Key Management Interoperability Protocol

Usage Guide

Draft Version 0.98

Last revision February 10, 2009

Table of Contents

1.0 Introduction

This Key Management Interoperability Protocol Usage Guide is intended to complement the Key Management Interoperability Protocol Specification by providing guidance on how to implement the Key Management Interoperability Protocol (KMIP) most effectively to ensure interoperability. In particular, it includes the following guidance:

- Clarification of assumptions and requirements that drive or influence the design of KMIP and implementation of KMIP-compliant key management.
- Definition of required and optional profiles for authentication and communication privacy between KMIP participants (clients and servers).
- Specific recommendations for implementation of particular KMIP functionality.
- Clarification of mandatory and optional capabilities for conformant implementations.
- Functionality considered for inclusion in KMIP V1.0 but deferred to subsequent versions of the standard.

Further assistance for implementing KMIP is provided by the KMIP Use Cases for Proof of Concept Testing document that describes a set of recommended test cases and provides the TTLV (Type/Tag/Length/Value) format for the message exchanges defined by those use cases.

2.0 Assumptions

The section describes assumptions that underlie the KMIP protocol and implementation of clients and servers that utilize the protocol.

2.1 Islands of Trust

Clients are necessarily given key material but they must only use that keying material for the purposes explicitly listed in the delivery payload. Clients that ignore these instructions and use the keys in ways not explicitly allowed by the server are non-compliant. There is no requirement for the key management system, however, to enforce this behavior.

2.2 Message Security

KMIP relies on TLS/SSL to authenticate the client and on the underlying protocol to provide confidentiality, integrity, message authentication and protection against replay attack. KMIP offers a wrapping mechanism for Key Value that does not rely on the transport the messages travel over; this is intended for importing or exporting managed objects.

2.3 State-less Server

The protocol operates on the assumption that the server is state-less, which means that there is no concept of "sessions" inherent in the protocol. State-less server operation is much more reliable and easier to implement, and is consistent with possible implementation scenarios, such as web-services-based servers. This does not mean that the server itself maintains no state, only that the protocol does not require this.

2.4 Extensible Protocol

The protocol provides for "private" or vendor-specific extensions, which allow for differentiation among vendor implementations. However, any objects, attributes and operations included in an implementation must always be implemented as specified, regardless of whether they are optional or required.

2.5 Support for Cryptographic Objects

The protocol supports all reasonable key management system related cryptographic objects. This list currently includes:

- Symmetric Keys
- Split (multi-part) Keys
- Asymmetric Key Pairs and their components
- Digital Certificates
- Derived Keys
- Opaque (non-interpretable) cryptographic objects

2.6 Client-Server Message-based Model

The protocol operates primarily in a client-server, message-based model (the exceptions are the Put and Notify operations). This means that most protocol exchanges are initiated by a client sending a request message to a server, which then sends a reply to the client. The protocol also provides optional mechanisms to allow for unsolicited notification of events to clients, and unsolicited delivery of cryptographic objects to clients, that is, a "push" model. These latter features are optionally supported by servers and clients. Clients must register in order to receive such events/notifications. Registration is implementation specific and not described in the specification.

2.7 Synchronous and Asynchronous Messages

The protocol allows two modes of operation. Synchronous (mandatory) operations are those in which each request from a client sends a request and waits for a response from the server. Polled Asynchronous operations (optional) are those in which the client sends a request, the server responds with a "pending" status and the client polls the server for

the completed response and completion status. Server implementations may choose not to support the Polled Asynchronous feature of the protocol.

2.8 Support for "Intelligent Clients" and "Key Using Devices"

The protocol supports intelligent clients, such as end-user workstations, which are capable of requesting all of the functions of KMIP. It also allows subsets of the protocol, and possible alternate message representations, in order to support less capable devices which only need a subset of the features of KMIP.

2.9 Batched Requests and Responses

The protocol contains a mechanism for sending batched requests and receiving batched responses, to allow for higher throughput on operations that deal with a large number of entities, e. g. requesting dozens or hundreds of keys from a server at one time, and performing operations in a group. An option is provided to continue processing requests after an earlier one fails or to stop processing the remaining requests in the batch. Note that there is no option to treat an entire batch as atomic, that is, if a request in the batch fails then preceding requests in the batch are undone or rolled back. A special ID Placeholder is provided in KMIP to allow related requests in a batch to be pipelined.

2.10 Reliable Message Delivery

The reliable message delivery function is relegated to the transport protocol, and not part of the key management protocol itself.

2.11 Large Responses

For requests that are capable of large responses, a mechanism in the protocol allows a client to specify in a request the maximum allowed size of a response. The server must be able to indicate in a response to such a request that the response would have been too large and therefore not returned.

2.12 Key Life-cycle and Key State

The KMIP Specification describes the key life-cycle model, based on the NIST 800-57 key state definitions, supported by the KMIP protocol. Particular implications of the key life-cycle model in terms of defining time-related attributes of objects are discussed in section 4.5 below.

3.0 KMIP Profiles

This section describes two KMIP profiles. These profiles describe mechanisms by which authentication and communications privacy are established outside KMIP. Both profiles must be supported by any conforming implementation of KMIP.

3.1 SSL/TLS Profile (Mandatory)

Transport Layer Security (TLS) and its predecessor, Secure Sockets Layer (SSL), are cryptographic protocols that provide secure communications for data transfers, using cryptographic mechanisms to provide both authentication of participants and privacy of the communication. SSL 2.0 has known security issues and all current implementations of HTTP/S support more recent protocols. Therefore this profile prohibits the use of SSL 2.0 and recommends SSL 3.1 or TLS 1.0.

In this profile, a KMIP client and server must use SSL/TLS to negotiate a mutually-authenticated connection by using a handshaking procedure.

1. The handshake begins when a client connects to a TLS-enabled server requesting a secure connection, presenting a list of supported cryptographic functions.
2. From this list, the server picks the strongest cipher and hash function that it also supports and notifies the client of the decision.
3. The server sends back its identification in the form of a digital certificate and requests a certificate from the client.
4. The client validates the server certificate. In order to generate the session keys used for the secure connection, the client then encrypts a random number with the server's public key, and sends the result to the server, along with the requested client certificate.
5. From the random number, both parties generate key material for encryption and decryption of all subsequent communication.

Mutual authentication ensures that both the client and the server provide their certificates during the handshake. However, the client certificate used in the SSL session may also be included in any client-initiated KMIP messages between the client and server as the value of the Credentials object in the message, with credential type of certificate. Similar, the server certificate used in the SSL session may be included in any server-initiated KMIP messages between the client and server as the value of the Credentials object in the message, with credential type of certificate.

In SSL and TLS, choices of algorithms are expressed as cipher suites. The following subsections specify cipher suites that are required or discouraged, respectively. The use of any other cipher suite not discussed below is optional.

3.1.1 Mandatory cipher suites

The mandatory cipher suites for the SSL/TLS profile are:

- A TLS-capable instance **must** support `TLS_RSA_WITH_AES_128_CBC_SHA`
- An SSL-capable instance **must** support `SSL_RSA_WITH_AES_128_CBC_SHA`

3.1.2 Discouraged cipher suites

As discussed in "WS-I Basic Security Profile", the cipher suites defined in the SSL and TLS specifications that use anonymous Diffie-Hellman (i. e. those that have *DH_anon* in their symbolic name) are vulnerable to man-in-the-middle attacks. It is recommended that such cipher suites be avoided. This profile recommends against the use of the following cipher suites due to their lack of confidentiality services:

- SSL_RSA_WITH_NULL_SHA
- TLS_RSA_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_MD5
- TLS_RSA_WITH_NULL_MD5

It is also recommended that cipher suites that use 40 or 56 bit keys be avoided, due to their relative ease of compromise through brute-force attack.

See
http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp409.pdf
for a list of NIST-recommended cipher suites.

3.2 HTTPS Profile

Hypertext Transfer Protocol over Secure Socket Layer or https is a URI (Universal Resource Indicator) scheme used to indicate a secure HTTP connection, requiring a different default TCP port (443) and an additional encryption and authentication layer, implemented by SSL/TLS, between HTTP and TCP. The establishment of the trust relationship between the client and server is the same as in the SSL/TLS profile described above.

As in the SSL/TLS profile, the client certificate used in the SSL session must be included in any client-initiated KMIP messages between the client and server as the value of the Credentials object in the message, with credential type of certificate. Similarly, the server certificate used in the SSL session must be included in any server-initiated KMIP messages between the client and server as the value of the Credentials object in the message, with credential type of certificate.

4.0 Usage Guidelines

This section provides guidance on using the functionality described in the Key Management Interoperability Protocol Specification.

4.1 Authentication

As discussed in the Authentication section of the Key Management Interoperability Protocol Specification, a conforming KMIP implementation must support mutual authentication of the client and server as described in the SSL/TLS and HTTPS profiles in Section 3 of this Key Management Interoperability Protocol Usage Guide. Other mechanisms for client and server authentication are possible and optional for KMIP implementations.

The Credential attribute can be used for additional identification of a client. It does not, however, assert that an identity has been authenticated. Therefore it should not be used as an alternative to the transport-level authentication described above.

KMIP implementations that use other vendor-specific mechanisms for authenticating the client and server may also use the Credential attribute to include additional identification information. The Credential attribute may also be used for authentication of the client to the server, but only if the communication is secured to prevent man-in-the-middle attacks that can result in replacement of the credential attribute value.

If an “authentication not successful” error can be returned, it should be returned in preference to any other result status. This prevents status code probing by a client that can't authenticate.

Server decisions regarding which operations to reject if there is insufficiently strong authentication of the client are not specified in the protocol. However, see Section 4.2 for recommendations regarding particular operations for which authentication and authorization are particularly important.

4.2 Authorization for Revoke, Recover, Delete and Archive Operations

Neither authentication nor authorization is handled by the KMIP protocol directly. In particular, the Credential attribute is not guaranteed to be an authenticated identity of the requesting client. However, the mandatory profiles described in this KMIP Usage Guide describe how client identity must be established for KMIP-compliant implementations. This authentication must be performed for all KMIP operations, with the single exception of the Query operation.

Certain operations that may be requested by a client via KMIP, particularly revoke, recover, delete and archive can have a significant impact on the availability of a key, on server performance and on key security. When a server receives a request for one of these operations, it should ensure that the client has established an authenticated identity (see the profiles in Section 3). It should also ensure that client requesting the operation is

an object creator, security officer or other identity authorized to issue the request. It may also require additional authentication to ensure that the object owner or security officer has issued the request. Even with such authentication and authorization, requests for these operations should be considered only a "hint" to the key management system, which may or may not choose to act upon this request.

4.3 Using Notify and Put Operations

The Notify and Put operations are the only operations in the KMIP protocol that are initiated by the server rather than the client. As the functionality provided by these operations can be accomplished through client-initiated requests (using a polling model from the client to request notification, for example), these operations are optional for conforming KMIP implementations. However, they provide a mechanism for optimized communication between KMIP servers and clients and have therefore been included in the KMIP specification.

In using Notify and Put, the following constraints and guidelines must be observed:

- Registration of the client with the server, such that the server knows how to locate the client to which a Notify or Put is being sent and which events for the Notify are supported, is required for Notify and Put operations. However, such registration is outside the scope of the KMIP protocol. This also includes specification of whether a given client supports Put and Notify, and what attributes may be included in a Put for a particular client.
- Communication between the client and the server must be properly authenticated to forestall man-in-the-middle attacks in which the client receives Notify or Put operations from an unauthenticated server. Authentication for a particular client/server implementation must at a minimum be accomplished using one of the mandatory authentication mechanisms. Further strengthening of the client/server communications integrity by means of signed message content and/or wrapped keys is recommended. Attribute values other than "Last Changed Date" should not be included in a Notify to minimize risk of exposure of attribute information.
- In order to minimize possible divergence of key or state information between client and server as a result of server-initiated communication, any client receiving Notify or Put messages must return acknowledgements of these messages to the server. This acknowledgement can be at communication layers below the KMIP layer, such as by using transport-level acknowledgement provided in TCP/IP
- For client devices that are incapable of responding to messages from the server, communication with the server must happen via a proxy entity that communicates with the server, using KMIP, on behalf of the client. Communication between a proxy entity and the client can be secured using other, potentially proprietary mechanisms.

4.4 Usage Allocation

Usage should be allocated and handled carefully since power outages or other types of client failures (crashes) may render allocated usage lost. For example, in the case a key is used for encryption of tape drives, such a loss of the usage allocation information following a client failure during encryption may result in the necessity for the entire tape backup session to be re-encrypted using a different key, if the server cannot allocate more usage. This can be addressed through such approaches as, caching usage allocation information on stable storage at the client, and/or having conservative allocation policies at the server (e.g., by keeping the maximum possible usage allocation per client request moderate). In general, usage allocations should be as small as possible; it is preferable to use multiple smaller allocation requests rather than a single larger request, to minimize the likelihood of unused allocation.

4.5 Key State and Times

The KMIP specification provides a number of time-related attributes, including the following:

- Initial Date: The date and time when the KMIP Managed Cryptographic Object was first created or registered at the server
- Activation Date: The date and time when the KMIP Managed Cryptographic Object may begin to be used
- Process Start Date: The date and time when an KMIP Managed Symmetric or Asymmetric Key Object may begin to be used for process purposes
- Protect Stop Date: The date and time when an KMIP Managed Symmetric or Asymmetric Key Object may no longer be used for protect purposes
- Deactivation Date: The date and time when the KMIP Managed Cryptographic Object may no longer be used for any purpose, except for decryption, signature verification, or unwrapping, but only under extraordinary circumstances and when special permission is granted
- Destroy Date: The date and time when the KMIP Managed Cryptographic Object was destroyed
- Compromise Occurrence Date: The date and time when the KMIP Managed Cryptographic Object was first believed to be compromised
- Compromise Date: The date and time when the KMIP Managed Cryptographic Object is entered into the compromised state
- Archive Date: The date and time when the KMIP Managed Object was placed in Off-Line storage

These attributes apply to all KMIP key-related objects (symmetric keys, asymmetric keys, etc). However, certain of these attributes (such as Initial Date) cannot be specified in template-related objects.
In using these attributes, the following guidelines should be observed:

- As discussed for each of these attributes in Section 3 of the KMIP Specification, a number of these times are set once and cannot be modified by client or server. However, several of the time attributes (particularly Activation Date, Protect Start Date, Process Stop Date and Deactivation Date) can be set by server and/or requested by the client. Coordination of time-related attributes between client and server, therefore, is primarily the responsibility of the server, as it establishes the key and manages its state. However, special conditions related to time-related attributes, governing when the server accepts client modifications to time-related attributes, may be negotiated by policy exchange between the client and server, outside the Key Management Interoperability Protocol.

  In general, state transitions will occur as a result of operational requests. However, clients may need to specify times in the future for such things as activation time, deactivation time and so on.

  It is allowed in KMIP for clients to specify times in the past for such attributes as activation time, deactivation time and so on. This is intended primarily for clients that were disconnected from the server at the time the client performed that operation on a given key.

- It is valid to have a deactivation time when there is no activation time. This means, however, that the key is not yet active even though its deactivation time has been specified. A valid deactivation time must be greater than or equal to activation time.

- Protect stop date must be greater than or equal to process start date. KMIP implementations should consider specifying both these attributes, particularly for symmetric keys, as a key may be needed for decryption (process) long after it is no longer appropriate to use it for encryption of new objects (protect).

- If a delete operation is performed, resulting in the destroy time being set, and the object has not already been deactivated, the deactivation of the object must also be performed prior to delete so that destroy time is greater than or equal to deactivation time. Although not required, it is highly desirable to set other related attributes, such as protect stop time, if they have not already been set. KMIP allows the specification of attributes on a per-client basis, such that a server could maintain different set of attributes for different clients. This flexibility may be necessary in some cases, such as when a server must maintain availability of a key for some clients even after a key moved to

inactive state for most clients. However, such an approach can result in significant inconsistencies regarding the object state from the point of view of all participating clients and should therefore be avoided. It is highly recommended that a server maintain a consistent state for each object, across all clients that have or can request that object.

4.6 Template

A server can maintain different policy templates for different clients. As in the state transitions described above, however, this practice is discouraged.

4.7 Archive Operations

When the Archive operation is performed, it is recommended that an object identifier and a minimal set of attributes be retained within the server for operational efficiency. In such a case, the retained attributes may include Unique Identifier and State.

4.8 Message Extensions

Any number of vendor-specific extensions may be included in the Message Extension optional structure. This allows KMIP implementations to create multiple extensions to the protocol.

4.9 Unique Identifiers

For clients which require unique identifiers in a special form (such as IBM tape drives requiring 12- byte IDs), out-of-band registration/configuration can be used to communicate this requirement to the server.

4.10 Result Message Text

KMIP specifies result status, result reason and result message as normative message contents. For result status and result reason, the enumerations provided in the KMIP specification are the normative values. The values for result message text, on the other hand, are implementation-specific. In consideration of internationalization, it is recommended that any vendor implementation of KMIP provide appropriate language support for return message. How a client specifies the language for Result Messages is outside the scope of the KMIP.

4.11 Certificate Transitions

There is a possible state transition for certificate objects that is not represented by the NIST state transition diagram. Certificate suspension (or certificate hold) allows a certificate to be revoked (certificate appears on a CRL with a revocation reason of "certificateHold") and then reinstated (certificate appears on a CRL with revocation reason of "removeFromCRL"). There is no facility in the SP 800-57 diagram to reinstate a deactivated/compromised object. This state transition can be accomplished in KMIP by returning an object from Revoked to Active state. However, there is no indication in the protocol that an object has undergone such a transition.

.

4.12 Query

Query does not explicitly support client requests to determine what operations require authentication. To determine whether an operation requires authentication, a client must request that operation.

4.13 Canceling Asynchronous Operations

If an asynchronous operation is cancelled, no information is returned in the result code regarding any operations that may have been partially completed. Identification and remediation of partially completed operations is the responsibility of the server.

It is the responsibility of the server to determine when to discard the status of asynchronous operations. The determination of how long a server should retain the status of an asynchronous operation is implementation-dependent and not defined by KMIP.

Once a client has received the status on an asynchronous operation other than “pending”, any subsequent request for status of that operation may return either the same status as in a previous polling request or an “unavailable” response.

4.14 Multi-instance Hash

The Digest attribute contains the output of hashing a managed object such as a key or a certificate. The server always generates the SHA-256 hash when the object is created or generated. KMIP allows multiple digests to be associated with the same managed object. For example, it is common practice for public trusted CAs to publish two digests (often referred to as the fingerprint or the thumbprint) of their certificate one calculated using the SHA-1 algorithm and another using the MD-5 algorithm. In this case, each digest would be calculated by the server using a different hash algorithm.

4.15 Returning Related Objects

The key block is intended to return a single object and associated attributes and other data. For those cases in which multiple related objects are needed by a client, such as the private key and the related certificate required by RACF and JKS, the client should issue multiple Get requests to obtain these related objects.

4.16 Reducing Multiple Requests through Use of Batch

KMIP supports batch operations in order to reduce the number of calls between the client and server for related operations. For example, Locate and Get are likely to be commonly accomplished within a single batch request.

KMIP does not ensure that batch operations are atomic on the server side. But such atomicity can be implemented by the server and in such a case, the client can use the

optional “undo” mode to request roll-back for batch operations implemented as atomic transactions. However, support for “undo” mode is not required by the protocol, nor is there a guarantee that a server that supports “undo” mode has effectively implemented atomic batches, such as by preventing interleaving of batch requests. The use of “undo”, therefore, should be restricted to those cases in which the client can be assured, through mechanisms outside of KMIP, of the server effectively supporting atomicity for batch operations.

4.17 Maximum Message Size

When a server is processing messages in a batch, it should compare the resultant message size after each message with the specified maximum message size. If the message is too large, it should prepare a maximum message size response at that point, rather than continuing with operations in the batch. This increases the client’s ability to understand what operations have and have not been completed.

When processing individual requests within the batch, the server that has encountered a maximum message size error should not return attribute values or other information as part of the response.

4.18 Using Offset in Re-key and Re-certify Operations

Both the re-key and the re-certify operations allow the specification of an offset interval.

The re-key operation allows the client to specify an offset interval for activation of the key. This offset specifies the duration of time between the time the request is made and when the activation of the key will occur. If an offset is specified, all other times for the new key will be determined using the intervals from Activation Date to Process Start Date, Protect Stop Date, etc as defined from the original key.

The re-certify operations allows the client to specify an offset interval that indicates the difference between the Initial Date of the new certificate and the Activation Date of the new certification. As with re-key, all other times for the certificate will be determined using the intervals as defined from the original certificate.

4.19 Locate Queries

Locate queries can be formulated to address any of the following conditions:

- Exact match of a transition. Locate the key(s) that transitioned to a certain state at a specified time (t).
- Range match of a transition. Locate the key(s) that transitioned to a certain state at any time between two specified times (t and t’).

- Exact match of a state at a given instance. Locate the key(s) that are in a certain state at a specified time (t).

- Match of a state during through a time range. Locate the key(s) that are in a certain state at any time between two specified times (t and t').

- Match of a state at some point during a time range. Locate the key(s) that are in a certain state at some time between two specified times (t and t'). In this case, the transition to that state could have happened before the start of the specified time range.

This is accomplished by allowing any date/time attribute to be present either once (for an exact match) or at most twice (for a range match).

For instance, if the state we are interested in is Active, the Locate queries would be the following (corresponding to the bulleted list above):

- Exact match of a transition: Locate (ActivationDate(t))

- Range match of a transition: Locate (ActivationDate(t), ActivationDate(t'))

- Exact match of a state at a given instance: Locate (ActivationDate(0), ActivationDate(t), DeactivationDate(t+1), DeactivationDate(MAX_INT), CompromiseDate(t+1), CompromiseDate(MAX_INT)) This looks for keys that transitioned to an Active state before t, and transitioned to Deactivated or Compromised after t (because we don't want the keys that also transitioned to Deactivated or Compromised before t). The server assumes that keys that don't have a DeactivationDate or CompromiseDate equivalent to MAX_INT (i.e., infinite).

- Match of a state during through a time range: Locate (ActivationDate(0), ActivationDate(t), DeactivationDate(t'+1), DeactivationDate(MAX_INT), CompromiseDate(t'+1), CompromiseDate(MAX_INT))

- Match of a state at some point during a time range: Locate (ActivationDate(0), ActivationDate(t'-1), DeactivationDate(t+1), DeactivationDate(MAX_INT), CompromiseDate(t+1), CompromiseDate(MAX_INT))

The queries would be similar for Initial Date, Deactivation Date, Compromise Date and Destroy Date.

In the case of the Destroyed-Compromise state, there are two dates recorded: Destroy Date and CompromiseDate. For this state, the Locate operation would be expressed as follows:

- Exact match of a transition: Locate (CompromiseDate(t), State(Compromised) and Locate (DestroyDate(t), State(Compromised)) KMIP doesn't support the OR in the Locate request, so two requests must be issued).
- Range match of a transition: Locate (CompromiseDate(t), CompromiseDate(t'), State(Compromised) and Locate (DestroyDate(t), DestroyDate(t'), State(Compromised))
- Exact match of a state at a given instance: Locate (CompromiseDate(0), CompromiseDate(t), DestroyDate(0), DestroyDate(t)) nothing else needed since there is no exit transition.
- Match of a state during through a time range: Locate (CompromiseDate(0), CompromiseDate(t), DestroyDate(0), DestroyDate(t))
- Match of a state at some point during a time range: Locate (CompromiseDate(0), CompromiseDate(t'-1), DestroyDate(0), DestroyDate(t'-1))

4.20 ID Placeholder

The table below shows the ID Placeholder input and output for various operations.

	ID Placeholder input	ID Placeholder output (in case of operation failure, a batch using ID Placeholder must stop)
Create	-	new Object
Create Key Pair	-	new Private Key (the new Public Key can be obtained in the batched via a Locate)
Register	-	new Object
Derive Key	- (because there can be more than one object)	New Symmetric Key
Locate	-	Object
Get	Object	no change

Request Object	Object	no change
Validate	-	-
Get Attributes List/Modify/Add/Delete	Object	no change
Activate	Object	no change
Revoke	Object	no change
Destroy	Object	no change
Archive/Recover	Object	no change
Certify	Public Key	new Certificate
Re-certify	Certificate	new Certificate
Re-key	Symmetric Key	New Symmetric Key
Obtain Lease	Object	no change
Get Usage Allocation	Keys	no change

4.21 Using Wrapped Keys with KMIP

KMIP provides the option to import and get keys in wrapped format. Clients who wish to get a wrapped key from the server are expected to include the Key Wrapping Specification in the Get Request Payload. The wrapping method will identify the type of mechanism used to wrap a key, but will not identify the algorithm or block cipher mode. These will be extracted from the specified Encryption Key or MAC/Signing Key. If a key has multiple Cryptographic Parameters defined, clients can pick one by including the parameters for the specified key in Key Wrapping Specification. Clients also have the option to omit the parameters for the specified key and use the default parameters, i.e. those with the lowest index.

The Key Value includes both the Key Material and if requested in Key Wrapping Specification certain attributes of the key. The Key Value can be encrypted, signed/MACed, or both encrypted and signed. In addition, client's have the option to wrap the key block according to ANSI TR-31 or other standard or vendor-specific key wrapping methods.

It is important to note that if Key Wrapping Specification is included in the Get Request Payload, the Key Value may not necessarily be encrypted. If Wrapping Method is MAC/sign, the returned Key Value will be in plaintext and the Key Wrapping Data will include the MAC or Signature of the Key Value.

4.21.1 Encrypt-only Example Using Wrapped Keys

If a key is to be wrapped with the Cryptographic Usage Mask attribute using AES key wrap, clients must include the following information in the Key Wrapping Specification:

- Wrapping Method: Encrypt
- Encryption Key Information
- Unique Key ID: Key ID of AES key
- Cryptographic Parameters: Block Cipher Mode is AES key wrap (not required if default block cipher mode for wrapping key is AES key wrap)
- Attribute Name: Cryptographic Usage Mask

The AES key must have the AES key wrap set as an allowable Block Cipher Mode. This must be verified by the server.

If the correct data was provided to the server and no conflicts exist, the server will AES key wrap the Key Value, including both the Key Material and the Attribute objects, and return the encrypted blob (octet string) inside the Key Block under Key Value. The Key Wrapping Data includes the same data as specified in the Key Wrapping Specification except for the Attribute Name.

4.21.2 MAC-only Example Using Wrapped Keys

A client that wishes to HMAC SHA-256 the Key Value including a custom attribute must specify the following in Key Wrapping Specification:

- Wrapping Method: MAC/sign
- MAC/Signature Key Information
- Unique Key ID: Key ID of HMAC SHA-256
- Attribute Name: x-Nonce

For HMAC, no crypto parameters need to be specified. The algorithm set for the key already identifies the hash function. The server will create an HMAC over the Key Value if the correct data was provided by the client and no conflicts exist. The Key Value will be returned in plaintext and the Key Block will include the following Key Wrapping Data:

- Wrapping Method: MAC/sign
- MAC/Signature Key Information
- Unique Key ID: Key ID of HMAC SHA-256
- MAC/Signature: HMAC of Key Value

In the example the custom attribute x-Nonce was included to help clients, who are relying on the proxy model, to detect replay attacks. End-clients, who communicate with the key management server, may not support SSL/TLS and may not be able to rely on the message protection mechanisms provided by a security protocol. A custom attribute can be created to hold a random number, counter, nonce, date, or time. The custom attribute needs to be created before requesting the server to return a wrapped key and is recommended to be set if clients frequently wrap/sign the same key with the same wrapping/signing key.

5.0 Conformance

Server implementations of the KMIP protocol must support all objects, attributes, operations and profiles not specified as "optional" in the KMIP Specification in order to be conformant to the specification. Server implementations that do not support objects, attributes, operations and profiles defined as "optional" can claim KMIP conformance, though they may be limited in terms of interoperability with other KMIP implementations.

Client implementations of the KMIP protocol may implement any subset of the KMIP protocol. For example, a client may implement only the Get and Locate operations for symmetric keys. In order to claim conformance, however, such a client must implement all aspects of any elements of the protocol (objects, attributes, operations, profiles) that it claims to support. In the example of Get/Locate support for symmetric keys, therefore, a conforming client implementation must support all required attributes for symmetric keys.

6.0 Deferred KMIP Functionality

The KMIP specification is currently missing items that have been judged candidates for future inclusion in the specification. These items currently include:

- Registration of Clients. This would allow in-band registration and management of clients, which currently can only be registered and/or managed using off-line mechanisms.
- Client-requested specification of additional clients allowed to use a key. This requires coordinated identities between the client and server, and as such is deferred until registration of clients is addressed.
- Registration of Notifications. This would allow clients to specify, using an in-band mechanism, information and events that they wish to be notified of, and what mechanisms should be used for such notifications, possibly including the configuration of pushed cryptographic material. This functionality would assume Registration of Clients as a prerequisite.
- Key Migration. This would standardize migration of keys from one HSM to another, using mechanisms already in the protocol or ones added for this purpose.

- Server to Server key management. This would extend the protocol to support communication between key management servers in different key management domains, for purposes of exporting and importing of cryptographic material and potentially policy information.
- Specification by client of key encoding. KMIP does not currently allow the client to specify the encoding in which a key should be returned; the server returns the key in whatever format it has or otherwise determines it should be returned (such as through out-of-band client specification of encoding). Client specification of encoding may be considered for the future.
- Multiple derived keys. This would allow creation of multiple derived keys from one or more input keys. Note, however, that the current version of KMIP provides the capability to derive multiple keys and initialization vectors by creating a Secret Data object and specifying a cryptographic length equal to the total length of the derived objects.
- XML encoding. Expression of KMIP in XML rather than in type/tag/length/value may be considered for the future.