **Permissions** sub-elements.

[b16] is the start of the **PermittedApplications** element. This element allows SKMS policies to be defined that allow only specific applications to use symmetric encryption keys associated with this policy. The absence of the **PermittedApplications** element indicates that any application on the client machine is permitted to use this symmetric key.

[b17] is the start of the first **PermittedApplication** element. This element identifies a specific application within a list of **PermittedApplications** that is allowed to use the symmetric key. There can be any number of **PermittedApplication** elements within a **PermittedApplications** element.

[b18] identifies the unique **ApplicationID** (as identified within the SKMS) of the **PermittedApplication**.

[b19] is the start of the **ApplicationName** element.

[b20] identifies the **ApplicationName** of the **PermittedApplication** (as identified within the SKMS).

[b21] is the closing tag of the **ApplicationName** element.

[b22] identifies the specific **ApplicationVersion** of the **PermittedApplication**. This is helpful when there are multiple versions of a specific application, and the policy-makers need to distinguish between the symmetric keys in use by a specific version of the application.

[b23] is the start of the **ApplicationDigestAlgorithm** element. This element permits enterprises to ensure that only a cryptographically-verified application is authorized to use the symmetric encryption key. This assumes that the implementation has an infrastructure where the SKCL is capable of determining a cryptographic value to uniquely identify an application within the run-time environment.

[b24] identifies the W3C-specified URL of the cryptographic algorithm used to calculate the message digest of the application's image.

[b25] is the closing tag of the **ApplicationDigestAlgorithm** element.

[b26] is the start of the **ApplicationDigestValue** element. This element permits enterprises to ensure that only a cryptographically-verified application is authorized to use the symmetric encryption key. This assumes that the implementation has an infrastructure where the SKCL is capable of determining a cryptographic value to uniquely identify an application within the run-time environment.

[b27] identifies the Base64-encoded message digest of the **PermittedApplication**'s image, based on the algorithm specified in the **ApplicationDigestAlgorithm** element.

[b28] is the closing tag of the **ApplicationDigestValue** element.

[b29] is the closing tag of the **PermittedApplication** element.

[b30] is the closing tag of the **PermittedApplications** element.

[b31] is the start of the **PermittedDates** element. This element permits enterprises to ensure that the symmetric encryption key can be used only during a specified date period. This assumes that the implementation has an infrastructure where the SKCL is capable of accurately determining the current date within the run-time environment.

[b32] is the start of the first **PermittedDate** element. There can be any number of **PermittedDate** elements within a **PermittedDates** element.
identifies the **Start Date** of the duration period during which the symmetric encryption key can be used by authorized applications.

[b34] identifies the **EndDate** of the duration period during which the symmetric encryption key can be used by authorized applications.

[b35] is the closing tag of the **PermittedDate** element.

[b36] is the closing tag of the **PermittedDates** element.

[b37] is the start of the **PermittedTimes** element. This element permits enterprises to ensure that the symmetric encryption key can be used only during a specified time period within any date. This assumes that the implementation has an infrastructure where the SKCL is capable of accurately determining the current time within the run-time environment.

[b38] is the start of the first **PermittedTime** element. There can be any number of **PermittedTime** elements within a **PermittedTimes** element.

[b39] identifies the **StartTime** of the duration period during which the symmetric encryption key can be used by authorized applications.

[b40] identifies the **EndTime** of the duration period during which the symmetric encryption key can be used by authorized applications.

[b41] is the closing tag of the **PermittedTime** element.

[b42] is the closing tag of the **PermittedTimes** element.

[b43] is the closing tag of the **Permissions** element.

[b44] is the closing tag of the **KeyUsePolicy** element.

[b45]-[b46] identifies the encryption algorithm used in the **EncryptionMethod** element to encrypt the symmetric encryption key itself, to transport to the requesting client. The symmetric key is encrypted using the Public Key or the requesting client. The **Algorithm** attribute uses the W3C-specified URLs for identifying the encryption and padding algorithms.

[b47] is the start of the **CipherData** element. This element is from the W3C XML Encryption namespace (as identified by the “xenc” qualifier in the element name).

[b48] is the start of the **CipherValue** element. This element contains the Base64-encoded ciphertext of the symmetric encryption key.

[b49] – [b51] is the Base64-encoded ciphertext of the symmetric encryption key.

[b52] is the closing tag of the **CipherValue** element.

[b53] is the closing tag of the **CipherData** element.

[b54] is the closing tag of the **Symkey** element.

[b55] is the closing tag of the **SymkeyResponse** element.

### 3.3 Request for an existing symmetric key

Typically, when a client application encrypts data, it must make accommodations to store the **GlobalKeyID** of the symmetric encryption key it uses to encrypt the plaintext, along with the ciphertext. Without the **GlobalKeyID**, neither the client application nor the SKS server can determine which key was used to encrypt specific ciphertext. When the client application needs to decrypt such ciphertext, it must request the symmetric key with the appropriate **GlobalKeyID** from the SKS server if it does not already have the key cached within its key-cache.

The client application (linked to the call SKCL) will an API method within the SKCL for the appropriate symmetric key. After the SKCL has ensured that the application is authorized to make such a request, and assuming that
the client application needs the key with the **GlobalKeyID** value of 10514-1-235, the SKCL assembles the following SKSML request. (The SOAP message is secured using WSS, but only the actual SKSML content is displayed and discussed here).

```
[c01] <ekmi:SymkeyRequest
[c02] xmlns:ekmi="http://docs.oasis-open.org/ekmi/2008/01">
[c03] <ekmi:GlobalKeyID>10514-1-235</ekmi:GlobalKeyID>
[c04] </ekmi:SymkeyRequest>
```

[c01] is the start of the *SymkeyRequest* element.

[c02] identifies the namespace to which this XML conforms, and the location of its XML Schema Definition (XSD).

[c03] identifies the **GlobalKeyID** (GKID) of the specific symmetric encryption key being requested by the client application.

[c04] is the closing tag of the *SymkeyRequest* element.

Note that the request for an existing symmetric key is no different from a request for a new symmetric key other than that the **GlobalKeyID** being requested has *non-zero values* for the Server ID and Key ID parts of the **GlobalKeyID**.

### 3.4 Response with an existing symmetric key

After an SKS server has performed its operations of authenticating the request, identifying the requester and determining whether the requester is authorized to receive the symmetric key, the SKS server sends back the symmetric key using a *SymkeyResponse* message secured within a WSS envelope. This *SymkeyResponse* is identical to the message described in Section 3.2 (since the *SymkeyRequest* was for the same symmetric key identified there) and is therefore, not repeated here for brevity.

### 3.5 Request for a new symmetric key of a specific **KeyClass**

There are business situations where an application might need a symmetric key that conforms to a **KeyUsePolicy** that has a specific **KeyClass** attribute within the policy. This is useful in situations where there may be many encryption policies within a company that apply to different type of data, geographical zones, applications, level of access to sensitive data, etc., and the requesting application needs a symmetric key which satisfies its business rules.

In those situations, a client application (that has been linked to the SKCL) will call an API method within the SKCL for a new symmetric key of a specific **KeyClass**. After the SKCL has ensured that the application is authorized to make such a request the SKCL assembles the following SKSML request:

```
[e01] <ekmi:SymkeyRequest
[e02] xmlns:ekmi="http://docs.oasis-open.org/ekmi/2008/01">
[e03] <ekmi:GlobalKeyID>10514-0-0</ekmi:GlobalKeyID>
[e04] <ekmi:KeyClasses>
[e05] <ekmi:KeyClass>HR-Class</ekmi:KeyClass>
[e06] </ekmi:KeyClasses>
[e07] </ekmi:SymkeyRequest>
```

[e01] is the start of the *SymkeyRequest* element.

[e02] identifies the namespace to which this XML conforms, and the location of its XML Schema Definition (XSD).

[e03] identifies the **GlobalKeyID** (GKID) being requested by the client application. The “zero” value for the Server ID and the Key ID indicates a request for a new symmetric key.

[e04] is the start of the *KeyClasses* element.
3.6 Response with a new symmetric key of a specific KeyClass

After an SKS server has performed its operations of authenticating the request, identifying the requester and determining whether the requester is authorized to receive a symmetric key of the requested KeyClass, the SKS server generates, escrows, encrypts and sends back the symmetric key using a SymkeyResponse message secured within a WSS envelope. This SymkeyResponse is identical to the message described in Section 3.2 (since the symmetric key identified there is of the KeyClass requested here) and is therefore, not repeated here for brevity.

3.7 Request for multiple new symmetric keys

There are business situations where an application needs many symmetric keys to encrypt different parts of a single record. This is useful in applications where many entities might have access to the same “master” record, but only some entities have access to sensitive data within “detail” records. In these situations, the use of multiple symmetric keys addresses this business requirement, while allowing the entire record to freely move to any system without fear of loss of confidentiality.

For example, within an Electronic Health Record (EHR) application, the application might store a Patient’s medical data as a single logical EHR within a database (even though they may be physically represented by many hundreds of detail records). This has the benefit of presenting a single view of a Patient's EHR to all actors within the use-case. However, the information necessary to a Physician treating the patient is quite different from the information necessary to an insurance company processing a claim, a government agency tracks diseases, a pharmacy filling out a prescription or a nurse administering treatment to the patient.

While there is a clear business advantage to maintaining all patient data as a single logical EHR, security and privacy requirements dictate that appropriate sensitive data must be made visible only to authorized entities.

In these situations, a client application (that has been linked to the SKCL) will call an API method within the SKCL for multiple new symmetric keys. In order to request multiple symmetric keys, the SKSML request must contain a KeyClass element within the KeyClasses element for every key the client application needs. Thus, if the EHR application were to request nine symmetric keys to encrypt different parts of the EHR, the client application would make the following SKSML SymkeyRequest:

```xml
<ekmi:SymkeyRequest
xmlns:ekmi="http://docs.oasis-open.org/ekmi/2008/01"
><ekmi:GlobalKeyID>10514-0-0</ekmi:GlobalKeyID>
<ekmi:KeyClasses>
  <ekmi:KeyClass>EHR-CDC</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-CRO</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-DEF</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-EMT</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-HOS</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-INS</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-NUR</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-PAT</ekmi:KeyClass>
  <ekmi:KeyClass>EHR-PHY</ekmi:KeyClass>
</ekmi:KeyClasses>
</ekmi:SymkeyRequest>
```

[g01] is the start of the SymkeyRequest element.

[g02] identifies the namespace to which this XML conforms, and the location of its XML Schema Definition (XSD).
[g03] identifies the **GlobalKeyID** (GKID) being requested by the client application. The “zero” value for the
Server ID and the Key ID indicates a request for a new symmetric key.

[g04] is the start of the **KeyClasses** element.

[g05] identifies a **KeyClass** for a symmetric encryption key being requested by the client application, ostensibly
for encrypting data that can later be decrypted by an authorized application at the Center for Disease Control
(CDC) and any other application that has been granted access to keys of the EHR-CDC **KeyClass**.

[g06] identifies a **KeyClass** for a symmetric encryption key being requested by the client application, for
encrypting data that can later be decrypted only by an authorized application at Clinical Research Organizations
(CRO) and any other application that has been granted access to keys of the EHR-CRO **KeyClass**.

[g07] identifies a default **KeyClass** (EHR-DEF) for a symmetric encryption key for encrypting data that can later
be decrypted by any authorized application within the EHR system.

[g08] identifies a **KeyClass** for a symmetric encryption key for encrypting data that was collected by an
Emergency Medical Technician (EMT), and which can later be decrypted only by authorized applications at the
hospital that have access to keys of the EHR-EMT **KeyClass**.

[g09] identifies a **KeyClass** for a symmetric encryption key for encrypting data collected by a hospital where the
patient might have used for treatment. Data encrypted by keys of this EHR-HOS **KeyClass** can later be
decrypted only by authorized application that has access to keys of this **KeyClass**.

[g10] identifies a **KeyClass** (EHR-INS) for a symmetric encryption key that might be used for encrypting data
which will be submitted to an insurance company for claims processing.

[g11] identifies a **KeyClass** for a symmetric encryption key for encrypting data that can later be decrypted by
any authorized application used by nurses and/or other authorized entities in providing treatment to a patient at
the hospital (EHR-NUR).

[g12] identifies a **KeyClass** for a symmetric encryption key for encrypting patient related data (EHR-PAT) that
may not be medical in nature, but nevertheless sensitive.

[g13] identifies a **KeyClass** for a symmetric encryption key for encrypting data that can later be decrypted by
authorized physicians and other entities that have access to keys of the EHR-PHY **KeyClass**.

[g14] is the closing tag of the **KeyClasses** element.

[g15] is the closing tag of the **SymkeyRequest** element.

### 3.8 Response with multiple new symmetric keys

After an SKS server has performed its operations of authenticating the request, identifying the requester,
determining policies that apply to the requester, generating the symmetric encryption keys in conformance to the
defined policies and **KeyClasses** and finally escrowing the symmetric keys securely, it assembles the following
response and returns it to the client.

To reduce the verbosity of the response that includes nine symmetric encryption keys, the SKSML shows only
the details of two symmetric keys and the encapsulating element tags for the remaining seven keys. Regardless
of what the KeyUsePolicy and/or Permissions elements state in those seven keys, the schema for each response
conforms to the specification described in this document. Additionally, the SOAP message, as indicated earlier,
is secured using WSS, but only the actual SKSML content is displayed and discussed here.

```
<h01> <ekmi:SymkeyResponse
</h01>
```

```
<h02> xmlns:ekmi='http://docs.oasis-open.org/ekmi/2008/01'
</h02>
```

```
<h03> xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
</h03>
```

```
<h04> <ekmi:Symkey>
</h04>
```

```
<h05> <ekmi:GlobalKeyID>10514-4-3792</ekmi:GlobalKeyID>
</h05>
```

```
<h06> <ekmi:KeyUsePolicy>
</h06>
```

```
<h07> <ekmi:KeyUsePolicyID>10514-9</ekmi:KeyUsePolicyID>
</h07>
```

```
<h08> <ekmi:PolicyName>DES-EDE KeyUsePolicy for EHR-CDC</ekmi:PolicyName>
</h08>
```
<ekmi:KeyClass>EHR-CDC</ekmi:KeyClass>

<ekmi:KeyAlgorithm>
  http://www.w3.org/2001/04/xmlenc#tripledes-cbc
</ekmi:KeyAlgorithm>

<ekmi:KeySize>192</ekmi:KeySize>

<ekmi:Status>Active</ekmi:Status>

<ekmi:Permissions/>

<ekmi:EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>

<xenc:CipherData>
  <xenc:CipherValue>
    E9zWB/y93hVSzeTL1dQoDxmLNxTuxSffMNwCJmtldIqzQHBnpdQ81g6DKdKCFjJM4aw
    hQnywC95fYyv9h5FdqUiXG0ca8EU871zBoXBjDxjfg1pU8tWtx27STRcR/2fw2ava
    ULWtx27STRcRJMtaGttUxuLlWtx27STRcRpm1/>
  </xenc:CipherValue>
</xenc:CipherData>

<ekmi:Symkey>
  <ekmi:GlobalKeyID>10514-4-379</ekmi:GlobalKeyID>
  <ekmi:KeyUsePolicyID>10514-12</ekmi:KeyUsePolicyID>
  <ekmi:PolicyName>DES-EDE KeyUsePolicy for EHR-CRO</ekmi:PolicyName>
  <ekmi:KeyClass>EHR-CRO</ekmi:KeyClass>
  <ekmi:KeyAlgorithm>
    http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  </ekmi:KeyAlgorithm>
  <ekmi:KeySize>192</ekmi:KeySize>
  <ekmi:Status>Active</ekmi:Status>
  <ekmi:Permissions/>
  <ekmi:PermittedDates>
    <ekmi:StartDate>2008-01-01</ekmi:StartDate>
    <ekmi:EndDate>2009-12-31</ekmi:EndDate>
  </ekmi:PermittedDates>
</ekmi:Symkey>

<xenc:CipherData>
  <xenc:CipherValue>
    qUIqXG0ca8EU871zBoXBjDoDxmLNxTuxSffMNwCJmtldIqzQHBnpdQ81g6DKdKCFjJM4aw
    hQnywC95fYyv9h5FdqUiXG0ca8EU871zBoXBjDxjfg1pU8tWtx27STRcR/2fw2ava
    ULWtx27STRcRJMtaGttUxuLlWtx27STRcRpm1/>
  </xenc:CipherValue>
</xenc:CipherData>

<ekmi:Symkey>
  <ekmi:GlobalKeyID>10514-4-3795</ekmi:GlobalKeyID>
  <ekmi:KeyUsePolicyID>10514-12</ekmi:KeyUsePolicyID>
  <ekmi:PolicyName>DES-EDE KeyUsePolicy for EHR-DEF</ekmi:PolicyName>
  <ekmi:KeyClass>EHR-DEF</ekmi:KeyClass>
  <ekmi:KeyAlgorithm>
    http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  </ekmi:KeyAlgorithm>
  <ekmi:KeySize>192</ekmi:KeySize>
  <ekmi:Status>Active</ekmi:Status>
  <ekmi:Permissions/>
  <ekmi:PermittedDates>
    <ekmi:StartDate>2008-01-01</ekmi:StartDate>
    <ekmi:EndDate>2009-12-31</ekmi:EndDate>
  </ekmi:PermittedDates>
</ekmi:Symkey>

<ekmi:Symkey>
  <ekmi:GlobalKeyID>10514-4-3797</ekmi:GlobalKeyID>
  <ekmi:KeyUsePolicyID>10514-12</ekmi:KeyUsePolicyID>
  <ekmi:PolicyName>DES-EDE KeyUsePolicy for EHR-EMT</ekmi:PolicyName>
  <ekmi:KeyClass>EHR-EMT</ekmi:KeyClass>
  <ekmi:KeyAlgorithm>
    http://www.w3.org/2001/04/xmlenc#tripledes-cbc
  </ekmi:KeyAlgorithm>
  <ekmi:KeySize>192</ekmi:KeySize>
  <ekmi:Status>Active</ekmi:Status>
  <ekmi:Permissions/>
  <ekmi:PermittedDates>
    <ekmi:StartDate>2008-01-01</ekmi:StartDate>
    <ekmi:EndDate>2009-12-31</ekmi:EndDate>
  </ekmi:PermittedDates>
</ekmi:Symkey>
...  

[h67]  <ekmi:KeyClass>EHR-INS</ekmi:KeyClass>
...  

[h70]  <ekmi:GlobalKeyID>10514-4-3799</ekmi:GlobalKeyID>
...  

[h71]  <ekmi:KeyClass>EHR-NUR</ekmi:KeyClass>
...  

[h74]  <ekmi:GlobalKeyID>10514-4-3801</ekmi:GlobalKeyID>
...  

[h75]  <ekmi:KeyClass>EHR-PAT</ekmi:KeyClass>
...  

[h78]  <ekmi:GlobalKeyID>10514-4-3803</ekmi:GlobalKeyID>
...  

[h79]  <ekmi:KeyClass>EHR-PHY</ekmi:KeyClass>

[h01] is the start of the SymkeyResponse element.

[h02] and [h03] identify the namespaces to which this XML conforms, and the location of their XML Schema Definitions (XSD).

[h04] is the start tag of the first Symkey element which contains the symmetric encryption key and related elements.

[h05] identifies the GlobalKeyID (GKID) assigned by the SKS server of this first symmetric key. In this example, the concatenated values of the Domain ID, Server ID and Key ID indicate that the key belongs to the organization represented by the PEN, 10514; it was generated on an SKS server with a Server ID of 4 and is the 3792nd unique key generated on that SKS server.

[h06] is the start of the KeyUsePolicy element that applies just to this symmetric key. This element contains details of the policy to which SKCL implementations must conform when using the symmetric key.

[h07] identifies the unique KeyUsePolicyID (KUPID) which identifies this policy within the SKMS.

[h08] provides a descriptive name for this key-use policy, which is helpful to human readers when identifying this policy.

[h09] identifies the KeyClass to which this symmetric key belongs. In the case of this example, the first symmetric key in the response conforms to the EHR-CDC class which, presumably, might be a key that covers data encrypted for/by the Center for Disease Control (CDC) within an Electronic Health Record (EHR) system. Key-classes are useful to applications that wish to encrypt plaintext with a key that has specific characteristics. The requesting application is expected to know what KeyClass it needs before it asks for a key corresponding to that class.

[h10] is the start tag of the KeyAlgorithm element.
identifies the cryptographic algorithm that this symmetric key must be used with. For this symmetric key example, the algorithm is the Triple Data Encryption Standard (3DES) with Cipher Block Chaining (CBC) padding. The URL is a standard notation for this algorithm and padding as defined within [XMLEncryption].

is the closing tag of the **KeyAlgorithm** element.

specifies the size of the symmetric encryption key in bits. For this Triple-DES key, it is 192-bits.

indicates the **Status** of this **KeyUsePolicy** and whether it is an active policy or not. This is useful in situations where an application may wish to re-use a symmetric key to encrypt related data to the data originally encrypted with the symmetric key. While it is possible for the symmetric key object to be active in the database, it is conceivable that the **KeyUsePolicy** used by the key has changed and the application technically needs to use a new symmetric key to encrypt new data.

is the start – and in this example, the end too - of the **Permissions** element. This element provides a sophisticated mechanism for controlling how, where, when and by which applications symmetric keys be used. While there are many sub-elements within a **Permissions** element, not all **KeyUsePolicy** objects might use all **Permissions** sub-elements.<ekmi:RequestedKeyClass>Payroll-Tax-Class</ekmi:RequestedKeyClass>. The example shown for this symmetric key indicates that there are no other specific restrictions on the use of this symmetric key by authorized client applications; i.e. any authorized client application may use it at any time, on any date, in any location for any purpose.

is the closing tag of the **KeyUsePolicy** element.

and **identify the encryption algorithm used in the **EncryptionMethod** element to encrypt the symmetric encryption key itself, to transport to the requesting client. The symmetric key is encrypted using the PublicKey or the requesting client. The **Algorithm** attribute uses the W3C-specified URLs for identifying the encryption and padding algorithms.

is the start of the **CipherData** element. This element is from the W3C XML Encryption namespace (as identified by the 'xenc' qualifier in the element name).

is the start of the **CipherValue** element. This element contains the Base64-encoded ciphertext of the symmetric encryption key.

is the Base64-encoded ciphertext of the symmetric encryption key.

is the closing tag of the **CipherValue** element.

is the closing tag of the **CipherData** element.

is the closing tag of the first **Symkey** element within this **SymkeyResponse**.

- [h56] represents the second **Symkey** element in this **SymkeyResponse**. The differences in this symmetric key element from the first, can be summarized as follows:

  - [h28] identifies a different **GlobalKeyId** (10514-4-3793) for this symmetric key;
  - [h30] identifies a different **KeyUsePolicy** (10514-12) for this symmetric key;
  - [h32] identifies a different **KeyClass** (EHR-CRO) for this symmetric key;
  - [h38] - [h45] defines a different **Permissions** element for this symmetric key;
  - [h51] - [h53] contains a different **CipherValue** for this symmetric key;

is the container for the third **Symkey** element in this response. For the sake of brevity, all the usual **Symkey** elements have been dispensed with, but the unique **GlobalKeyId** and **KeyClass** are shown to indicate the SKS server's response to the request. In this example, they are 10514-4-3795 and EHR-DEF respectively.

contain the remaining **Symkey** elements of this **SymkeyResponse**. Once again, for brevity, all the details of the **Symkey** elements are dispensed with, except for the unique GKIDs and KeyClasses.

is the closing tag of the **SymkeyResponse** element.
Note that it is possible for a request to contain the same *KeyClass* multiple times; there is no requirement that they need to be unique within a request if an application has a legitimate business need for multiple symmetric keys of the same *KeyClass*. The SKS server will respond with unique symmetric keys, all belonging to the *KeyClass* requested by the client application.

### 3.9 Response with an SKS error

While one hopes that all authorized requesters will get favorable responses from the SKS server, there are situations in which the client application can receive an error to a request for a symmetric key. The following XML shows one example of such an error response. Depending on the type of error, the actual message content might be different.

```
<i01> <ekmi:SymkeyResponse
<i02> xmlns:ekmi='http://docs.oasis-open.org/ekmi/2008/01'
<i03> xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
<i04> <ekmi:SymkeyError
<i05> <ekmi:RequestedGlobalKeyID>10514-2-22</ekmi:RequestedGlobalKeyID
<i06> <ekmi:ErrorCode>SKS-100004</ekmi:ErrorCode
<i07> <ekmi:ErrorMessage>Unauthorized request for key</ekmi:ErrorMessage
<i08> </ekmi:SymkeyError
<i09> </ekmi:SymkeyResponse
```

[i01] is the start of the *SymkeyResponse* element.

[i02] and [i03] identify the namespaces to which this XML conforms, and the location of their XML Schema Definitions (XSD).

[i04] is the start of the *SymkeyError* element, which tells the Symmetric Key Client Library (SKCL) that the request for a symmetric key resulted in an error.

[i05] indicates the *RequestedGlobalKeyID* the client requested. Returning the GKID in the error response allows the SKCL to associate the error message with the requesting application on the client machine.

[i06] is an *ErrorCode* returned by the SKS server. The code may be one of the standard error codes defined in this specification, or may be a vendor-specific error code.

[i07] is the text of the *ErrorMessage*, localized to the region and language of the requesting client application.

[i08] is the closing tag of the *SymkeyError* tag.

[i09] is the closing tag of the *SymkeyResponse* tag.

### 3.10 Response with symmetric keys and errors

When a client application requests multiple symmetric keys, the SKS server may respond in one of three ways. The SKS server may:

i. Return all symmetric keys as requested;

ii. Return no symmetric keys – i.e. it returns all errors;

iii. Return some symmetric keys and some errors.

The following SKSML shows the third case, where the server returns some symmetric keys and errors in response to a request for multiple keys (such as the one shown in Section 3.7 Request for multiple new symmetric keys).

The following SKSML shows the third case, where the server returns some symmetric keys and errors in response to a request for multiple keys (such as the one shown in Section 3.7 Request for multiple new symmetric keys).

In a response that contains a mix of symmetric keys and errors, all symmetric keys precede all errors – i.e. the *SymkeyResponse* element will not consist of *Symkeys* interspersed with *SymkeyErrors* in between; all *Symkeys* (if any) will start from the top of the response till the first *SymkeyError* element.
<ekmi:SymkeyResponse
 xmlns:ekmi='http://docs.oasis-open.org/ekmi/2008/01'
 xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>

  <ekmi:Symkey>
    <ekmi:GlobalKeyID>10514-4-3792</ekmi:GlobalKeyID>
    <ekmi:KeyUsePolicy>
      <ekmi:KeyUsePolicyID>10514-9</ekmi:KeyUsePolicyID>
      <ekmi:PolicyName>DES-EDC Policy for EHR-CDC</ekmi:PolicyName>
      <ekmi:KeyClass>EHR-CDC</ekmi:KeyClass>
      <ekmi:KeyAlgorithm>
        http://www.w3.org/2001/04/xmlenc#tripledes-cbc
      </ekmi:KeyAlgorithm>
      <ekmi:KeySize>192</ekmi:KeySize>
      <ekmi:Status>Active</ekmi:Status>
      <ekmi:Permissions/>
    </ekmi:KeyUsePolicy>
    <ekmi:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <xenc:CipherData>
      <xenc:CipherValue>
        E9zWB/y93hV5ze1LidQoDxmLnTuxSfFMWwCJnt1dIqzQHBnpdQ81g6DkdcFjJhQhywCx9sfyv9h5FdqUIQG0ca8E871zBoXbjDxjfgIpU8tLtx27STxRc2f
        UIWx7STxRcRJMaGxUxULtx27STxRcRPsY=
      </xenc:CipherValue>
    </xenc:CipherData>
  </ekmi:Symkey>

  <ekmi:Symkey>
    <ekmi:GlobalKeyID>10514-4-3793</ekmi:GlobalKeyID>
    <ekmi:KeyUsePolicy>
      <ekmi:KeyUsePolicyID>10514-12</ekmi:KeyUsePolicyID>
      <ekmi:PolicyName>DES-EDC Policy for EHR-CRO</ekmi:PolicyName>
      <ekmi:KeyClass>EHR-CRO</ekmi:KeyClass>
      <ekmi:KeyAlgorithm>
        http://www.w3.org/2001/04/xmlenc#tripledes-cbc
      </ekmi:KeyAlgorithm>
      <ekmi:KeySize>192</ekmi:KeySize>
      <ekmi:Status>Active</ekmi:Status>
      <ekmi:Permissions/>
    </ekmi:KeyUsePolicy>
    <ekmi:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <xenc:CipherData>
      <xenc:CipherValue>
        qUiQXG0ca8E871zBoXbjDoDxmLnTuxSfFMWwCJnt1dIqzQHBnpdQ81g6DkdcFCJhQhywCx9sfyv9h5FdqUIQG0ca8E871zBoXbjDxjfgIpU8tLtx27STxRcRPsY=
        UJow/qimxi8+ huUYJmataGxUxULtx27STxRcRPsY=
      </xenc:CipherValue>
    </xenc:CipherData>
  </ekmi:Symkey>

  <ekmi:Symkey>
    <ekmi:GlobalKeyID>10514-4-3795</ekmi:GlobalKeyID>
    <ekmi:KeyUsePolicy>
      <ekmi:KeyUsePolicyID>10514-14</ekmi:KeyUsePolicyID>
      <ekmi:PolicyName>DES-EDC Policy for EHR-DEF</ekmi:PolicyName>
      <ekmi:KeyClass>EHR-DEF</ekmi:KeyClass>
      <ekmi:KeyAlgorithm>
        http://www.w3.org/2001/04/xmlenc#tripledes-cbc
      </ekmi:KeyAlgorithm>
      <ekmi:KeySize>192</ekmi:KeySize>
      <ekmi:Status>Active</ekmi:Status>
      <ekmi:Permissions/>
    </ekmi:KeyUsePolicy>
    <ekmi:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#rsa-1_5"/>
    <xenc:CipherData>
      <xenc:CipherValue>
        qUiQXG0ca8E871zBoXbjDoDxmLnTuxSfFMWwCJnt1dIqzQHBnpdQ81g6DkdcFCJhQhywCx9sfyv9h5FdqUIQG0ca8E871zBoXbjDxjfgIpU8tLtx27STxRcRPsY=
        UJow/qimxi8+ huUYJmataGxUxULtx27STxRcRPsY=
      </xenc:CipherValue>
    </xenc:CipherData>
  </ekmi:Symkey>
</ekmi:SymkeyResponse>
3.11 Request for a symmetric key-caching policy

When a client application (that has been linked to the SKCL) needs to encrypt sensitive data, it will call an API method within the SKCL for a new symmetric key. After the SKCL has ensured that the application is authorized to make such a request (by verifying that the configured/passed-in credentials can access the cryptographic key-store module on the client containing the PrivateKey used for signing SKSML requests), the SKCL assembles the following SKSML request:

```xml
<ekmi:SymkeyResponse>
  <ekmi:Symkey>
    <ekmi:GlobalKeyID>10514-4-3797</ekmi:GlobalKeyID>
    <ekmi:KeyClass>EHR-EMT</ekmi:KeyClass>
  </ekmi:Symkey>
  ...
  <ekmi:Symkey>
    <ekmi:GlobalKeyID>10514-4-3798</ekmi:GlobalKeyID>
    <ekmi:KeyClass>EHR-HOS</ekmi:KeyClass>
  </ekmi:Symkey>
  ...
  <ekmi:Symkey>
    <ekmi:GlobalKeyID>10514-4-3799</ekmi:GlobalKeyID>
    <ekmi:KeyClass>EHR-INS</ekmi:KeyClass>
  </ekmi:Symkey>
  ...
  <ekmi:Symkey>
    <ekmi:GlobalKeyID>10514-4-3801</ekmi:GlobalKeyID>
    <ekmi:KeyClass>EHR-NUR</ekmi:KeyClass>
  </ekmi:Symkey>
  ...
  ...
  <ekmi:SymkeyResponseError>
    <ekmi:RequestedGlobalKeyID>10514-0-0</ekmi:RequestedGlobalKeyID>
    <ekmi:RequestedKeyClass>EHR-PAT</ekmi:RequestedKeyClass>
    <ekmi:ErrorMessage>Unauthorized request for key</ekmi:ErrorMessage>
  </ekmi:SymkeyResponseError>
  <ekmi:SymkeyResponseError>
    <ekmi:RequestedGlobalKeyID>10514-0-0</ekmi:RequestedGlobalKeyID>
    <ekmi:RequestedKeyClass>EHR-PHY</ekmi:RequestedKeyClass>
    <ekmi:ErrorMessage>Unauthorized request for key</ekmi:ErrorMessage>
  </ekmi:SymkeyResponseError>
</ekmi:SymkeyResponse>
```

[01] - [j76] is no different from the response shown in Section 3.8 Response with multiple new symmetric keys.

[77] - [j82] identifies the first SymkeyError returned by the SKS server. It is not unlike the error described in Section 3.9 Response with an SKS error. However, there is one difference in this element on line [j79]. This element includes the RequestedKeyClass of the request.

[83] - [j88] identifies the second SymkeyError returned by the SKS server. It is similar to the error described in lines [j77] through [j82] including the RequestedKeyClass of the request.

[89] is the closing tag of the SymkeyResponse tag.

### 3.11 Request for a symmetric key-caching policy

When a client application (that has been linked to the SKCL) needs to encrypt sensitive data, it will call an API method within the SKCL for a new symmetric key. After the SKCL has ensured that the application is authorized to make such a request (by verifying that the configured/passed-in credentials can access the cryptographic key-store module on the client containing the PrivateKey used for signing SKSML requests), the SKCL assembles the following SKSML request:
The `KeyCachePolicyRequest` element is very simple, the Web Service Security (WSS) envelope – which provides security for all SKSML messages – expands the size of the message. The same request shown above, is displayed below in its entirety, with its WSS envelope. Please note that some content – such as Base64-encoded binary content - has been reformatted for aesthetics and clarity of the XML elements. The actual elements and data-types have been preserved from actual SKSML messages.

For an interpretation of the XML elements shown below, please refer to [WSS].

For the sake of brevity, this specification will dispense with showing the SOAP envelope and the WSS elements in all other examples, when discussing SKSML.
3.12 Response with a symmetric key-caching policy (1)

After an SKS server has performed its operations of authenticating a KeyCachePolicyRequest, identifying the requester, determining policies that apply to the requester, it assembles the following response and returns it to the client. (The SOAP message, as indicated earlier, is secured using WSS, but only the actual SKSML content is displayed and discussed here).

```xml
```

**KeyCachePolicyResponse**

```xml
<ekmi:KeyCachePolicyID>10514-1</ekmi:KeyCachePolicyID>
<ekmi:PolicyName>No Caching Policy</ekmi:PolicyName>
<ekmi:Description>This policy is for high-risk, always-connected machines on the network, which will never cache symmetric keys locally. This policy never expires (but checks monthly for any updates).</ekmi:Description>
```

3.12 Response with a symmetric key-caching policy (1)
3.13 Response with a symmetric key-caching policy (2)

This is a second example of a key-caching policy response that has additional elements in the policy permitting caching and specify the number of unused and used (for encryption) symmetric keys that may be cached by the client machine.

1100 [m01]  <ekmi:KeyCachePolicyResponse
1101 [m02]     xmlns:ekmi='http://docs.oasis-open.org/ekmi/2008/01'
1102 [m03]     xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'
1103 [m04]  <ekmi:KeyCachePolicy
1105 [m05]    <ekmi:KeyCachePolicyID>10514-17</ekmi:KeyCachePolicyID>
1106 [m06]    <ekmi:PolicyName>Corporate Laptop Key Caching Policy</ekmi:PolicyName>
1107 [m07]    <ekmi:Description>
1108       This policy defines how company-issued laptops will manage symmetric keys used for file/disk encryption in their local cache. This policy must be used by all laptops that use the company EKMI.
1109    </ekmi:Description>
1110 [m08]  </ekmi:KeyCachePolicy>
1111 [m09]  </ekmi:KeyCachePolicyResponse>
<ekmi:KeyCachePolicyResponse>

<ekmi:KeyCachePolicyID>KCPID</ekmi:KeyCachePolicyID>

<ekmi:PolicyName>PolicyName</ekmi:PolicyName>

<ekmi:Description>Description</ekmi:Description>

<ekmi:KeyClass>KeyClass</ekmi:KeyClass>

<ekmi:StartDate>StartDate</ekmi:StartDate>

<ekmi:EndDate>EndDate</ekmi:EndDate>

<ekmi:PolicyCheckInterval>PolicyCheckInterval</ekmi:PolicyCheckInterval>

<ekmi:Status>Status</ekmi:Status>

<ekmi:MaximumKeys>MaximumKeys</ekmi:MaximumKeys>

<ekmi:MaximumDuration>MaximumDuration</ekmi:MaximumDuration>

</ekmi:NewKeysCacheDetail>

</ekmi:KeyCachePolicyResponse>

[m01] is the start of the KeyCachePolicyResponse element.

[m02] and [m03] identify the namespaces to which this XML conforms, and the location of their XML Schema Definitions (XSD).

[m04] is the start of the first – and only - KeyCachePolicy element.

[m05] identifies the KeyCachePolicyID (KCPID) assigned by the SKS server for the key-caching policy being returned.

[m06] provides a descriptive name for this key-cache policy through the PolicyName element, which is helpful to human readers when identifying this policy.

[m07] is the start tag of the Description element.

[m08] - [m10] provides a human-readable description about this key-cache policy.

[m11] is the closing tag of the Description element.

[m12] specifies the KeyClass to which this policy applies. Only keys that belong to this key-class are subject to this caching policy.

[m13] specifies the date and time that this KeyCachePolicy is effective. This is accomplished through a StartDate element. In this example, the policy is effective as of January 01, 2008.

[m14] specifies the date and time that this KeyCachePolicy becomes invalid. This is accomplished through a EndDate element. In this example, the policy expires on December 31, 2008.

[m15] specifies the frequency at which this client must check with the SKS server for updates to the key-caching policy. This is specified in seconds in the PolicyCheckInterval element; in this example it is a monthly interval.

[m16] indicates the Status of this KeyCachePolicy and whether it is an active policy or not.

[m17] is the start of the NewKeysCacheDetail element, which provides details about how many new symmetric keys - that haven't been used for any encryption transactions – may be cached by the client and for how long.

[m18] indicates the maximum number of new (unused) symmetric keys that may be cached by the client. This is specified through the MaximumKeys element. When the client uses a symmetric key, this reduces the number of new symmetric keys. In this case, the SKCL connects to the SKS server (if it is on the network) and requests a new symmetric key to add to its new-key cache.

[m19] indicates the maximum duration that new (unused) symmetric keys may be cached by the client. This is specified through the MaximumDuration element in seconds. If there are any new keys that exceed this duration limit, the SKCL deletes the specific symmetric key and replaces it with a new symmetric key from the SKS server.

[m20] is the closing tag of the NewKeysCacheDetail element.
3.14 Response with multiple symmetric key-caching policies (3)

This is a third example of a key-caching policy response that has multiple key-cache policies that apply to different classes of symmetric keys.

<m21> is the start of the **UsedKeysCacheDetail** element, which provides details about how many used symmetric keys – those that HAVE been used for any encryption transactions – may be cached by the client and for how long.

[m22] indicates the maximum number of used symmetric keys that may be cached by the client through the **MaximumKeys** element. If the client already has the maximum number of used-keys in its cache, using the First-In-First-Out (FIFO) method, it deletes the oldest symmetric key in the cache to replace with the key that transitioned from the ‘new’ to ‘used’ status.

[m23] indicates the maximum duration that used symmetric keys may be cached by the client through the **MaximumDuration** element in seconds. If there are any used keys that exceed this duration limit, the SKCL deletes the specific symmetric key. While this may temporarily reduce the number of used symmetric keys in the cache, the SKCL takes the most conservative position when making this decision.

[m24] is the closing tag of the **UsedKeysCacheDetail** element.

[m25] is the closing tag of the **KeyCachePolicy** element.

[m26] is the closing tag of the **KeyCachePolicyResponse** element.

This is a third example of a key-caching policy response that has multiple key-cache policies that apply to different classes of symmetric keys.

<m01> <ekmi:KeyCachePolicyResponse
    xmlns:ekmi='http://docs.oasis-open.org/ekmi/2008/01'
    xmlns:xenc='http://www.w3.org/2001/04/xmlenc#'>
  <ekmi:KeyCachePolicy>
    <ekmi:KeyCachePolicyID>10514-1</ekmi:KeyCachePolicyID>
    <ekmi:PolicyName>No Caching Policy</ekmi:PolicyName>
    <ekmi:Description>This policy is for high-risk, always-connected machines on the network, which will never cache symmetric keys locally. This policy never expires (but checks monthly for any updates).</ekmi:Description>
    <ekmi:KeyClass>NoCachingClass</ekmi:KeyClass>
    <ekmi:StartDate>2008-01-01T00:00:01.0</ekmi:StartDate>
    <ekmi:EndDate>1969-01-01T00:00:00.0</ekmi:EndDate>
    <ekmi:PolicyCheckInterval>2592000</ekmi:PolicyCheckInterval>
    <ekmi:Status>Active</ekmi:Status>
  </ekmi:KeyCachePolicy>
  <ekmi:KeyCachePolicy>
    <ekmi:KeyCachePolicyID>10514-17</ekmi:KeyCachePolicyID>
    <ekmi:PolicyName>Corporate Laptop Key Caching Policy</ekmi:PolicyName>
    <ekmi:Description>This policy defines how company-issued laptops will manage symmetric keys used for file/disk encryption in their local cache. This policy must be used by all laptops that use the company EKMI.</ekmi:Description>
    <ekmi:KeyClass>LaptopKeysCachingClass</ekmi:KeyClass>
    <ekmi:StartDate>2008-01-01T00:00:01.0</ekmi:StartDate>
    <ekmi:EndDate>2008-12-31T00:00:01.0</ekmi:EndDate>
    <ekmi:PolicyCheckInterval>2592000</ekmi:PolicyCheckInterval>
    <ekmi:Status>Active</ekmi:Status>
    <ekmi:NewKeysCacheDetail>
      <ekmi:MaximumKeys>3</ekmi:MaximumKeys>
      <ekmi:MaximumDuration>7776000</ekmi:MaximumDuration>
    </ekmi:NewKeysCacheDetail>
    <ekmi:UsedKeysCacheDetail>
  </ekmi:KeyCachePolicy>
</ekmi:KeyCachePolicyResponse>
<ekmi:KeyCachePolicy>
  <ekmi:KeyCachePolicyID>10514-17</ekmi:KeyCachePolicyID>
  <ekmi:PolicyName>Corporate Laptop Key Caching Policy</ekmi:PolicyName>
  <ekmi:Description>This policy defines how company-issued laptops will manage symmetric keys used for file/disk encryption in their local cache. This policy must be used by all laptops.</ekmi:Description>
  <ekmi:KeyClass>LaptopKeysCachingClass</ekmi:KeyClass>
  <ekmi:StartDate>2008-01-01T00:00:01.0</ekmi:StartDate>
  <ekmi:EndDate>2008-12-31T00:00:01.0</ekmi:EndDate>
  <ekmi:PolicyCheckInterval>2592000</ekmi:PolicyCheckInterval>
  <ekmi:Status>Active</ekmi:Status>
</ekmi:KeyCachePolicy>

<ekmi:NewKeysCacheDetail>
  <ekmi:MaximumKeys>3</ekmi:MaximumKeys>
  <ekmi:MaximumDuration>7776000</ekmi:MaximumDuration>
</ekmi:NewKeysCacheDetail>

<ekmi:UsedKeysCacheDetail>
  <ekmi:MaximumKeys>3</ekmi:MaximumKeys>
  <ekmi:MaximumDuration>7776000</ekmi:MaximumDuration>
</ekmi:UsedKeysCacheDetail>

<m01>is the start of the KeyCachePolicyResponse element.</m01>

<m02> and <m03> identify the namespaces to which this XML conforms, and the location of their XML Schema Definitions (XSD).

<m04>is the start of the first of three KeyCachePolicy elements in this response.

<m05>identifies the KeyCachePolicyID (KCPID) assigned to this policy by the SKS server. In this example, the concatenated values of the Domain ID and Policy ID indicate that the key belongs to the organization represented by the PEN, 10514; and is the first key-caching policy within the SKMS.

<m06>provides a descriptive name for this key-cache policy through the PolicyName element, which is helpful to human readers when identifying this policy.

<m07>is the start tag of the Description element.

<m08> - <m10> provides a human-readable description about this key-cache policy.

<m11>is the closing tag of the Description element.

<m12>specifies the KeyClass to which this policy applies. Only keys that belong to this key-class are subject to this caching policy.

<m13>specifies the date and time that this KeyCachePolicy is effective. This is accomplished through a StartDate element. In this example, the policy is effective as of January 01, 2008.

<m14>specifies the date and time that this KeyCachePolicy becomes invalid. This is accomplished through a EndDate element. In this example, the use of the UNIX "epoch" date (January 01, 1969) indicates that this policy never expires.

<m15>specifies the frequency at which this client must check with the SKS server for updates to the key-caching policy. This is specified in seconds in the PolicyCheckInterval element; in this example it is a monthly interval.
indicates the Status of this KeyCachePolicy and whether it is an active policy or not.

[m17] is the closing tag of the first KeyCachePolicy element.

[m18] is the start of the second of three KeyCachePolicy elements in this response.

[m19] identifies the KeyCachePolicyID (KCPID) assigned by the SKS server for the key-caching policy being returned.

[m20] provides the descriptive name for this key-cache policy through the PolicyName element.

[m21] is the start tag of the Description element.

[m22] - [m24] provides a human-readable description about this key-cache policy.

[m25] is the closing tag of the Description element.

[m26] specifies the KeyClass to which this policy applies. Only keys that belong to this key-class are subject to this caching policy.

[m27] specifies the date and time that this KeyCachePolicy is effective. This is accomplished through a StartDate element. In this example, the policy is effective as of January 01, 2008.

[m28] specifies the date and time that this KeyCachePolicy becomes invalid. This is accomplished through a EndDate element. In this example, the policy expires on December 31, 2008.

[m29] specifies the frequency at which this client must check with the SKS server for updates to the key-caching policy. This is specified in seconds in the PolicyCheckInterval element; in this example it is a monthly interval.

[m30] indicates the Status of this KeyCachePolicy and whether it is an active policy or not.

[m31] is the start of the NewKeysCacheDetail element, which provides details about how many new symmetric keys – that haven't been used for any encryption transactions – may be cached by the client and for how long.

[m32] indicates the maximum number of new (unused) symmetric keys that may be cached by the client. This is specified through the MaximumKeys element. When the client uses a symmetric key, this reduces the number of new symmetric keys. In this case, the SKCL connects to the SKS server (if it is on the network) and requests a new symmetric key to add to its new-key cache.

[m33] indicates the maximum duration that new (unused) symmetric keys may be cached by the client. This is specified through the MaximumDuration element in seconds. If there are any new keys that exceed this duration limit, the SKCL deletes the specific symmetric key and replaces it with a new symmetric key from the SKS server.

[m34] is the closing tag of the NewKeysCacheDetail element.

[m35] is the start of the UsedKeysCacheDetail element, which provides details about how many used symmetric keys – those that HAVE been used for any encryption transactions – may be cached by the client and for how long.

[m36] indicates the maximum number of used symmetric keys that may be cached by the client through the MaximumKeys element. If the client already has the maximum number of used-keys in its cache, using the First-In-First-Out (FIFO) method, it deletes the oldest symmetric key in the cache to replace with the key that transitioned from the "new" to "used" status.

[m37] indicates the maximum duration that used symmetric keys may be cached by the client through the MaximumDuration element in seconds. If there are any used keys that exceed this duration limit, the SKCL deletes the specific symmetric key. While this may temporarily reduce the number of used symmetric keys in the cache, the SKCL takes the most conservative position when making this decision.

[m38] is the closing tag of the UsedKeysCacheDetail element.

[m39] is the closing tag of the second KeyCachePolicy element.

[m40] is the start of the third KeyCachePolicy element in this response.
identifies the `KeyCachePolicyID` (KCPID) assigned by the SKS server for the key-caching policy being returned.

[m42] provides the descriptive name for this key-cache policy through the `PolicyName` element.

[m43] is the start tag of the `Description` element.

[m44] - [m46] provides a human-readable description about this key-cache policy.

[m47] is the closing tag of the `Description` element.

[m48] specifies the `KeyClass` to which this policy applies. Only keys that belong to this key-class are subject to this caching policy.

[m49] specifies the date and time that this `KeyCachePolicy` is effective.

[m50] specifies the date and time that this `KeyCachePolicy` becomes invalid.

[m51] specifies the frequency at which this client must check with the SKS server for updates to the key-caching policy.

[m52] indicates the `Status` of this `KeyCachePolicy` and whether it is an active policy or not.

[m53] is the start of the `NewKeysCacheDetail` element, which provides details about how many new symmetric keys may be cached by the client and for how long.

[m54] indicates the maximum number of new (unused) symmetric keys that may be cached by the client. This is specified through the `MaximumKeys` element. When the client uses a symmetric key, this reduces the number of new symmetric keys. In this case, the SKCL connects to the SKS server (if it is on the network) and requests a new symmetric key to add to its new-key cache.

[m55] indicates the maximum duration that new (unused) symmetric keys may be cached by the client. This is specified through the `MaximumDuration` element in seconds. If there are any new keys that exceed this duration limit, the SKCL deletes the specific symmetric key and replaces it with a new symmetric key from the SKS server.

[m56] is the closing tag of the `NewKeysCacheDetail` element.

[m57] is the start of the `UsedKeysCacheDetail` element, which provides details about how many used symmetric keys – those that HAVE been used for any encryption transactions – may be cached by the client and for how long.

[m58] indicates the maximum number of used symmetric keys that may be cached by the client through the `MaximumKeys` element. If the client already has the maximum number of used-keys in its cache, using the First-In-First-Out (FIFO) method, it deletes the oldest symmetric key in the cache to replace with the key that transitioned from the “new” to “used” status.

[m59] indicates the maximum duration that used symmetric keys may be cached by the client through the `MaximumDuration` element in seconds. If there are any used keys that exceed this duration limit, the SKCL deletes the specific symmetric key. While this may temporarily reduce the number of used symmetric keys in the cache, the SKCL takes the most conservative position when making this decision.

[m60] is the closing tag of the `UsedKeysCacheDetail` element.

[m61] is the closing tag of the third and final `KeyCachePolicy` element in this response.

[m62] is the closing tag of the `KeyCachePolicyResponse` element.
Appendix A. Acknowledgments

The following individuals have participated in the creation of this specification and are gratefully acknowledged:

Participants:

•
### Appendix B. Revision History

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Author</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRAFT 5</td>
<td>June 17, 2008</td>
<td>Arshad Noor</td>
<td>Initial version</td>
</tr>
</tbody>
</table>
Appendix C. Non-Normative Text