Multi-vendor Key Management Does it actually work?

Tim Hudson

Cryptsoft Pty Ltd

Abstract

- 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.

Standards

- 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

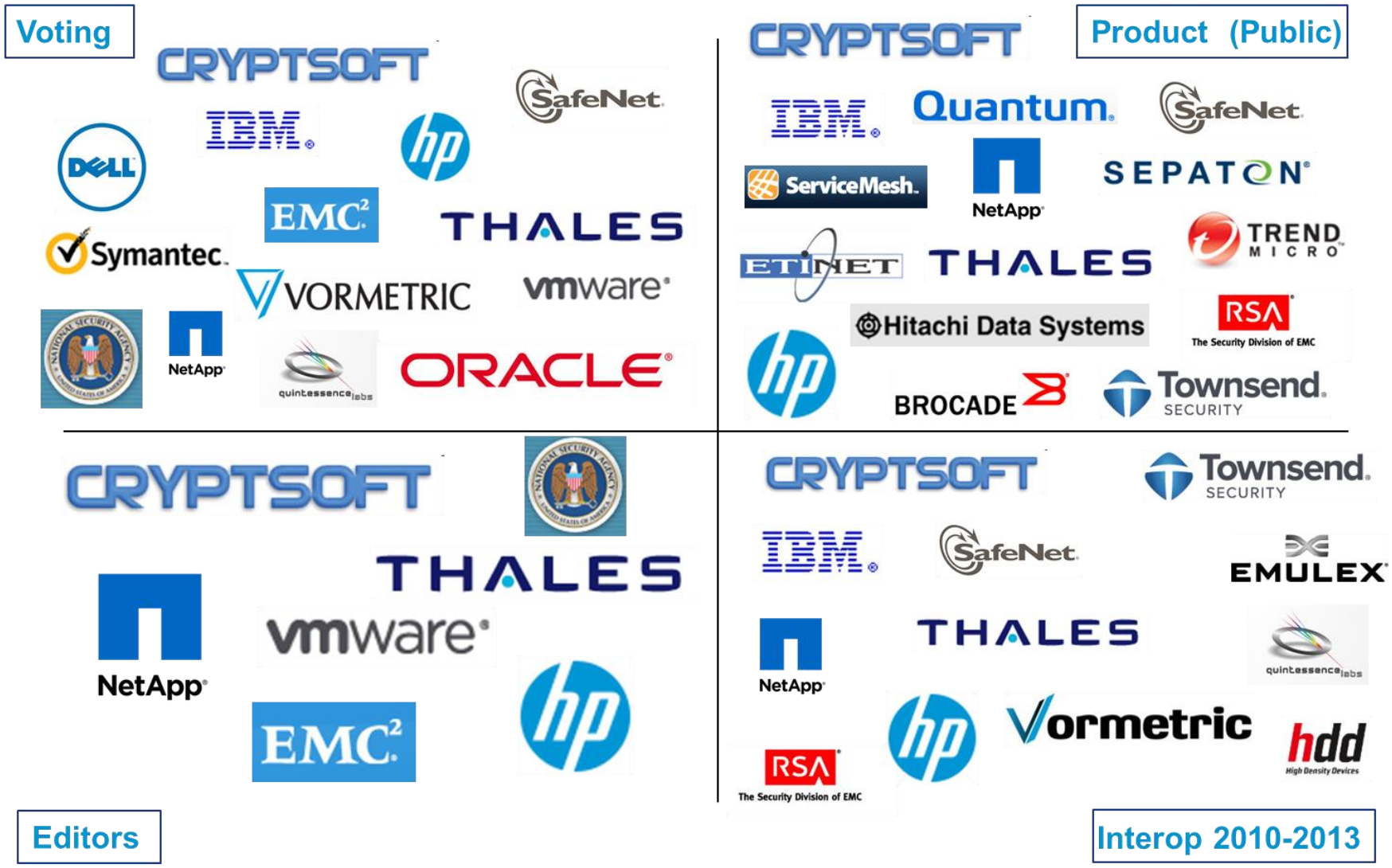
- Solution
  - Make key management a problem some one else has to solve
  - Externalise it from your problem domain
  - Give the customer “freedom of choice”
  - Make it a wire protocol
  - Let specialist vendors argue out the “hard” problem

Key Management Standards

- Problem
  - How do you pick which standard?

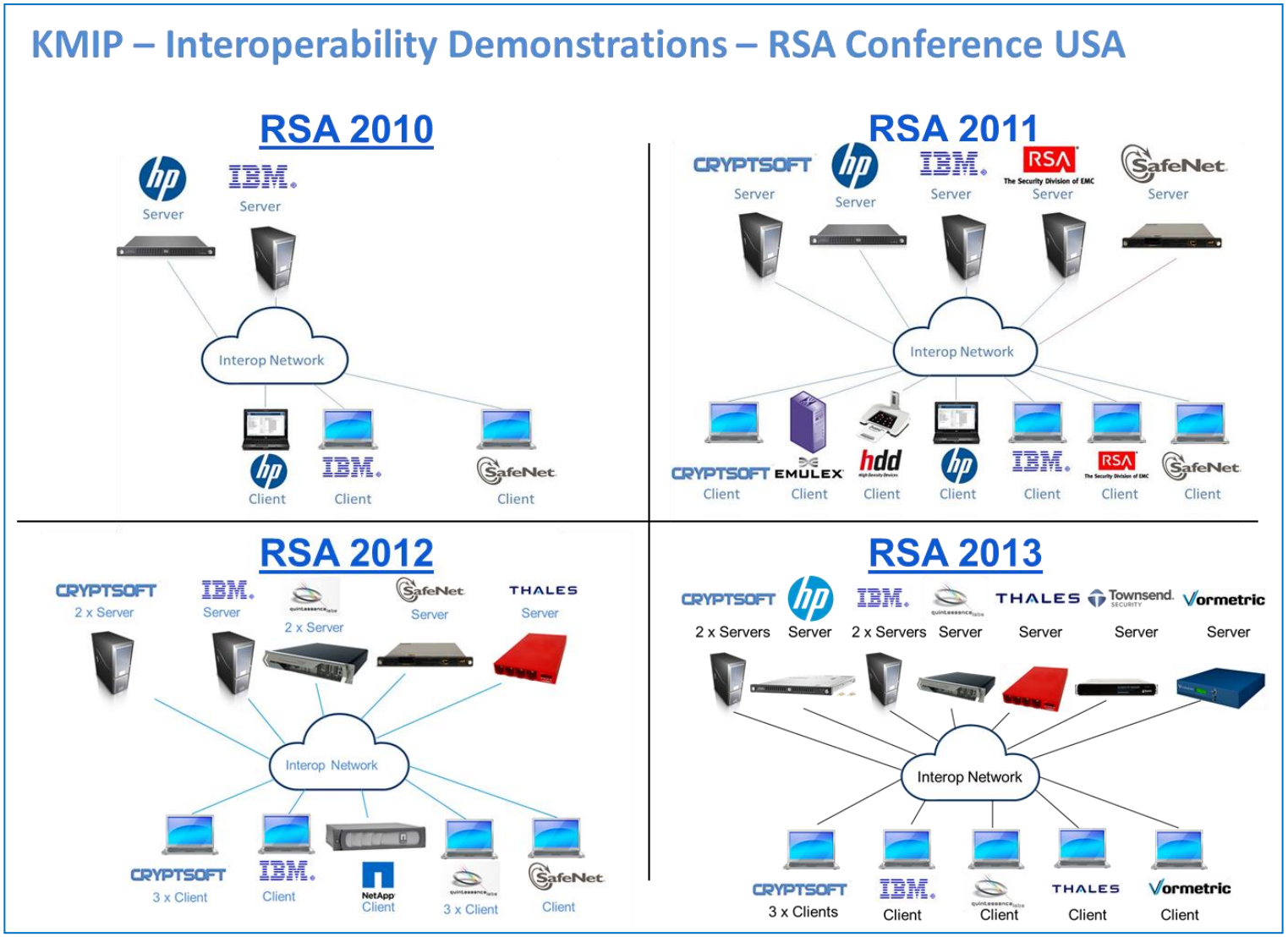
- Solution
  - Popularity test - follow the crowd
  - Fashion test – pick your favourite vendor and follow them
  - Simplicity test – weigh the standards
  - Complexity test – run tools over the standards document
  - Taste test - read the standards

KMIP - Vendors

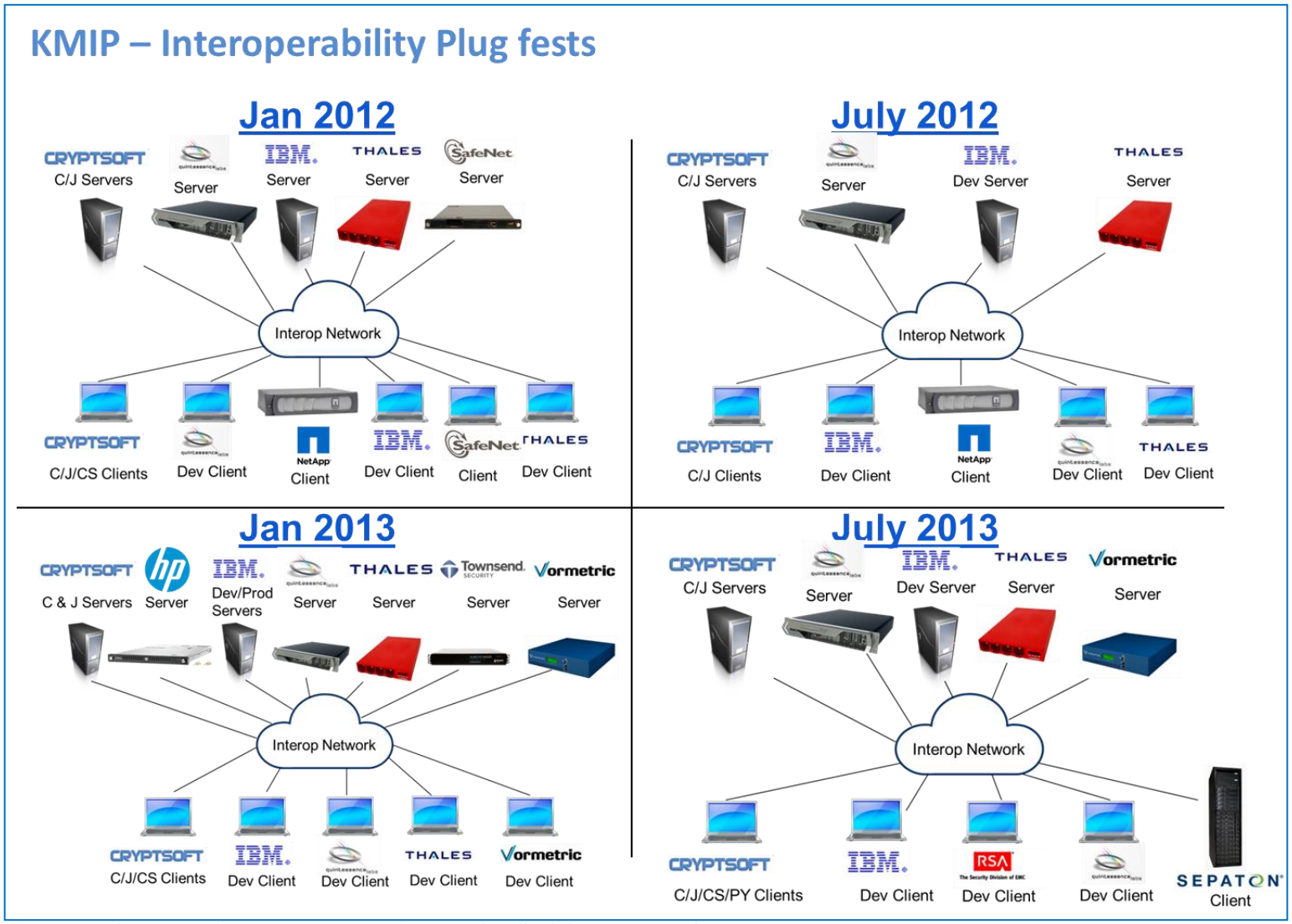


KMIP – Vendors – Interop Demo

KMIP – Interoperability Demonstrations – RSA Conference USA



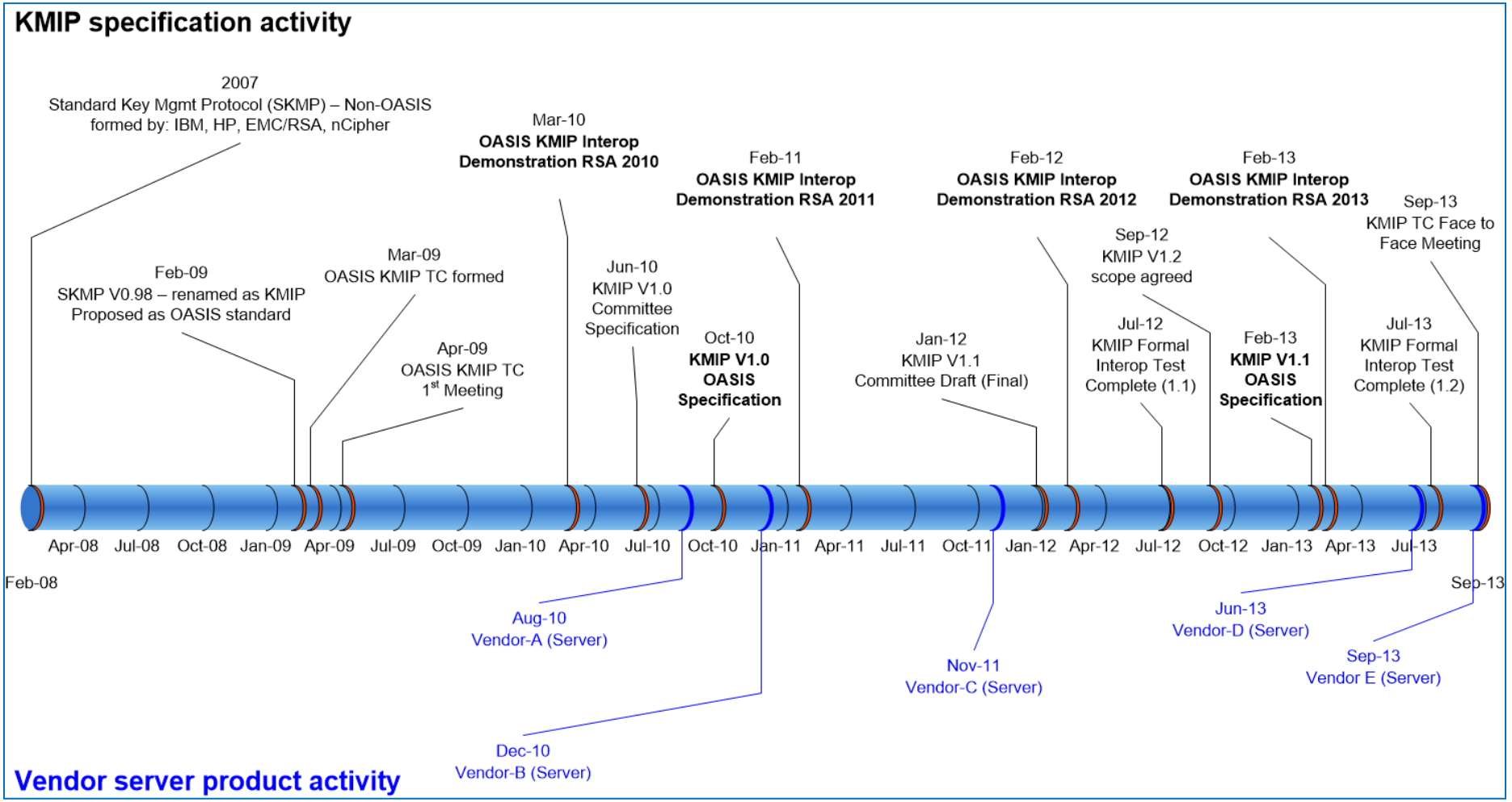
KMIP – Vendors – Plug Fests



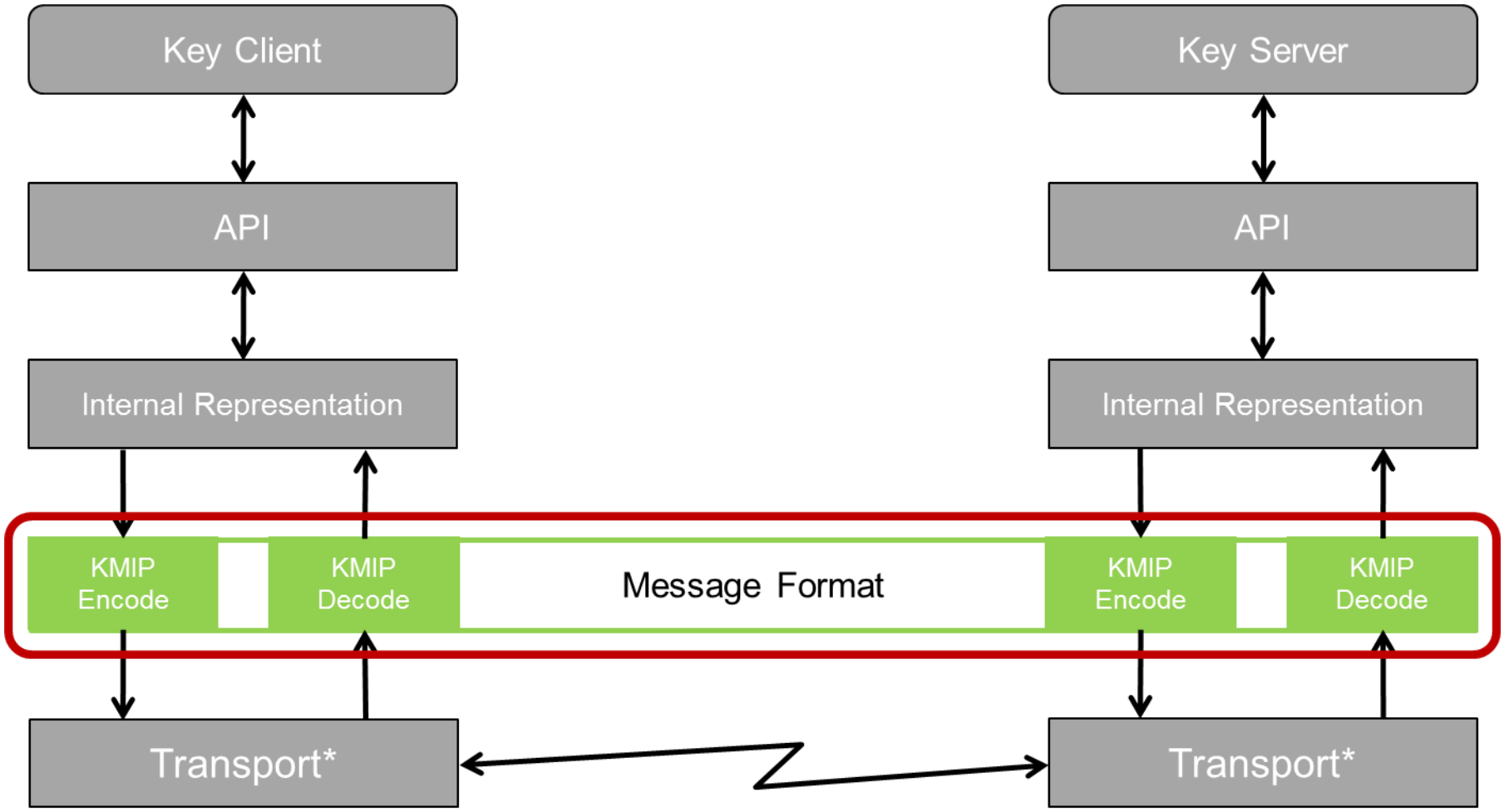
Key Management Standards

❒ 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 History



KMIP – Transport Level Encoding



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

KMIP Specification

- Specification
  - Base Objects
  - Managed Objects
  - Attributes
  - Client-to-Server Operations
  - Server-to-Client Operations
- Message Contents
- Message Format
- Authentication
- Message Encoding
- Transport

OASIS KMIP Core Concepts

- 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

KMIP - Value

Object	Encoding	REQUIRED
Key Block	Structure	
Key Format Type	Enumeration, see 9.1.3.2.3	Yes
Key Compression Type	Enumeration, see 9.1.3.2.2	No
Key Value	Byte String: for wrapped Key Value; Structure: for plaintext Key Value, see 2.1.4	Yes
Cryptographic Algorithm	Enumeration, see 9.1.3.2.12	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.
Cryptographic Length	Integer	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.
Key Wrapping Data	Structure, see 2.1.5	No, SHALL only be present if the key is wrapped.

Table 5: Key Block Object Structure

KMIP - Attributes

Object	Encoding	
Cryptographic Usage Mask	Integer	

Table 67: Cryptographic Usage Mask Attribute

SHALL always have a value	Yes
Initially set by	Server or Client
Modifiable by server	Yes
Modifiable by client	No
Deletable by client	No
Multiple instances permitted	No
When implicitly set	Create, Create Key Pair, Register, Derive Key, Certify, Re-certify, Re-key
Applies to Object Types	All Cryptographic Objects, Templates

Table 68: Cryptographic Usage Mask Attribute Rules

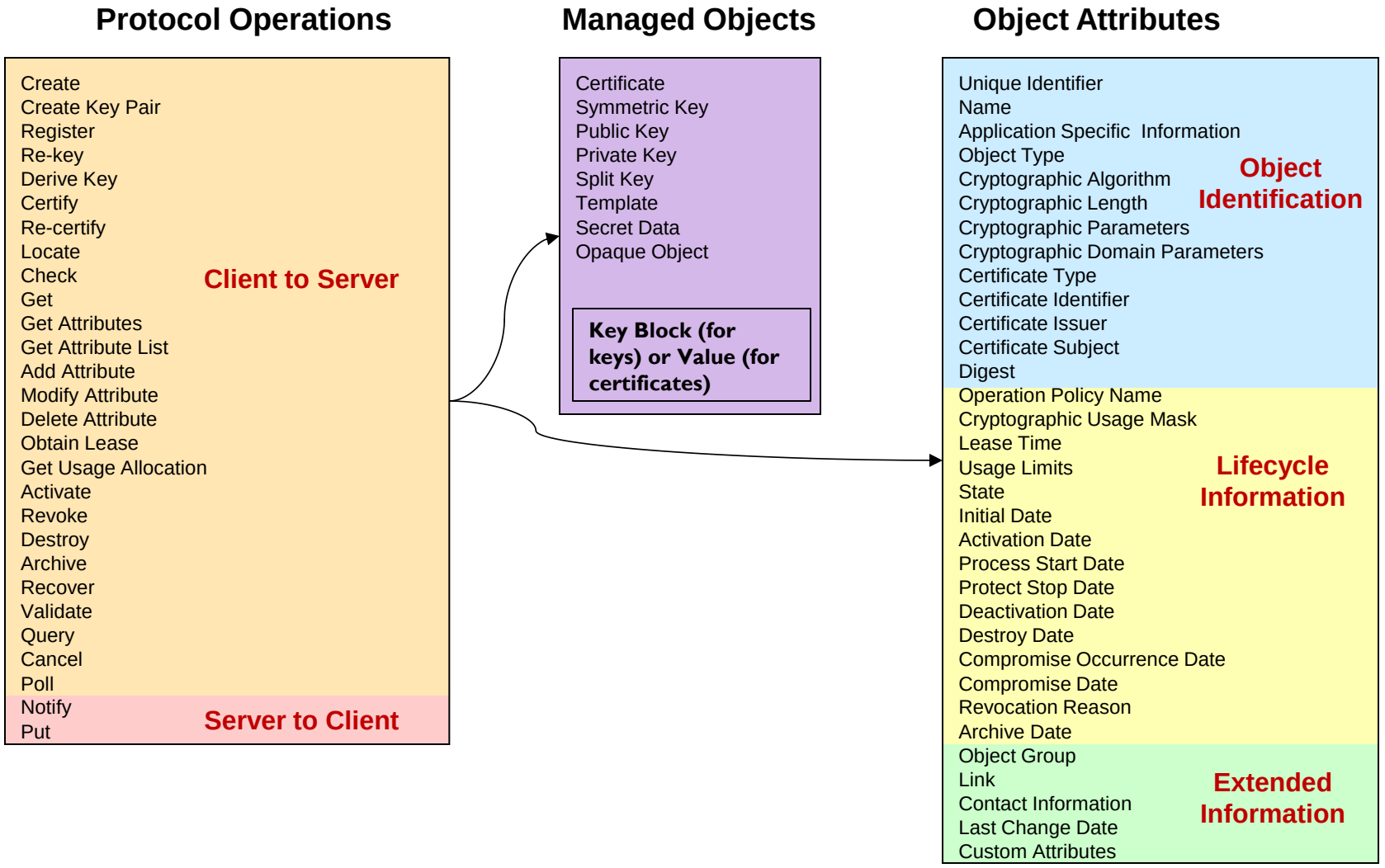
OASIS KMIP Core Concepts

- 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

- All Objects
  - Unique Identifier
  - Object Type
  - Initial Date
  - Last Change Date
  - Lease Time
  - State*
- Cryptographic Objects
  - Cryptographic Algorithm
  - Cryptographic Length
  - Cryptographic Usage Mask
  - Digest

KMIP Operations



KMIP - Operations

- Establish
  - Create
  - Register
  - Create Key Pair
  - Derive Key
  - Certify
- Retrieve
  - Locate
  - Get Attributes
  - Get Attribute List
  - Get

KMIP - Operations

- Manage Usage
  - Check
  - Obtain Lease
  - Get Usage Allocation
- Manage State
  - Activate
  - Revoke
  - Recover
  - Archive
  - Destroy

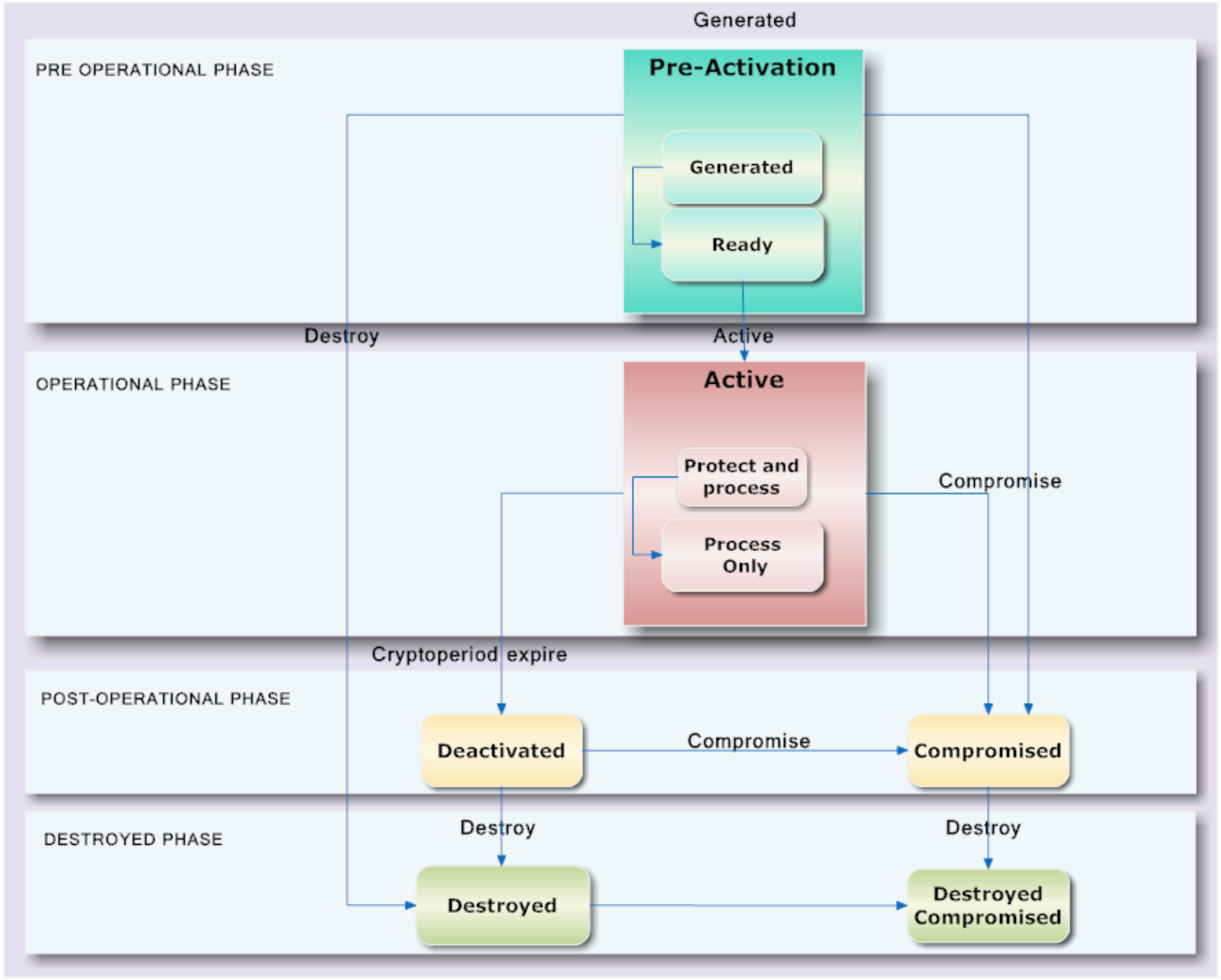
KMIP - Operations

- Manage Info
  - Add Attribute
  - Modify Attribute
  - Delete Attribute
- Server Info
  - Query
  - Poll
  - Cancel
- Rotate
  - Re-Key
  - Re-Key Key Pair
  - Re-Certify [1.1]
- 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



KMIP – State Transitions

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

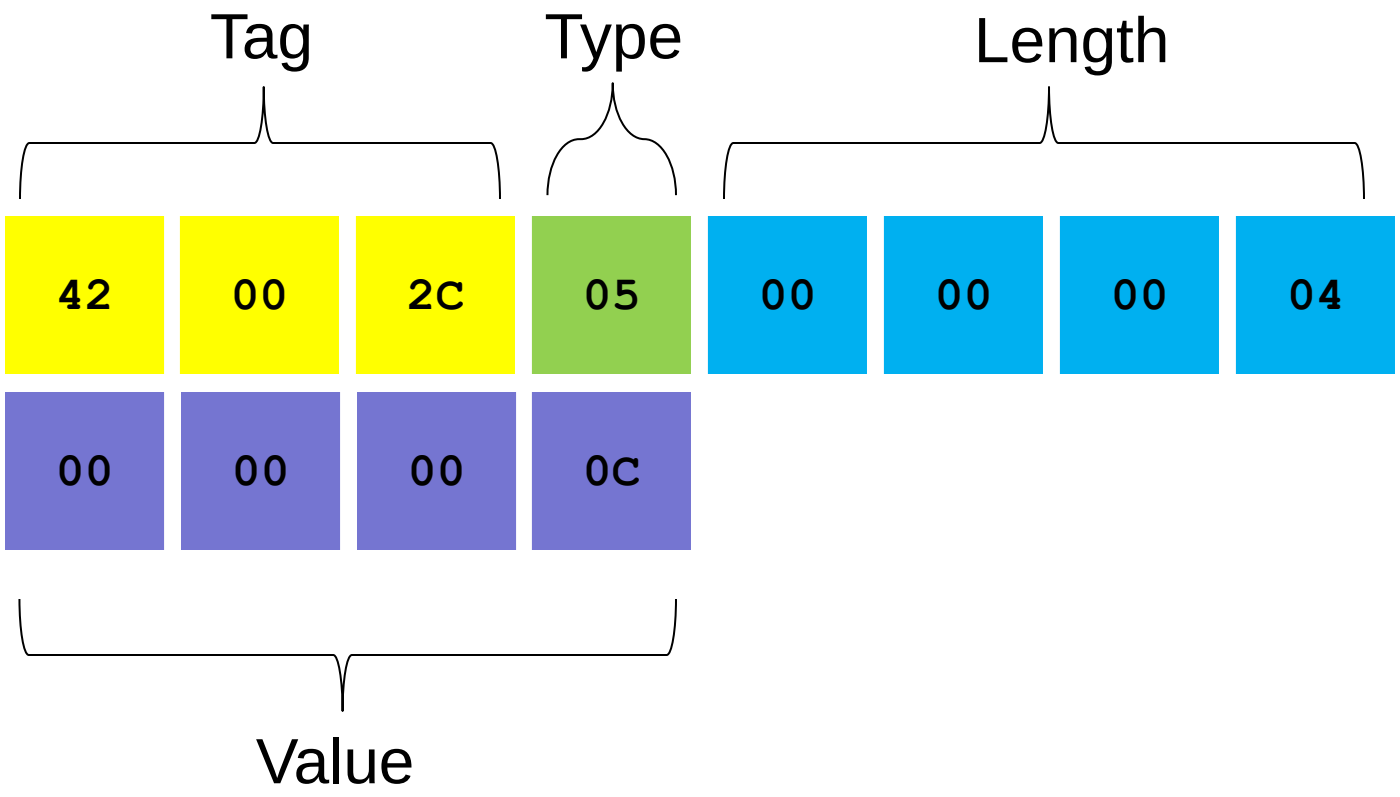
KMIP – State Transitions

- Date Attributes
  - Initial Date
  - Destroy Date
  - Last Change Date
  - Archive Date
  - Activation Date
  - Deactivation Date
  - Compromise Date
  - Compromise Occurrence Date
  - Process Start Date
  - Protect Stop Date
  - Validity Date
  - Original Creation Date

OASIS KMIP Message Encoding

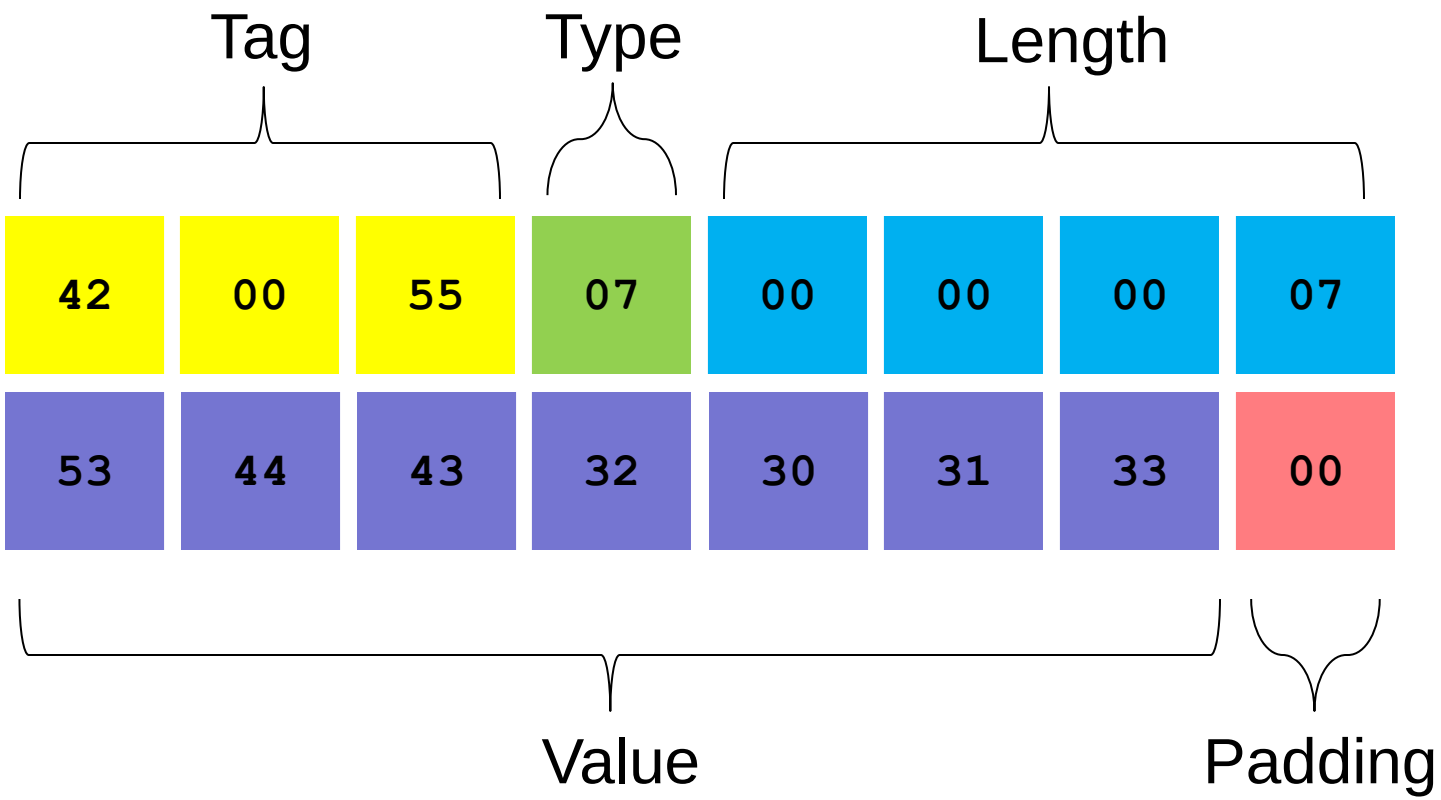
- TTLV encoding (base specification)
  - Tag
  - Type
  - Length
  - Value encoding

KMIP TTLV



Cryptographic Usage Mask = Encrypt | Decrypt

KMIP TTLV



Name Value = SDC2013

OASIS KMIP Message Encoding

- 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

OASIS KMIP Message Format

Request Header		
Object	REQUIRED in Message	Comment
Request Header	Yes	Structure
Protocol Version	Yes	See 6.1
Maximum Response Size	No	See 6.3
Asynchronous Indicator	No	If present, SHALL be set to True, see 6.7
Authentication	No	See 6.6
Batch Error Continuation Option	No	If omitted, then Stop is assumed, see 6.13
Batch Order Option	No	If omitted, then False is assumed, see 6.12
Time Stamp	No	See 6.5
Batch Count	Yes	See 6.14

Table 187: Request Header Structure

OASIS KMIP Message Format

Response Header		
Object	REQUIRED in Message	Comment
Response Header	Yes	Structure
Protocol Version	Yes	See 6.1
Time Stamp	Yes	See 6.5
Batch Count	Yes	See 6.14

Table 189: Response Header Structure

OASIS KMIP Message Format

Request Batch Item		
Object	REQUIRED in Message	Comment
Batch Item	Yes	Structure, see 6.15
Operation	Yes	See 6.2
Unique Batch Item ID	No	REQUIRED if Batch Count > 1, see 6.4
Request Payload	Yes	Structure, contents depend on the Operation, see 4and 5
Message Extension	No	See 6.16

Table 188: Request Batch Item Structure

OASIS KMIP Message Format

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

Table 190: Response Batch Item Structure

KMIP Formats - HEX


```
42007801000001204200770100000038420069010000002042006a020000000400000001000000000
42006b020000000400000000000000042000d020000000400000001000000042000f01000000d8
42005c050000000400000001000000042007901000000c042005705000000040000000200000000
42009101000000a84200080100000030420000a0700000017437279707461f6772617068696320416c
676f726974686d0042000b050000000400000003000000042000801000000304200a0700000014
437279707461f6772617068696320 4c656e67746800000000042000b020000000400000008000000000
42000801000000304200a0700000018437279707461f677261706869632055736167652 04d61736b
42000b020000000400000000c00000000
```

KMIP Formats – TTLV

OFFSET	DATA															
00000000:	[1]42	00	78	01	00	00	01	20	[2]42	00	77	01	00	00	00	38
00000010:	[3]42	00	69	01	00	00	00	20	[4]42	00	6a	02	00	00	00	04
00000020:	00	00	00	01	00	00	00	00	[5]42	00	6b	02	00	00	00	04
00000030:	00	00	00	00	00	00	00	00	[6]42	00	0d	02	00	00	00	04
00000040:	00	00	00	01	00	00	00	00	[7]42	00	0f	01	00	00	00	d8
00000050:	[8]42	00	5c	05	00	00	00	04	00	00	00	01	00	00	00	00
00000060:	[9]42	00	79	01	00	00	00	c0	[A]42	00	57	05	00	00	00	04
00000070:	00	00	00	02	00	00	00	00	[B]42	00	91	01	00	00	00	a8
00000080:	[C]42	00	08	01	00	00	00	30	[D]42	00	0a	07	00	00	00	17
00000090:	43	72	79	70	74	6f	67	72	61	70	68	69	63	20	41	6c
000000a0:	67	6f	72	69	74	68	6d	00	[E]42	00	0b	05	00	00	00	04
000000b0:	00	00	00	03	00	00	00	00	[F]42	00	08	01	00	00	00	30
000000c0:	[G]42	00	0a	07	00	00	00	14	43	72	79	70	74	6f	67	72
000000d0:	61	70	68	69	63	20	4c	65	6e	67	74	68	00	00	00	00
000000e0:	[H]42	00	0b	02	00	00	00	04	00	00	00	80	00	00	00	00
000000f0:	[I]42	00	08	01	00	00	00	30	[J]42	00	0a	07	00	00	00	18
00000100:	43	72	79	70	74	6f	67	72	61	70	68	69	63	20	55	73
00000110:	61	67	65	20	4d	61	73	6b	[K]42	00	0b	02	00	00	00	04
00000120:	00	00	00	0c	00	00	00	00								

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** 4	**00000001** 0x00000001	**00000000** 00000000
[5]	**42006b** PROTOCOL_VERSION_MINOR	**02** INTEGER	**00000004** 4	**00000000** 0x00000000	**00000000** 00000000
[6]	**42000d** BATCH_COUNT	**02** INTEGER	**00000004** 4	**00000001** 0x00000001	**00000000** 00000000
[7]	**42000f** BATCH_ITEM	**01** STRUCTURE	**000000d8** 216		
[8]	**42005c** OPERATION	**05** ENUMERATION	**00000004** 4	**00000001** CREATE	**00000000** 00000000
[9]	**420079** REQUEST_PAYLOAD	**01** STRUCTURE	**000000c0** 192		
[10]	**420057** OBJECT_TYPE	**05** ENUMERATION	**00000004** 4	**00000002** SYMMETRIC_KEY	**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)


```
<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

Code … in Python


```
def ttlv_encode(ttlv):
    msg = ''
    for item in ttlv:
        ttag,ttype,tlen,tval=item
        if ttype == KMIP_ITEM_TYPE_STRUCTURE:
            tval = ttlv_encode(tval);
        else:
            pad = 0
            if ttype in KMIP_TYPE_IS32:
                pad = 4
                tval = struct.pack('>I',int(tval));
            elif ttype == 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 ttype 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]
        tag_type = ttag << 8 | ttype
        msg += struct.pack('>I',tag_type)+struct.pack('>I',tlen)+tval
    return msg
```

Code … in Python


```
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));
        elif ttype == KMIP_ITEM_TYPE_DATE_TIME:
            l_tval = struct.unpack('>q',tval)[0]
            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));
        elif ttype in KMIP_TYPE_IS32:
            i_tval = struct.unpack('>i',tval)[0]
            msg.append((ttag,ttype,tlen,i_tval));
        elif ttype in KMIP_TYPE_IS64:
            l_tval = struct.unpack('>q',tval)[0]
            msg.append((ttag,ttype,tlen,l_tval));
        else:
            msg.append((ttag,ttype,tlen,tval))
        buf = buf[8+padlen:]
    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

Status

- 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?

KMIP – Create a Key

Ignoring Request and Response Headers and Footers

```
<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>
```

KMIP – Create a Key

Ignoring Request and Response Headers and Footers

```
<Operation type="Enumeration" value="Create"/>
<ResultStatus type="Enumeration" value="Success"/>
<ResponsePayload>
  <ObjectType type="Enumeration" value="SymmetricKey"/>
  <UniqueIdentifier type="TextString"
                    value="8d136d97-b085-4ef8-b88d-b4accbf09b1f"/>
</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>
```

KMIP – Create a Key

Ignoring Request and Response Headers and Footers

```
<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="b956aeabe78afcec782b7c3709e00d34c1b11c2146ee6e6bb9fc015e4ea310f2"/>
   </KeyValue>
   <CryptographicAlgorithm type="Enumeration" value="AES"/>
   <CryptographicLength type="Integer" value="256"/>
  </KeyBlock>
 </SymmetricKey>
</ResponsePayload>
```

Status

- 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
  - 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

Questions?

Tim Hudson
Technical Director
Cryptsoft Pty Ltd
tjh@cryptsoft.com