Multi-vendor Key Management
Does it actually work?

Tim Hudson
Cryptsoft Pty Ltd
A standard for interoperable key management exists but what actually happens when you try to use products and key management solutions from multiple vendors?

Does it work?

Are any benefits gained?

Practical experience from implementing the OASIS Key Management Interoperability Protocol (KMIP) and from deploying and interoperability testing multiple vendor implementations of KMIP form the bulk of the material covered.

Guidance will be provided on the key issues you should require that your vendors address, and how to distinguish between simple vendor tick-box approaches to standards conformance and actual interoperable solutions.
Abstract

- Learning Objectives
  - In-depth knowledge of the core of the OASIS KMIP
  - Awareness of requirements for practical interoperability
  - Guidance on the importance of conformance testing
Key Management Standards

- NSA EKMS
- OASIS EKMI
- ANSI X9.24
- IEEE P1619.3
- OASIS KMIP
- IETF KEYPROV

- NIST SP 800-57
- NIST SP 800-130
- NIST SP 800-152
- ISO 11770
Key Management Standards

- NSA EKMS – Electronic Key Management System
- EKMI – Enterprise Key Management Infrastructure
- KMIP – Key Management Interoperability Protocol
- P1619.3 - Standard for Key Management Infrastructure for Cryptographic Protection of Stored Data
- X9.24 – Retail Financial Services Symmetric Key Management
- SP800-57 Recommendation for Key Management (General, Best Practices for Key Management Organizations, Application Specific Key Management Guidance)
- SP800-130 Framework for Designing Cryptographic Key Management Systems
- SP800-152 Profile for US Federal Cryptographic Key Management Systems (CKMS)
- ISO11770 – Key Management (Framework, Mechanism using Symmetric Techniques, Mechanisms using Asymmetric Techniques, Mechanisms based on Weak Secrets)
Standards

- **Established** vendor approach to standards
  - Own it
  - Subvert it
  - Delay it

There are no other choices.
Office of Strategic Services (OSS)

- Insist on doing everything through "channels." Never permit short-cuts to be taken in order to expedite decisions.
- Make "speeches." Talk as frequently as possible and at great length. Illustrate your "points" by long anecdotes and accounts of personal experiences. Never hesitate to make a few appropriate "patriotic" comments.
- When possible, refer all matters to committees, for "further study and consideration." Attempt to make the committees as large as possible - never less than five.
- Bring up irrelevant issues as frequently as possible.
- Haggle over precise wordings of communications, minutes, resolutions.
- Refer back to matters decided upon at the last meeting and attempt to reopen the question of the advisability of that decision.
- Advocate "caution." Be "reasonable" and urge your fellow-conferees to be "reasonable" and avoid haste which might result in embarrassments or difficulties later on.
- Be worried about the propriety of any decision - raise the question of whether such action as is contemplated lies within the jurisdiction of the group or whether it might conflict with the policy of some higher echelon.

http://svn.cacert.org/CAcert/CAcert_Inc/Board/oss/oss_sabotage.html
Key Management Standards

- If you are an established vendor then standardisation of interfaces with your product is simply not in your commercial best interest
- Customers with high barriers to migration to competitors are good business!!!
- Interoperability standards enable:
  - Customer migration
  - Competitive pricing pressure
  - Effective comparison of products
  - Multi-vendor heterogeneous deployments
  - Self-help (radically reduced lucrative professional services revenue)
- Generally
  - No one wants to be first (customers easily picked off)
  - More so no one wants to be last (customers already have migrated)
Key Management Standards

- Problem domain
  - Encryption (and decryption) is easy
  - Key Management is hard

- Solution
  - Make key management a problem some one else has to solve
  - Externalise it from your problem domain
  - Give the customer “freedom of choice”
  - Make it a wire protocol
  - Let specialist vendors argue out the “hard” problem
Key Management Standards

- Problem
  - How do you pick which standard?

- Solution
  - Popularity test - follow the crowd
  - Fashion test – pick your favourite vendor and follow them
  - Simplicity test – weigh the standards
  - Complexity test – run tools over the standards document
  - Taste test - read the standards
KMIP - Vendors

Voting

Product (Public)

Editors

Interop 2010-2013
KMIP – Vendors – Interop Demo

KMIP – Interoperability Demonstrations – RSA Conference USA

RSA 2010

RSA 2011

RSA 2012

RSA 2013
KMIP – Vendors – Plug Fests

KMIP – Interoperability Plug fests

Jan 2012

July 2012

Jan 2013

July 2013
Key Management Standards

- OASIS KMIP 1.0 – Oct 2010
  - Specification: 105 pages
  - Profiles: 16 pages
  - Usage Guide: 44 pages
  - Use Cases (Test Cases): 168 pages

- OASIS KMIP 1.1 – Jan 2013
  - Specification: 164 pages (+56%)
  - Profiles: 39 pages (+143%)
  - Usage Guide: 63 pages (+43%)
  - Test Cases: 513 pages (+205%)

- OASIS KMIP 1.2 – est Jan 2014
  - Specification: 188 pages (+14%)
  - Profiles (multiple): 871 pages (+2133%)
  - Usage Guide: 78 pages (+24%)
  - Test Cases: 880 pages (+70%)
  - Use Cases: 130 pages
KMIP – Transport Level Encoding

Key Client

API

Internal Representation

KMIP Encode

KMIP Decode

Transport*

Key Server

API

Internal Representation

KMIP Encode

KMIP Decode

Transport*

Message Format

* Transport requires a secure communication protocol (e.g. HTTPS, TLS, etc…)

2013 Storage Developer Conference. © Cryptsoft Pty Ltd. All Rights Reserved.
KMIP Specification

- Specification
  - Base Objects
  - Managed Objects
  - Attributes
  - Client-to-Server Operations
  - Server-to-Client Operations

- Message Contents
  - Message Format
  - Authentication
  - Message Encoding
  - Transport
KMIP Core Concepts

- Base and Managed Objects
- Values
- Attributes
- Operations
- Vendor extensions
OASIS KMIP Core Concepts

- Every Object has a “Value”
  - Value is set at object creation
  - Value cannot be changed
  - Value may be “incomplete”
  - Value may be in varying formats
OASIS KMIP Core Concepts

- Every Object has a “Type”
  - Certificate
  - Symmetric Key
  - Public Key
  - Private Key
  - Split Key
  - Template
  - Secret Data
  - Opaque Object
OASIS KMIP Core Concepts

- Every Object has a set of “Attributes”
  - Every attribute has a string name
  - Every attribute has a type
  - May be simple types or complex types
  - Some set by server once and cannot be changed
  - Some set by client once and cannot be changed
  - Most are singleton (only one instance)
  - Server defined non-standard extensions are prefixed with “y-” in their string name
  - Client defined non-standard extensions are prefixed with “x-” in their string name
OASIS KMIP Core Concepts

- KMIP Data Types
  - Structure
  - Integer
  - Long Integer
  - Big Integer
  - Enumeration
  - Boolean
  - Text String
  - Byte String
  - Date Time
  - Interval

32-bit
64-bit
### Table 5: Key Block Object Structure

<table>
<thead>
<tr>
<th>Object</th>
<th>Encoding</th>
<th>REQUIRED</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Block</td>
<td>Structure</td>
<td></td>
</tr>
<tr>
<td>Key Format Type</td>
<td>Enumeration, see 9.1.3.2.3</td>
<td>Yes</td>
</tr>
<tr>
<td>Key Compression Type</td>
<td>Enumeration, see 9.1.3.2.2</td>
<td>No</td>
</tr>
<tr>
<td>Key Value</td>
<td>Byte String: for wrapped Key Value; Structure: for plaintext Key Value, see 2.1.4</td>
<td>Yes</td>
</tr>
<tr>
<td>Cryptographic Algorithm</td>
<td>Enumeration, see 9.1.3.2.12</td>
<td>Yes, MAY be omitted only if this information is available from the Key Value. Does not apply to Secret Data or Opaque Objects. If present, the Cryptographic Length SHALL also be present.</td>
</tr>
<tr>
<td>Cryptographic Length</td>
<td>Integer</td>
<td>Yes, MAY be omitted only if this information is available from the Key Value. Does not apply to Secret Data or Opaque Objects. If present, the Cryptographic Algorithm SHALL also be present.</td>
</tr>
<tr>
<td>Key Wrapping Data</td>
<td>Structure, see 2.1.5</td>
<td>No, SHALL only be present if the key is wrapped.</td>
</tr>
</tbody>
</table>
## KMIP - Attributes

<table>
<thead>
<tr>
<th>Object</th>
<th>Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic Usage Mask</td>
<td>Integer</td>
</tr>
</tbody>
</table>

**Table 67: Cryptographic Usage Mask Attribute**

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHALL always have a value</td>
<td>Yes</td>
</tr>
<tr>
<td>Initially set by</td>
<td>Server or Client</td>
</tr>
<tr>
<td>Modifiable by server</td>
<td>Yes</td>
</tr>
<tr>
<td>Modifiable by client</td>
<td>No</td>
</tr>
<tr>
<td>Deletable by client</td>
<td>No</td>
</tr>
<tr>
<td>Multiple instances permitted</td>
<td>No</td>
</tr>
<tr>
<td>When implicitly set</td>
<td>Create, Create Key Pair, Register, Derive Key, Certify, Re-certify, Re-key</td>
</tr>
<tr>
<td>Applies to Object Types</td>
<td>All Cryptographic Objects, Templates</td>
</tr>
</tbody>
</table>

**Table 68: Cryptographic Usage Mask Attribute Rules**
OASIS KMIP Core Concepts

- Every Object has a set of “Attributes”
  - Every attribute has a string name
  - Every attribute has a type
  - May be simple types or complex types
  - Some set by server once and cannot be changed
  - Some set by client once and cannot be changed
  - Most are singleton (only one instance)
  - Server defined non-standard extensions are prefixed with “y-” in their string name
  - Client defined non-standard extensions are prefixed with “x-” in their string name
## KMIP - Attributes

<table>
<thead>
<tr>
<th>All Objects</th>
<th>Cryptographic Objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Identifier</td>
<td>Cryptographic Algorithm</td>
</tr>
<tr>
<td>Object Type</td>
<td>Cryptographic Length</td>
</tr>
<tr>
<td>Initial Date</td>
<td></td>
</tr>
<tr>
<td>Last Change Date</td>
<td>Cryptographic Usage Mask</td>
</tr>
<tr>
<td>Lease Time</td>
<td>Digest</td>
</tr>
<tr>
<td>State*</td>
<td></td>
</tr>
</tbody>
</table>
KMIP Operations

Protocol Operations

Create
Create Key Pair
Register
Re-key
Derive Key
Certify
Re-certify
Locate
Check
Get
Get Attributes
Get Attribute List
Add Attribute
Modify Attribute
Delete Attribute
Obtain Lease
Get Usage Allocation
Activate
Revoke
Destroy
Archive
Recover
Validate
Query
Cancel
Poll
Notify
Put

Managed Objects

Certificate
Symmetric Key
Public Key
Private Key
Split Key
Template
Secret Data
Opaque Object

Key Block (for keys) or Value (for certificates)

Object Attributes

Unique Identifier
Name
Application Specific Information
Object Type
Cryptographic Algorithm
Cryptographic Length
Cryptographic Parameters
Cryptographic Domain Parameters
Certificate Type
Certificate Identifier
Certificate Issuer
Certificate Subject
Digest
Operation Policy Name
Cryptographic Usage Mask
Lease Time
Usage Limits
State
Initial Date
Activation Date
Process Start Date
Protect Stop Date
Deactivation Date
Destroy Date
Compromise Occurrence Date
Compromise Date
Revocation Reason
Archive Date
Object Group
Link
Contact Information
Last Change Date
Custom Attributes

Object Identification

Lifecycle Information

Extended Information

Client to Server

Server to Client
KMIP - Operations

- Establish
  - Create
  - Register
  - Create Key Pair

- Retrieve
  - Locate
  - Get Attributes
  - Get Attribute List
  - Derive Key
  - Certify
  - Get
KMIP - Operations

- **Manage Usage**
  - Check
  - Obtain Lease

- **Get Usage Allocation**

- **Manage State**
  - Activate
  - Revoke
  - Recover

- **Archive**
- **Destroy**
KMIP - Operations

- Manage Info
  - Add Attribute
  - Modify Attribute
  - Delete Attribute

- Rotate
  - Re-Key
  - Re-Key Key Pair
  - Re-Certify [1.1]

- Server Info
  - Query
  - Poll
  - Cancel

- Client
  - Notify
  - Put

- Other
  - Validate
KMIP – Operations – KMIP 1.2

- Crypto
  - Encrypt
  - Decrypt
  - Sign
  - SignatureVerify

- Split Key
  - Create Split Key
  - Join Split Key

- MAC
  - MACVerify

- HASH

- RNG Seed

- RNG Retrieve
NIST SP 800-57 Key Lifecycle

- **Pre-Operational Phase**:
  - Pre-Activation
    - Generated
    - Ready
  - Active
    - Protect and process
    - Process Only
    - Compromise

- **Operational Phase**:
  - Cryptoperiod expire
  - Deactivated
    - Compromise
  - Compromised

- **Post-Operational Phase**:
  - Destroy
  - Destroyed
  - Destroyed Compromised

- **Destroyed Phase**:
  - Destroy
  - Destroyed

Diagram shows the lifecycle stages and transitions between them.
KMIP – State Transitions

- State Enumeration
  - Pre-Active
  - Active
  - Deactivated
  - Compromised

- Destroyed
  - Destroyed
  - Compromised
KMIP – State Transitions

- Date Attributes
  - Initial Date
  - Destroy Date
  - Last Change Date
  - Archive Date
  - Activation Date
  - Deactivation Date
  - Compromise Date

- Compromise Attributes
  - Compromise Occurrence Date
  - Process Start Date
  - Protect Stop Date
  - Validity Date
  - Original Creation Date
OASIS KMIP Message Encoding

- TTLV encoding (base specification)
  - Tag
  - Type
  - Length
  - Value encoding
KMIP TTLV

Tag: 42 00 2C 05
Type: 00 00 00 04
Length: 00 00 00 0C
Value

Cryptographic Usage Mask = Encrypt | Decrypt
KMIP TTLV

Name Value = SDC2013
Non-TTLV based encoding extensions

- HTTPS (Committee Specification Draft in KMIP 1.2 work packages)
- JSON (Committee Specification Draft in KMIP 1.2 work packages)
- XML (Committee Specification Draft for KMIP 1.2 work packages)
- SOAP [debated, no formal work package]
OASIS KMIP Message Format

- Request Message
  - Request Header
  - 1 or more Batch Items
- Response Message
  - Response Header
  - 1 or more Batch Items
- Batch processing:
  - Able to be asynchronous and out of order
  - Response and request batches matched by Unique Batch Item ID
<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED in Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Request Header</td>
<td>Yes</td>
<td>Structure</td>
</tr>
<tr>
<td>Protocol Version</td>
<td>Yes</td>
<td>See 6.1</td>
</tr>
<tr>
<td>Maximum Response Size</td>
<td>No</td>
<td>See 6.3</td>
</tr>
<tr>
<td>Asynchronous Indicator</td>
<td>No</td>
<td>If present, SHALL be set to True, see 6.7</td>
</tr>
<tr>
<td>Authentication</td>
<td>No</td>
<td>See 6.6</td>
</tr>
<tr>
<td>Batch Error Continuation Option</td>
<td>No</td>
<td>If omitted, then Stop is assumed, see 6.13</td>
</tr>
<tr>
<td>Batch Order Option</td>
<td>No</td>
<td>If omitted, then False is assumed, see 6.12</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>No</td>
<td>See 6.5</td>
</tr>
<tr>
<td>Batch Count</td>
<td>Yes</td>
<td>See 6.14</td>
</tr>
</tbody>
</table>

Table 187: Request Header Structure
<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED in Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Response Header</td>
<td>Yes</td>
<td>Structure</td>
</tr>
<tr>
<td>Protocol Version</td>
<td>Yes</td>
<td>See 6.1</td>
</tr>
<tr>
<td>Time Stamp</td>
<td>Yes</td>
<td>See 6.5</td>
</tr>
<tr>
<td>Batch Count</td>
<td>Yes</td>
<td>See 6.14</td>
</tr>
</tbody>
</table>

Table 189: Response Header Structure
### Table 188: Request Batch Item Structure

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED in Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Item</td>
<td>Yes</td>
<td>Structure, see 6.15</td>
</tr>
<tr>
<td>Operation</td>
<td>Yes</td>
<td>See 6.2</td>
</tr>
<tr>
<td>Unique Batch Item ID</td>
<td>No</td>
<td>REQUIRED if Batch Count &gt; 1, see 6.4</td>
</tr>
<tr>
<td>Request Payload</td>
<td>Yes</td>
<td>Structure, contents depend on the Operation, see 4 and 5</td>
</tr>
<tr>
<td>Message Extension</td>
<td>No</td>
<td>See 6.16</td>
</tr>
</tbody>
</table>
## OASIS KMIP Message Format

### Table 190: Response Batch Item Structure

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED in Message</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>Batch Item</td>
<td>Yes</td>
<td>Structure, see 6.15</td>
</tr>
<tr>
<td>Operation</td>
<td>Yes, if specified in Request Batch Item</td>
<td>See 6.2</td>
</tr>
<tr>
<td>Unique Batch Item ID</td>
<td>No</td>
<td>REQUIRED if present in Request Batch Item, see 6.4</td>
</tr>
<tr>
<td>Result Status</td>
<td>Yes</td>
<td>See 6.9</td>
</tr>
<tr>
<td>Result Reason</td>
<td>Yes, if Result Status is Failure</td>
<td>REQUIRED if Result Status is Failure, otherwise OPTIONAL, see 6.10</td>
</tr>
<tr>
<td>Result Message</td>
<td>No</td>
<td>OPTIONAL if Result Status is not Pending or Success, see 6.11</td>
</tr>
<tr>
<td>Asynchronous Correlation Value</td>
<td>No</td>
<td>REQUIRED if Result Status is Pending, see 6.8</td>
</tr>
<tr>
<td>Response Payload</td>
<td>Yes, if not a failure</td>
<td>Structure, contents depend on the Operation, see 4and 5</td>
</tr>
<tr>
<td>Message Extension</td>
<td>No</td>
<td>See 6.16</td>
</tr>
</tbody>
</table>
KMIP Formats - HEX

42007801000001204200770100000038420069010000002042006a02000000040000000010000000
42006b02000000040000000000000042000d0200000004000000010000000042000e02000000042000f01000000d8
42005c0500000004000000010000000042007901000000c042005705000000040000000020000000
42009101000000a8420008010000003042000a070000001743727970746f6772617068696320416c
676f726974686d0042000b05000000040000000300000000420008010000003042000a0700000014
43727970746f67726170686963204c656e677468000000042000b02000000040000000800000000
420008010000003042000a070000001843727970746f67726170686963205573616765204d61736b
42000b020000000400000000c00000000
# KMIP Formats – TTLV

<table>
<thead>
<tr>
<th>OFFSET</th>
<th>DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>00000000:</td>
<td>142 00 78 01 00 00 01 20 242 00 77 01 00 00 00 38</td>
</tr>
<tr>
<td>00000100:</td>
<td>342 00 69 01 00 00 00 20 442 00 6a 02 00 00 00 04</td>
</tr>
<tr>
<td>00000200:</td>
<td>00 00 00 01 00 00 00 00 542 00 6b 02 00 00 00 04</td>
</tr>
<tr>
<td>00000300:</td>
<td>00 00 00 00 00 00 00 00 642 00 0d 02 00 00 00 04</td>
</tr>
<tr>
<td>00000400:</td>
<td>00 00 00 01 00 00 00 00 742 00 0f 01 00 00 00 d8</td>
</tr>
<tr>
<td>00000500:</td>
<td>842 00 5c 05 00 00 00 04 00 00 00 01 00 00 00 00</td>
</tr>
<tr>
<td>00000600:</td>
<td>942 00 79 01 00 00 00 c0 A42 00 57 05 00 00 00 04</td>
</tr>
<tr>
<td>00000700:</td>
<td>00 00 00 02 00 00 00 00 842 00 91 01 00 00 00 a8</td>
</tr>
<tr>
<td>00000800:</td>
<td>C42 00 08 01 00 00 00 03 D42 00 0a 07 00 00 00 17</td>
</tr>
<tr>
<td>00000900:</td>
<td>43 72 79 70 74 6f 67 72 61 70 68 69 63 20 41 6c</td>
</tr>
<tr>
<td>00000a00:</td>
<td>67 6f 72 69 74 68 6d 00 E42 00 0b 05 00 00 00 04</td>
</tr>
<tr>
<td>00000b00:</td>
<td>00 00 00 03 00 00 00 00 F42 00 08 01 00 00 00 30</td>
</tr>
<tr>
<td>00000c00:</td>
<td>G42 00 0a 07 00 00 00 14 43 72 79 70 74 6f 67 72</td>
</tr>
<tr>
<td>00000d00:</td>
<td>61 70 68 69 63 20 4c 65 6e 67 74 68 00 00 00 00</td>
</tr>
<tr>
<td>00000e00:</td>
<td>H42 00 0b 02 00 00 00 04 00 00 00 80 00 00 00 00</td>
</tr>
<tr>
<td>00000f00:</td>
<td>I42 00 08 01 00 00 00 30 J42 00 0a 07 00 00 00 18</td>
</tr>
<tr>
<td>00001000:</td>
<td>43 72 79 70 74 6f 67 72 61 70 68 69 63 20 55 73</td>
</tr>
<tr>
<td>00001100:</td>
<td>61 67 65 20 4d 61 73 6b K42 00 0b 02 00 00 00 04</td>
</tr>
<tr>
<td>00001200:</td>
<td>00 00 00 0c 00 00 00 00</td>
</tr>
</tbody>
</table>
KMIP Formats – TTLV

INDEX | TAG | TYPE | LENGTH | VALUE | PAD
--- | --- | --- | --- | --- | ---
1 | 420078 | REQUEST_MESSAGE | 01 | STRUCTURE | 00000120 288
2 | 420077 | REQUEST_HEADER | 01 | STRUCTURE | 00000038 56
3 | 420069 | PROTOCOL_VERSION | 01 | STRUCTURE | 00000020 32
4 | 42006a | PROTOCOL_VERSION_MAJOR | 02 | INTEGER | 00000004 0x00000001 00000000
5 | 42006b | PROTOCOL_VERSION_MINOR | 02 | INTEGER | 00000004 0x00000000 00000000
6 | 42000d | BATCH_COUNT | 02 | INTEGER | 00000004 0x00000000 00000000
7 | 42000f | BATCH_ITEM | 01 | STRUCTURE | 000000d8 216
8 | 42005c | OPERATION | 05 | ENUMERATION | 00000004 00000000 00000000
9 | 420079 | REQUEST_PAYLOAD | 01 | STRUCTURE | 000000c0 192
10 | 420057 | OBJECT_TYPE | 05 | ENUMERATION | 00000004 00000000 00000000
11 | 420091 | TEMPLATE_ATTRIBUTE | 01 | STRUCTURE | 000000a8 168
KMIP Formats – JSON (committee draft)

```
{"tag":"RequestMessage", "value": [
{"tag":"RequestHeader", "value": [
{"tag":"ProtocolVersion", "value": [
{"tag":"ProtocolVersionMajor", "type":"Integer", "value":"0x00000001"},
{"tag":"ProtocolVersionMinor", "type":"Integer", "value":"0x00000000"}
]},
{"tag":"BatchCount", "type":"Integer", "value":"0x00000001"}
]},
{"tag":"BatchItem", "value": [
{"tag":"Operation", "type":"Enumeration", "value":"Create"},
{"tag":"RequestPayload", "value": [
{"tag":"ObjectType", "type":"Enumeration", "value":"SymmetricKey"},
{"tag":"TemplateAttribute", "value": [
{"tag":"Attribute", "value": [
{"tag":"AttributeName", "type":"TextString", "value":"Cryptographic Algorithm"},
{"tag":"AttributeValue", "type":"Enumeration", "value":"AES"}
]},
{"tag":"Attribute", "value": [
{"tag":"AttributeName", "type":"TextString", "value":"Cryptographic Length"},
{"tag":"AttributeValue", "type":"Integer", "value":"0x00000080"}
]},
{"tag":"Attribute", "value": [
{"tag":"AttributeName", "type":"TextString", "value":"Cryptographic Usage Mask"},
{"tag":"AttributeValue", "type":"Integer", "value":"Decrypt|Encrypt"}
]
]}
]]
]}
```
KMIP Formats – XML (committee draft)

```xml
<RequestMessage>
  <RequestHeader>
    <ProtocolVersion>
      <ProtocolVersionMajor type="Integer" value="1"/>
      <ProtocolVersionMinor type="Integer" value="0"/>
    </ProtocolVersion>
    <BatchCount type="Integer" value="1"/>
  </RequestHeader>
  <BatchItem>
    <Operation type="Enumeration" value="Create"/>
    <RequestPayload>
      <ObjectType type="Enumeration" value="SymmetricKey"/>
      <TemplateAttribute>
        <Attribute>
          <AttributeName type="TextString" value="Cryptographic Algorithm"/>
          <AttributeValue type="Enumeration" value="AES"/>
        </Attribute>
        <Attribute>
          <AttributeName type="TextString" value="Cryptographic Length"/>
          <AttributeValue type="Integer" value="128"/>
        </Attribute>
        <Attribute>
          <AttributeName type="TextString" value="Cryptographic Usage Mask"/>
          <AttributeValue type="Integer" value="Decrypt Encrypt"/>
        </Attribute>
      </TemplateAttribute>
    </RequestPayload>
  </BatchItem>
</RequestMessage>
```
Status Check

- Does it work?
  - Yes
- Are any benefits gained?
  - Yes
- Practical experience from implementing KMIP
  - Vendors don’t read specifications
  - Interoperability without actual testing (plug fests) is pointless – it just does not work
  - Vendors get really super sensitive when you puncture their carefully crafted product marketing reality distortion field
```python
def ttlv_encode(ttlv):
    msg = ''
    for item in ttlv:
        ttag,tttype,tlen,tval=item
        if tttype == KMIP_ITEM_TYPE_STRUCTURE:
            tval = ttlv_encode(tval);
        else:
            pad = 0
            if tttype in KMIP_TYPE_IS32:
                pad = 4
                tval = struct.pack('>I',int(tval));
            elif tttype == KMIP_ITEM_TYPE_DATE_TIME:
                if isinstance(tval,datetime.datetime):
                    tval = int(time.mktime(tval.timetuple()))
                    tval -= time.timezone
                    tval = struct.pack('>q',tval);
                else:
                    tval = struct.pack('>q',tval);
            elif tttype in KMIP_TYPE_IS64:
                tval = struct.pack('>q',long(tval));
            else:
                pad = 8 - (len(tval) % 8)
                if pad == 8:
                    pad = 0
                if pad:
                    tval = tval+ttlv_padding[0:pad]
                ttype = ttag << 8 | tttype
                tag_type = ttag << 8 | tttype
                msg += struct.pack('>I',tag_type)+struct.pack('>I',tlen)+tval
    return msg
```
def ttlv_decode(buf):
    msg = []
    while buf:
        tag_type = struct.unpack('>I', buf[0:4])[0]
        ttag = tag_type >> 8:
        ttype = tag_type & 0xff
        tlen = struct.unpack('>I', buf[4:8])[0]
        pad = 0
        if tlen % 8 != 0:
            pad = 8 - (tlen % 8)
        padlen = tlen + pad
        tval = buf[8:8 + tlen]
        if ttype == KMIP_ITEM_TYPE_STRUCTURE:
            msg.append((ttag, ttype, tlen, ttlv_decode(buf[8:8 + padlen])))
        elif ttype == KMIP_ITEM_TYPE_BOOLEAN:
            if ttlv_use_python_types:
                l_tval = struct.unpack('>q', tval)[0]
            if l_tval == 1:
                msg.append((ttag, ttype, tlen, True))
            else:
                msg.append((ttag, ttype, tlen, False))
        else:
            l_tval = struct.unpack('>q', tval)[0]
            msg.append((ttag, ttype, tlen, l_tval))
        if ttlv_use_python_types:
            dt1 = datetime.datetime.fromtimestamp(l_tval, ttlv_tz_utc)
            msg.append((ttag, ttype, tlen, dt1))
        else:
            msg.append((ttag, ttype, tlen, l_tval))
    return msg
Status Check

- Practical experience from implementing KMIP (cont)
  - TTLV encode/decode < 0.5% of a KMIP SDK yet vendors still get this very basic bit wrong!
  - Test cases, test cases, test cases
  - Profiles are critically important
  - Formal conformance testing by a trusted third party vendor is essential
Guidance covering the key issues to require that your vendors address

- Who have you tested with?
- What profiles do you support?
- Who else (other than yourself) can attest to your conformance to the specification?
- Which competitor products do you interoperate with?
- What are you contributing to the specification?
- Do your own company products use the specification or do you continue to use a vendor proprietary protocol?
- How many customers have you migrated off your own proprietary protocol?
<Operation type="Enumeration" value="Create"/>
<RequestPayload>
<ObjectType type="Enumeration" value="SymmetricKey"/>
<TemplateAttribute>
  <Attribute>
    <AttributeName type="TextString" value="Cryptographic Algorithm"/>
    <AttributeValue type="Enumeration" value="AES"/>
  </Attribute>
  <Attribute>
    <AttributeName type="TextString" value="Cryptographic Length"/>
    <AttributeValue type="Integer" value="256"/>
  </Attribute>
  <Attribute>
    <AttributeName type="TextString" value="Cryptographic Usage Mask"/>
    <AttributeValue type="Integer" value="Decrypt Encrypt"/>
  </Attribute>
</TemplateAttribute>
</RequestPayload>
<Operation type="Enumeration" value="Create"/>
<ResultStatus type="Enumeration" value="Success"/>
<ResponsePayload>
  <ObjectType type="Enumeration" value="SymmetricKey"/>
  <UniqueIdentifier type="TextString"
      value="8d136d97-b085-4ef8-b88d-b4acbf09b1f"/>
</ResponsePayload>
KMIP – Create a Key

Ignoring Request and Response Headers and Footers

<Operation type="Enumeration" value="Get"/>
<RequestPayload>
  <UniqueIdentifier type="TextString"
      value="8d136d97-b085-4ef8-b88d-b4accbf09b1f"/>
</RequestPayload>

Ignoring Request and Response Headers and Footers

```xml
<Operation type="Enumeration" value="Get"/>
<ResultStatus type="Enumeration" value="Success"/>
<ResponsePayload>
    <ObjectType type="Enumeration" value="SymmetricKey"/>
    <UniqueIdentifier type="TextString"
        value="8d136d97-b085-4ef8-b88d-b4accbf09b1f"/>
    <SymmetricKey>
        <KeyBlock>
            <KeyFormatType type="Enumeration" value="Raw"/>
            <KeyValue>
                <KeyMaterial type="ByteString"
                    value="b956aeabe78afcecc782b7c3709e00d34c1b11c2146ee6e6bb9fc015e4ea310f2"/>
            </KeyValue>
            <CryptographicAlgorithm type="Enumeration" value="AES"/>
            <CryptographicLength type="Integer" value="256"/>
        </KeyBlock>
    </SymmetricKey>
</ResponsePayload>
```
Guidance on KMIP

- Figure out the correct Object Type(s) to use
- Figure out which vendors implement which subsets so you know if you can work with them
- Look at what others have already done – breaking new ground is not generally a good idea
- What life-cycle states can you support
  - For many vendors this is nothing, create-once, always use, never re-key
- What other useful context information can be provided
  - Custom attributes are good to use x-WhateverYouWant
- If you have externalised your key management what can your customers do with the key management vendor
  - There has to be a “value add” there – and that requires context – which requires custom attributes – as context is often product specific
Live Demonstration

- Does it actually work?
- Proof is in actually doing it …
  - Moving keys between vendors …
KMIP Conformance Testing

- SNIA Storage Security Industry Forum (SSIF)
- KMIP Conformance Program

- [http://www.snia.org/forums/SSIF/kmip](http://www.snia.org/forums/SSIF/kmip)
  - Read the FAQ linked off there for more details
  - KMIP Conformance testing for KMIP Servers
  - KMIP Conformance testing for KMIP Clients
  - Testing against OASIS KMIP Profiles
KMIP Profiles

- HTTPS Profile
- KMIP JSON Profile
- KMIP XML Profile
- KMIP Symmetric Key Foundry FIPS140 Profile
- KMIP Symmetric Key Life-Cycle Profile
- KMIP Asymmetric Key Life-Cycle Profile
- KMIP Opaque Managed Object Store Profile
- KMIP Symmetric Key Cloud Server Profile (*)
- KMIP Tape Library Profile
- KMIP Storage Array with Self-Encrypting Drives Profile
- KMIP Suite B Profile (*)
KMIP Profiles (KMIP 1.2 specific)

- Baseline Cryptographic Server
- Baseline Cryptographic Client
- RNG Cryptographic Server
- RNG Cryptographic Client
- Advanced Cryptographic Server
- Advanced Crypto Client
Questions?

Tim Hudson
Technical Director
Cryptsoft Pty Ltd

tjh@cryptsoft.com