PyKMIP Documentation

Release 0.8.dev

Peter Hamilton

Sep 15, 2020

Contents

PyKMIP is a Python implementation of the Key Management Interoperability Protocol (KMIP), an OASIS communication standard for the management of objects stored and maintained by key management systems. KMIP defines how key management operations and operation data should be encoded and communicated between client and server applications. Supported operations include the full CRUD key management lifecycle, including operations for managing object metadata and for conducting cryptographic operations. Supported object types include:

- symmetric/asymmetric encryption keys
- passwords/passphrases
- certificates
- opaque data blobs, and more

For more information on KMIP, check out the OASIS KMIP Technical Committee and the OASIS KMIP Documentation.

CHAPTER 1

Installation

You can install PyKMIP via `pip`:

```
$ pip install pykmip
```

See *Installation* for more information.

CHAPTER 2

Layout

PyKMIP provides both client and server functionality, allowing developers to incorporate the full key management lifecycle into their projects. For more information, check out the various articles below.

2.1 Installation

You can install PyKMIP via `pip`:

```
$ pip install pykmip
```

2.1.1 Supported platforms

PyKMIP is tested on Python 2.7, 3.4, 3.5, 3.6, and 3.7 on the following operating systems:

- Ubuntu 12.04, 14.04, and 16.04

PyKMIP also works on Windows and MacOSX however these platforms are not officially supported or tested.

2.1.2 Building PyKMIP on Linux

You can install PyKMIP from source via `git`:

```
$ git clone https://github.com/openkmip/pykmip.git
$ python pykmip/setup.py install
```

If you are on a fresh Linux build, you may also need several additional system dependencies, including headers for Python, OpenSSL, `libffi`, and `libsqlite3`.

Ubuntu

Replace `python-dev` with `python3-dev` if you are using Python 3.0+.

```
$ sudo apt-get install python-dev libffi-dev libssl-dev libsqlite3-dev
```

2.2 Changelog

2.2.1 0.10 - February 25, 2020

- Add server debug logging for message encodings
- Add server Locate filtering for all currently supported attributes
- Add server Locate filtering using offset and maximum item constraints
- Add server cryptography engine support for AES GCM mode
- Add server support for the SplitKey object
- Add client/server support for the ApplicationSpecificInformation attribute
- Add client/server support for the ObjectGroup and Sensitive attributes
- Add client/server support for the DeleteAttribute operation
- Add client/server support for the SetAttribute operation
- Add client/server support for the ModifyAttribute operation
- Add a variety of unit and integration tests to cover all new functionality
- Add new ProxyKmipClient demo scripts to show how to use the new operations
- Add pending deprecation warnings for Python 2.7 and 3.4 due to their EOL
- Update server Locate filtering to return results sorted by creation date
- Update encoding support for SplitKey objects
- Update the Travis CI configuration to better support default Python versions
- Update library and testing dependencies to maintain Python 3.4 support
- Update the library documentation to reflect new features and security details
- Fix a bug with how key pair names are handled by the client for KMIP 2.0

2.2.2 0.9.1 - June 21, 2019

- Fix a bug with Locate attribute handling for KMIP 2.0 clients

2.2.3 0.9 - June 18, 2019

- Add support for Python 3.7
- Add KMIP 2.0 enumerations
- Add a new OrderedEnum subclass to handle sortable enumerations

- Add KMIP 2.0-style attribute handling
- Add utilities to convert between TemplateAttributes and Attributes
- Add utilities to handle bit mask style enumerations
- Add positional argument handling for pytest calls when using tox
- Update the library documentation to include KMIP 2.0 information
- Update client exception handling and logging to simplify debugging
- Update library logging defaults to log at INFO but still support DEBUG
- Update the Travis CI configuration to support testing on Ubuntu 16.04
- Update the Travis CI configuration to output log files on test failures
- Update the server to support KMIP 1.3, 1.4, and 2.0
- Update the PyKMIP clients to support changing their KMIP version
- Update server session logging for authentication failures
- Update the PyKMIP object hierarchy to propagate the KMIP version
- Update the server TLS handshake handling to avoid thread hanging
- Update the Create and Register payloads to support KMIP 2.0
- Update the Locate and CreateKeyPair payloads to support KMIP 2.0
- Update the DeriveKey and GetAttributes payloads to support KMIP 2.0
- Update the GetAttributeList and Query payloads to support KMIP 2.0
- Update server attribute policy to handle KMIP 2.0 deprecated attributes
- Remove escape sequences to comply with Python 3.6 style deprecations
- Fix various deprecation warnings caused by dependency upgrades
- Fix a bug with decoding revocation messages for the Revoke operation
- Fix a bug with specifying the function list in the Query demo script

2.2.4 0.8 - May 18, 2018

- Add Sphinx-based client and server library documentation
- Add server support for third-party authentication systems
- Add client support for the Check operation
- Add client support for the Rekey operation
- Add client support for attestation credentials
- Add a functional test suite for server authentication and access control
- Add payloads for the Archive, Cancel, and GetUsageAllocation operations
- Add payloads for the ObtainLease, Poll, and Recover operations
- Update the server to support group-based operation policies
- Update the server to support live loading of operation policy files
- Update the server to support custom backend database file paths

- Update the server to raise PermissionDenied on access control violations
- Update the client to support custom configuration file paths
- Update the ProxyKmipClient to support custom names for the Register operation
- Update the ProxyKmipClient to set cryptographic usage masks for Derived keys
- Update the README to reference the new documentation
- Update the Travis CI configuration to include building the documentation
- Update the Travis CI configuration to run integration and functional tests
- Remove support for Python 3.3
- Fix a denial-of-service bug by setting the server socket timeout
- Fix a ProxyKmipClient bug with generic cryptographic parameter handling
- Fix a ProxyKmipClient bug with cryptographic usage mask processing

2.2.5 0.7 - November 14, 2017

- Add support for Python 3.6
- Add support for the InitialDate attribute
- Add server support for the GetAttributeList operation
- Add server support for the Locate operation
- Add client and server support for the MAC operation
- Add client and server support for the Revoke operation
- Add client and server support for the Encrypt operation
- Add client and server support for the Decrypt operation
- Add client and server support for the DeriveKey operation
- Add client and server support for the Sign operation
- Add client and server support for the SignatureVerify operation
- Add client and server support for retrieving wrapped keys
- Add client and server support for storing wrapped keys
- Add KMIP 1.4 enumerations
- Add server config option enabling certificate extension checks
- Add server config option defining set of usable TLS ciphers
- Add server config option setting the server log level
- Update the server to enforce checking object state and usage masks
- Update server Locate support to allow object name filtering
- Remove support for Python 2.6
- Fix bug with multithreading support with the SQLite backend
- Fix bug with how open() is mocked in the server test suite
- Fix bug with mismapped polymorphic identity for certificate objects

- Fix bug with socket interrupt handling under Python 3.5
- Fix bug with detached instance errors in the server test suite

2.2.6 0.6 - December 14, 2016

- Add support for Python 3.5
- Add support for the State and OperationPolicyName attributes
- Add server support for the Activate and GetAttributes operations
- Add server support for certificate-based client authentication
- Add server support for object access control via operation policies
- Add server support for loading of user-defined operation policies
- Add client support for the GetAttributes operation
- Update clients to support operation policy names with objects
- Update ProxyKmipClient to support names when creating new objects
- Remove coveralls integration
- Fix bug with early server termination on missing request credential
- Fix bug with closing the client while unconnected to a server
- Fix bug with default values overriding server config file settings
- Fix bug with early server termination on bad client certificates
- Fix bug with deprecated usage of the bandit config file
- Fix bug with ProxyKmipClient registering unset object attributes

2.2.7 0.5 - April 14, 2016

- Add KmipServer server implementation
- Add KmipSession to manage threaded client/server connections
- Add KmipEngine for processing core server application logic
- Add KmipEngine support for CRUD operations for managed objects
- Add SQLAlchemy/SQLite support for KmipEngine data storage
- Add CryptographyEngine component for cryptographic operations
- Add pending deprecation warning for Python 2.6 support
- Add pending deprecation warning for the KMIPServer implementation
- Add support for building Sphinx documentation
- Add support for SQLAlchemy tables to all Pie objects
- Add Python magic methods to Attribute and Name objects
- Add Attribute class unit tests
- Add bin script to run the KmipServer
- Add setup entry points to run the KmipServer

- Update DiscoverVersions demo with optional versions argument
- Update all demo scripts to setup their own logging infrastructure
- Update README with information on the KmipServer implementation
- Remove expired certificate files from the integration test suite
- Remove default package log configuration and configuration file
- Fix bug with Locate payload parsing optional values
- Fix bug with DateTime string tests and move to UTC representation

2.2.8 0.4.1 - December 2, 2015

- Add support for the GetAttributeList operation
- Add integration with Travis CI, Codecov/Coveralls, and Bandit
- Add client/server failover support using multiple IP addresses
- Add additional attribute unit tests
- Update implementations of KMIP primitives
- Reorganize server code to prepare for refactoring
- Remove use of exec when handling library version numbers
- Remove broken server script

2.2.9 0.4 - August 14, 2015

- Add the official Pie API for a simpler KMIP interface
- Add the ProxyKmipClient implementation of the Pie API
- Add key, secret, and opaque objects to the Pie object hierarchy
- Add unit demos for all ProxyKmipClient operations
- Add complete unit and integration test suites for the Pie package
- Add KMIPProxy client support/demos for the Activate and Revoke operations
- Add KMIPProxy client connection timeout support
- Add KMIPProxy integration tests for asymmetric key and secret/opaque objects
- Add improved request error logging for the KMIPServer
- Update README with additional information about the clients and Pie API
- Remove AUTHORS in favor of Git commit history
- Fix bug with dangling file handle when setting __version__
- Fix bug with dangling socket connection upon client destruction

2.2.10 0.3.3 - June 25, 2015

- Add the core ManagedObject class hierarchy for the new Pie API
- Add updated Boolean primitive implementation and test suite
- Add integration tests for symmetric key creation and registration
- Update demo and client logging to log at the INFO level by default
- Update README with improved testing instructions
- Fix bug causing enumerations to be encoded as signed integers
- Fix bug with mismatched EncodingOption tag
- Fix bug with relative path use for version number handling
- Fix bug with Integer primitive breaking on valid long integer values

2.2.11 0.3.2 - June 11, 2015

- Add support for registering and retrieving Certificates
- Update unit demos to work with Certificates
- Reorganize test suite into unit and integration test suites
- Remove old demo scripts
- Fix bug with incorrect KeyMaterialStruct tag
- Fix bug causing infinite recursion with object inheritance

2.2.12 0.3.1 - April 23, 2015

- Add KMIP profile information to the client
- Add support for registering/retrieving SecretData and Opaque objects
- Update the SecretFactory to build Public/PrivateKeys with user data

2.2.13 0.3 - March 14, 2015

- Add client support for the DiscoverVersions and Query operations
- Add client support for the CreateKeyPair and ReKeyKeyPair operations
- Add support for registering and retrieving PublicKeys and PrivateKeys
- Add unit demos demonstrating how to use individual KMIP client operations
- Add custom configuration support to the KMIP client
- Add inline documentation for new KMIP objects, attributes and payloads
- Add additional unit test suites for new KMIP objects, attributes and payloads
- Add dependency for the six library to handle Python version support
- Update README with a condensed description and breakdown of the library
- Fix bug with unindexed format strings (impacts Python 2.6)

- Fix missing certificate file issue when installing library from PyPI

2.2.14 0.2 - November 17, 2014

- Add configuration file support
- Add client support for the Locate operation
- Update README with additional information and reStructuredText format

2.2.15 0.1.1 - September 12, 2014

- Fix bug with auto-installing third party dependencies

2.2.16 0.1.0 - August 28, 2014

- Add support for Python 3.3 and 3.4
- Add support for KMIP client/server SSL connections
- Remove all Thrift library dependencies

2.2.17 0.0.1 - August 12, 2014

- Initial release
- Add support for Python 2.6 and 2.7
- Add KMIP client and server
- Add client/server support for Create, Get, Register, and Destroy operations
- Add unit test suite

2.3 Frequently Asked Questions

Table of Contents

2.3.1 What algorithms are available for creating symmetric encryption keys? For asymmetric encryption keys (i.e., key pairs)?

The KMIP specification supports a wide variety of symmetric and asymmetric key algorithms. Support for these algorithms, including corresponding key lengths, will vary across different KMIP-compliant devices, so check with your KMIP vendor or with your appliance documentation to determine which ones are available.

For a full list of the cryptographic algorithms supported by the KMIP specification, see *cryptographic_algorithm*. The following algorithms are supported by the PyKMIP server.

Symmetric Key Algorithms

- 3DES
- AES
- Blowfish
- Camellia
- CAST5
- IDEA
- RC4

Asymmetric Key Algorithms

- RSA

2.3.2 How does the PyKMIP server handle client identity and authentication?

See *Authentication*.

2.3.3 How does the PyKMIP server manage access control for the keys and objects it stores?

See *Access Control*.

2.3.4 What built-in operation policies does the PyKMIP server support?

See *Reserved Operation Policies*.

2.4 Development

Development for PyKMIP is open to all contributors. Use the information provided here to inform your contributions and help the project maintainers review and accept your work.

2.4.1 Getting Started

File a new issue on the project issue tracker on GitHub describing the work you intend on doing. This is especially recommended for any sizable contributions, like adding support for a new KMIP operation or adding a new cryptographic backend for the server. Provide as much information on your feature request as possible, using information from the KMIP specifications or existing feature support in PyKMIP where applicable.

The issue number for your new issue should be included at the end of the commit message of each patch related to that issue.

If you simply want to request a new feature but do not intend on working on it, file your issue as normal and the project maintainers will triage it for future work.

2.4.2 Writing Code

New code should be written in its own Git branch, ideally branched from `HEAD` on `master`. If other commits are merged into `master` after your branch was created, be sure to rebase your work on the current state of `master` before submitting a pull request to GitHub.

New code should generally follow `PEP 8` style guidelines, though there are exceptions that will be allowed in special cases. Run the `flake8` tests to check your code before submitting a pull request (see *Running Tests*).

2.4.3 Writing Documentation

Like new code, new documentation should be written in its own Git branch. All PyKMIP documentation is written in RST format and managed using `sphinx`. It can be found under `docs/source`.

If you are interested in contributing to the project documentation, install the project documentation requirements:

```
$ pip install -r doc-requirements.txt
```

To build the documentation, navigate into the `docs` directory and run:

```
$ make html
```

This will build the PyKMIP documentation as HTML and place it under the new `docs/build/html` directory. View it using your preferred web browser.

2.4.4 Commit Messages

Commit messages should include a single line title (75 character max) followed by a blank line and a description of the change, including feature details, testing and documentation updates, feature limitations, known issues, etc.

The issue number for the issue associated with the commit should be included at the end of the commit message, if it exists. If the commit is the final one for a specific issue, use `Closes #XXX` or `Fixes #XXX` to link the issue and close it simultaneously. For example, see ths commit for Issue #312:

```
Fix bug generating detached instance errors in server tests

This patch fixes a bug that generates intermittent sqlalchemy
DetachedInstanceErrors during the KMIP server engine unit test
execution. Specifically, this fix disables instance expiration on
commit for the sqlalchemy sessions used throughout the unit tests,
allowing access to instance attributes even if the instance is
```

(continues on next page)

(continued from previous page)

```
detached from a session.

Fixes #312
```

2.4.5 Bug Fixes

If you have found a bug in PyKMIP, file a new issue and use the title format `Bug: <brief description here>`. In the body of the issue please provide as much information as you can, including Python version, PyKMIP version, operating system version, and any stacktraces or logging information produced by PyKMIP related to the bug. See What to put in your bug report for a breakdown of bug reporting best practices.

If you are working on a bug fix for a bug in `master`, follow the general guidelines above for branching and code development (see *Writing Code*).

If you are working on a bug fix for an older version of PyKMIP, your branch should be based on the latest commit of the repository branch for the version of PyKMIP the bug applies to (e.g., branch `release-0.6.0` for PyKMIP 0.6). The pull request for your bug fix should also target the version branch in question. If applicable, it will be pulled forward to newer versions of PyKMIP, up to and including `master`.

2.4.6 Running Tests

PyKMIP uses `tox` to manage testing across multiple Python versions. `tox` in turn uses `pytest` to run individual tests. Test infrastructure currently supports Python 2.7, 3.4, 3.5, 3.6, and 3.7. Additional test environments are provided for security, style, and documentation checks.

Note: All of the `tox` commands discussed in this section should be run from the root of the PyKMIP repository, in the same directory as the `tox.ini` configuration file.

The style checks leverage `flake8` and can be run like so:

```
$ tox -e pep8
```

The security checks use `bandit` and can be run like so:

```
$ tox -e bandit
```

The documentation checks leverage `sphinx` to build the HTML documentation in a temporary directory, verifying that there are no errors. These checks can be run like so:

```
$ tox -e docs
```

To run the above checks along with the entire unit test suite, simply run `tox` without any arguments:

```
$ tox
```

Unit Tests

The unit test suite tests many of the individual components of the PyKMIP code base, verifying that each component works correctly in isolation. Ideal code coverage would include the entire code base. To facilitate improving coverage, test coverage results are included with each Python unit test environment.

To test against a specific Python version (e.g., Python 2.7), run:

```
$ tox -e py27
```

To run an individual test suite method or class, use the `pytest -k` flag to specify the name of the method or class to execute. For example, to run the `TestProxyKmipClient` test suite class under Python 2.7, run:

```
$ tox -e py27 -- -k TestProxyKmipClient
```

For more information on the `-k` flag, see the pytest documentation.

Integration Tests

The integration test suite tests the functionality of the PyKMIP clients against a KMIP server, verifying that the right response data and status codes are returned for specific KMIP requests. A KMIP server must already be running and available over the network for the integration test cases to pass.

Code base coverage is not a goal of the integration test suite. Code coverage statistics are therefore not included in the output of the integration tests. For code coverage, run the unit tests above.

For the Travis CI tests run through GitHub, the KMIP server used for integration testing is actually an instance of the PyKMIP server, allowing us to verify the functionality of the clients and server simultaneously.

Any third-party KMIP server can be tested using the integration test suite. Simply add a section to the client configuration file containing the connection settings for the server and provide the name of the new section when invoking the integration tests.

To run the integration test suite, the configuration file section name for the client settings must be passed to the test suite using the `--config` configuration argument. Assuming the section name is `server_1`, the following `tox` command will set up and execute the integration tests:

```
$ tox -r -e integration -- --config server_1
```

Like the unit tests, use the `-k` flag to specify a specific test suite method or class.

```
$ tox -r -e integration -- --config server_1 -k TestProxyKmipClientIntegration
```

Functional Tests

The functional test suite tests capabilities and functionality specific to the PyKMIP server. While similar in structure to the integration test suite described above, the functional tests cannot be used with arbitrary third-party servers and require a very specific environment in which to operate successfully. Therefore, the functional tests are usually only used for continuous integration testing via Travis CI.

Like the integration test suite, code base coverage is not a goal of the functional test suite. For code coverage, run the unit tests above.

The functional tests specifically exercise third-party authentication and group-based access control features supported by the PyKMIP server. The third-party authentication system in this case is an instance of SLUGS. The PyKMIP client/server certificates and server operation policies must align exactly with the user/group information provided by SLUGS for the functional tests to pass. For more information, see the Travis CI build information under `.travis` in the PyKMIP repository.

To invoke the functional tests, the configuration file path must be passed to the test suite using the `--config-file` configuration argument. Assuming the file path is `/tmp/pykmip/client.conf`, the following `tox` command will set up and execute the functional tests:

```
$ tox -r -e functional -- --config-file /tmp/pykmip/client.conf
```

Like the unit and integration tests, use the `-k` flag to specify a specific test suite method or class.

```
$ tox -r -e functional -- --config-file /tmp/pykmip/client.conf -k test_policy_caching
```

For more information on the testing tools used here, see the following resources:

- tox
- flake8
- bandit

2.5 Security

The PyKMIP development team takes security seriously and will respond promptly to any reported security issue. Use the information provided here to inform your security posture.

2.5.1 Reporting a Security Issue

If you discover a new PyKMIP security issue, please follow responsible disclosure best practices and contact the project maintainers in private over email to discuss the issue before filing a public GitHub issue. When reporting a security issue, please include as much detail as possible. This includes:

- a high-level description of the issue
- information on how to cause or reproduce the issue
- any details on specific portions of the project code base related to the issue

Once you have provided this information, you should receive an acknowledgement. Depending upon the severity of the issue, the project maintainers will respond to collect additional information and work with you to address the security issue. If applicable, a new library subrelease will be produced across all actively supported releases to address and fix the issue.

If the developerment team decides the issue does not warrant the sensitivity of a security issue, you may file a public GitHub issue on the project issue tracker.

2.5.2 Known Vulnerabilities

The following are known vulnerabilities for older, unsupported versions of PyKMIP.

CVE	Brief Description	PyKMIP Version(s)	Mitigation
CVE-2018-1000872	Server Denial-of-Service	<=0.7.0	Upgrade to PyKMIP 0.8.0+

2.6 Client

The PyKMIP client allows developers to connect to a KMIP-compliant key management server and conduct key management operations.

2.6.1 Configuration

The client settings can be managed by a configuration file, by default located at `/etc/pykmip/pykmip.conf`. An example client configuration settings block, as found in the configuration file, is shown below:

```
[client]
host=127.0.0.1
port=5696
certfile=/path/to/certificate/file
keyfile=/path/to/certificate/key/file
ca_certs=/path/to/ca/certificate/file
cert_reqs=CERT_REQUIRED
ssl_version=PROTOCOL_SSLv23
do_handshake_on_connect=True
suppress_ragged_eofs=True
username=example_username
password=example_password
```

The configuration file can contain multiple settings blocks. Only one, `[client]`, is shown above. You can swap between different settings blocks by simply providing the name of the block as the `config` parameter (see below).

The different configuration options are defined below:

- **`host`** A string representing either a hostname in Internet domain notation or an IPv4 address.
- **`port`** An integer representing a port number. Recommended to be `5696` according to the KMIP specification.
- **`certfile`** A string representing a path to a PEM-encoded client certificate file. For more information, see the ssl documentation.
- **`keyfile`** A string representing a path to a PEM-encoded client certificate key file. The private key contained in the file must correspond to the certificate pointed to by `certfile`. For more information, see the ssl documentation.
- **`ca_certs`** A string representing a path to a PEM-encoded certificate authority certificate file. This certificate will be used to verify the server's certificate when establishing a TLS connection. For more information, see the ssl documentation.
- **`cert_reqs`** A flag indicating the enforcement level to use when validating the certificate received from the server. Options include: `CERT_NONE`, `CERT_OPTIONAL`, and `CERT_REQUIRED`. `CERT_REQUIRED` is the most secure option and should be used at all times. The other options can be helpful when debugging TLS connections. For more information, see the ssl documentation.
- **`ssl_version`** A flag indicating the SSL/TLS version to use when establishing a TLS connection with a server. Options are derived from the ssl module. The recommended value is `PROTOCOL_SSLv23` or `PROTOCOL_TLS`, which automatically allows the client to pick the most secure option provided by the server. For more information, see the ssl documentation.
- **`do_handshake_on_connect`** A boolean flag indicating when the client should perform the TLS handshake when establishing the TLS connection. The recommended value is `True`. For more information, see the ssl documentation.

  Note: This configuration option is deprecated and will be removed in a future version of PyKMIP.

- **`suppress_ragged_eofs`** A boolean flag indicating how the client should handle unexpected EOF from the TLS connection. The recommended value is `True`. For more information, see the ssl documentation.

Note: This configuration option is deprecated and will be removed in a future version of PyKMIP.

- **username** A string representing the username to use for KMIP requests. Optional depending on server access policies. Leave blank if not needed.
- **password** A string representing the password to use for KMIP requests. Optional depending on server access policies. Leave blank if not needed.

The client can also be configured manually via Python. The following example shows how to create the `ProxyKmipClient` in Python code, directly specifying the different configuration values:

```
>>> import ssl
>>> from kmip.pie.client import ProxyKmipClient, enums
>>> client = ProxyKmipClient(
...     hostname='127.0.0.1',
...     port=5696,
...     cert='/path/to/certificate/file',
...     key='/path/to/certificate/key/file',
...     ca='/path/to/ca/certificate/file',
...     ssl_version=ssl.PROTOCOL_SSLv23,
...     username='example_username',
...     password='example_password',
...     config='client',
...     config_file='/etc/pykmip/pykmip.conf',
...     kmip_version=enums.KMIPVersion.KMIP_1_2
... )
```

Settings specified at runtime, as in the above example, will take precedence over the default values found in the configuration file.

2.6.2 Usage

The following class documentation provides numerous examples detailing how to use the client. For additional examples, demo scripts for different operations are available in the `kmip/demos/pie` directory.

2.6.3 Class Documentation

class kmip.pie.client.**ProxyKmipClient**(*hostname=None*, *port=None*, *cert=None*, *key=None*, *ca=None*, *ssl_version=None*, *username=None*, *password=None*, *config='client'*, *config_file=None*, *kmip_version=None*)

A simplified KMIP client for conducting KMIP operations.

The ProxyKmipClient is a simpler KMIP client supporting various KMIP operations. It wraps the original KMIPProxy, reducing the boilerplate needed to deploy PyKMIP in client applications. The underlying proxy client is responsible for setting up the underlying socket connection and for writing/reading data to/from the socket.

Like the KMIPProxy, the ProxyKmipClient is not thread-safe.

Parameters

- **hostname** (*string*) – The host or IP address of a KMIP appliance. Optional, defaults to None.

- **port** (*int*) – The port number used to establish a connection to a KMIP appliance. Usually 5696 for KMIP applications. Optional, defaults to None.
- **cert** (*string*) – The path to the client's certificate. Optional, defaults to None.
- **key** (*string*) – The path to the key for the client's certificate. Optional, defaults to None.
- **ca** (*string*) – The path to the CA certificate used to verify the server's certificate. Optional, defaults to None.
- **ssl_version** (*string*) – The name of the ssl version to use for the connection. Example: 'PROTOCOL_SSLv23'. Optional, defaults to None.
- **username** (*string*) – The username of the KMIP appliance account to use for operations. Optional, defaults to None.
- **password** (*string*) – The password of the KMIP appliance account to use for operations. Optional, defaults to None.
- **config** (*string*) – The name of a section in the PyKMIP configuration file. Use to load a specific set of configuration settings from the configuration file, instead of specifying them manually. Optional, defaults to the default client section, 'client'.
- **config_file** (*string*) – The path to the PyKMIP client configuration file. Optional, defaults to None.
- **kmip_version** (*enum*) – A KMIPVersion enumeration specifying which KMIP version should be used to encode/decode request/response messages. Optional, defaults to None. If no value is specified, at request encoding time the client will default to KMIP 1.2.

kmip_version
The KMIP version that should be used to encode/decode request/response messages. Must be a KMIPVersion enumeration. Can be accessed and modified at any time.

open()
Open the client connection.

Raises

- **kmip.pie.exceptions.ClientConnectionFailure** – This is raised if the client connection is already open.
- **Exception** – This is raised if an error occurs while trying to open the connection.

close()
Close the client connection.

Raises **Exception** – This is raised if an error occurs while trying to close the connection.

activate(*uid=None*)
Activate a managed object stored by a KMIP appliance.

Parameters **uid** (*string*) – The unique ID of the managed object to activate. Optional, defaults to None.

Returns None

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.

- **TypeError** – This is raised if the input argument is invalid.

Activating a symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     key_id = c.register(symmetric_key)
...     c.activate(key_id)
```

check(*uid=None*, *usage_limits_count=None*, *cryptographic_usage_mask=None*, *lease_time=None*)

Check the constraints for a managed object.

Parameters

- **uid** (*string*) – The unique ID of the managed object to check.
- **usage_limits_count** (*int*) – The number of items that can be secured with the specified managed object.
- **cryptographic_usage_mask** (*list*) – A list of `kmip.core.enums.CryptographicUsageMask` enumerations specifying the operations allowed for the specified managed object.
- **least_time** (*int*) – The number of seconds that can be leased for the specified managed object.

Returns The string ID of the managed object that was checked.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input arguments are invalid.

```
>>> from kmip.pie import client
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.check(
...         uid="1",
...         usage_limits_count=50
...     )
'1'
```

create(*algorithm*, *length*, *operation_policy_name=None*, *name=None*, *cryptographic_usage_mask=None*)

Create a symmetric key on a KMIP appliance.

Parameters

- **algorithm** – A `kmip.core.enums.CryptographicAlgorithm` enumeration defining the algorithm to use to generate the symmetric key. See *cryptographic_algorithm* for more information.
- **length** (*int*) – The length in bits for the symmetric key.
- **operation_policy_name** (*string*) – The name of the operation policy to use for the new symmetric key. Optional, defaults to None
- **name** (*string*) – The name to give the key. Optional, defaults to None.
- **cryptographic_usage_mask** (*list*) – A list of `kmip.core.enums.CryptographicUsageMask` enumerations defining how the created key should be used. Optional, defaults to None. See *cryptographic_usage_mask* for more information.

Returns The string uid of the newly created symmetric key.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input arguments are invalid.

Creating an 256-bit AES key used for encryption and decryption would look like this:

```
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> with c:
...     key_id = c.create(
...         enums.CryptographicAlgorithm.AES,
...         256,
...         operation_policy_name='default',
...         name='Test_256_AES_Symmetric_Key',
...         cryptographic_usage_mask=[
...             enums.CryptographicUsageMask.ENCRYPT,
...             enums.CryptographicUsageMask.DECRYPT
...         ]
...     )
'449'
```

create_key_pair(*algorithm*, *length*, *operation_policy_name=None*, *public_name=None*, *public_usage_mask=None*, *private_name=None*, *private_usage_mask=None*)

Create an asymmetric key pair on a KMIP appliance.

Parameters

- **algorithm** – A `kmip.core.enums.CryptographicAlgorithm` enumeration defining the algorithm to use to generate the key pair. See *cryptographic_algorithm* for more information.
- **length** (*int*) – The length in bits for the key pair.
- **operation_policy_name** (*string*) – The name of the operation policy to use for the new key pair. Optional, defaults to None.
- **public_name** (*string*) – The name to give the public key. Optional, defaults to None.

- **public_usage_mask** (*list*) – A list of kmip.core.enums.CryptographicUsageMask enumerations indicating how the public key should be used. Optional, defaults to None. See *cryptographic_usage_mask* for more information.
- **private_name** (*string*) – The name to give the public key. Optional, defaults to None.
- **private_usage_mask** (*list*) – A list of kmip.core.enums.CryptographicUsageMask enumerations indicating how the private key should be used. Optional, defaults to None. See *cryptographic_usage_mask* for more information.

Returns The string uid of the newly created public key.

Returns The string uid of the newly created private key.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure
- **TypeError** – This is raised if the input arguments are invalid.

Creating an 2048-bit RSA key pair to be used for signing and signature verification would look like this:

```
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> with c:
...     key_id = c.create_key_pair(
...         enums.CryptographicAlgorithm.RSA,
...         2048,
...         operation_policy_name='default',
...         public_name='Test_2048_RSA_Public_Key',
...         public_usage_mask=[
...             enums.CryptographicUsageMask.VERIFY
...         ],
...         private_name='Test_2048_RSA_Private_Key',
...         private_usage_mask=[
...             enums.CryptographicUsageMask.SIGN
...         ]
...     )
('450', '451')
```

decrypt(*data*, *uid=None*, *cryptographic_parameters=None*, *iv_counter_nonce=None*)

Decrypt data using the specified decryption key and parameters.

Parameters

- **data** (*bytes*) – The bytes to decrypt. Required.
- **uid** (*string*) – The unique ID of the decryption key to use. Optional, defaults to None.
- **cryptographic_parameters** (*dict*) – A dictionary containing various cryptographic settings to be used for the decryption. Optional, defaults to None. See *cryptographic_parameters* for more information.
- **iv_counter_nonce** (*bytes*) – The bytes to use for the IV/counter/ nonce, if needed by the decryption algorithm and/or cipher mode. Optional, defaults to None.

Returns The decrypted data bytes.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Decrypting cipher text with a symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> with c:
...     key_id = c.create(
...         enums.CryptographicAlgorithm.AES,
...         256,
...         cryptographic_usage_mask=[
...             enums.CryptographicUsageMask.ENCRYPT,
...             enums.CryptographicUsageMask.DECRYPT
...         ]
...     )
...     c.activate(key_id)
...     c.decrypt(
...         (
...             b' \xb6:s0\x16\xea\t\x1b\x16\xed\xb2\x04-\xd6'
...             b'\xb6\\\xf3xJ\xfe\xa7[\x1eJ\x08I\xae\x14\xd2'
...             b\xdb\xe2'
...         ),
...         uid=key_id,
...         cryptographic_parameters={
...             'cryptographic_algorithm':
...                 enums.CryptographicAlgorithm.AES,
...             'block_cipher_mode': enums.BlockCipherMode.CBC,
...             'padding_method': enums.PaddingMethod.PKCS5
...         },
...         iv_counter_nonce=(
...             b'\x85\x1e\x87\x64\x77\x6e\x67\x96'
...             b'\xaa\xb7\x22\xdb\xb6\x44\xac\xe8'
...         )
...     )
...
b'This is a secret message.'
```

delete_attribute(*unique_identifier=None*, ***kwargs*)

Delete an attribute from a managed object.

Parameters

- **unique_identifier** (*string*) – The unique ID of the managed object from which to delete the specified attribute.
- ****kwargs** – A placeholder for attribute values used to identify the attribute to delete. See the examples below for more information.

Returns The string ID of the managed object from which the attribute was deleted.

Returns A `kmip.core.primitives.Struct` object representing the deleted attribute. Only returned for KMIP 1.0 - 1.4 messages.

For KMIP 1.0 - 1.4, the supported *kwargs* values are:

- *attribute_name* (string): The name of the attribute to delete. Required.
- *attribute_index* (int): The index of the attribute to delete. Defaults to zero.

```
>>> from kmip.pie import client
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.delete_attribute(
...         unique_identifier="1",
...         attribute_name="Name",
...         attribute_index=0
...     )
('1', Attribute(...))
```

For KMIP 2.0+, the supported *kwargs* values are:

- *current_attribute* (struct): A `kmip.core.objects.CurrentAttribute` object containing the attribute to delete. Required if the attribute reference is not specified.
- *attribute_reference* (struct): A `kmip.core.objects.AttributeReference` object containing the name of the attribute to delete. Required if the current attribute is not specified.

```
>>> from kmip.pie import client
>>> from kmip import enums
>>> from kmip.core import objects, primitives
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.delete_attribute(
...         unique_identifier="1",
...         current_attribute=objects.CurrentAttribute(
...             attribute=primitives.TextString(
...                 value="Object Group 1",
...                 tag=enums.Tags.OBJECT_GROUP
...             ),
...         ),
...         attribute_reference=objects.AttributeReference(
...             vendor_identification="Vendor 1",
...             attribute_name="Object Group"
...         )
...     )
'1'
```

derive_key(*object_type*, *unique_identifiers*, *derivation_method*, *derivation_parameters*, ***kwargs*)

Derive a new key or secret data from existing managed objects.

Parameters

- **object_type** – A `kmip.core.enums.ObjectType` enumeration specifying what type of object to derive. Only SymmetricKeys and SecretData can be specified. Required. See *object_type* for more information.
- **unique_identifiers** (*list*) – A list of strings specifying the unique IDs of the existing managed objects to use for derivation. Multiple objects can be specified to fit the requirements of the given derivation method. Required.

- **derivation_method** – A `kmip.core.enums.DerivationMethod` enumeration specifying how key derivation should be done. Required. See *derivation_method* for more information.
- **derivation_parameters** (*dict*) – A dictionary containing various settings for the key derivation process. Required. See *derivation_parameters* for more information.
- ****kwargs** – A placeholder for object attributes that should be set on the newly derived object. See the examples below for more information.

Returns The unique string ID of the newly derived object.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input arguments are invalid.

Deriving a new key using PBKDF2 would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> secret_data = objects.SecretData(
...     b'password',
...     enums.SecretDataType.PASSWORD,
...     masks=[
...         enums.CryptographicUsageMask.DERIVE_KEY
...     ]
... )
>>> with c:
...     password_id = c.register(secret_data)
...     c.activate(password_id)
...     c.derive_key(
...         enums.ObjectType.SYMMETRIC_KEY,
...         [password_id],
...         enums.DerivationMethod.PBKDF2,
...         {
...             'cryptographic_parameters': {
...                 'hashing_algorithm':
...                     enums.HashingAlgorithm.SHA_1
...             },
...             'salt': b'salt',
...             'iteration_count': 4096
...         },
...         cryptographic_length=128,
...         cryptographic_algorithm=enums.CryptographicAlgorithm.AES
...     )
...
'454'
```

Deriving a new secret using encryption would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
```

(continues on next page)

(continued from previous page)

```
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.BLOWFISH,
...     128,
...     (
...         b'\x01\x23\x45\x67\x89\xAB\xCD\xEF'
...         b'\xF0\xE1\xD2\xC3\xB4\xA5\x96\x87'
...     ),
...     masks=[
...         enums.CryptographicUsageMask.DERIVE_KEY
...     ]
... )
>>> with c:
...     key_id = c.register(key)
...     c.activate(key_id)
...     c.derive_key(
...         enums.ObjectType.SECRET_DATA,
...         [key_id],
...         enums.DerivationMethod.ENCRYPT,
...         {
...             'cryptographic_parameters': {
...                 'block_cipher_mode': enums.BlockCipherMode.CBC,
...                 'padding_method': enums.PaddingMethod.PKCS5,
...                 'cryptographic_algorithm':
...                     enums.CryptographicAlgorithm.BLOWFISH
...             },
...             'initialization_vector': (
...                 b'\xFE\xDC\xBA\x98\x76\x54\x32\x10'
...             ),
...             'derivation_data': (
...                 b'\x37\x36\x35\x34\x33\x32\x31\x20'
...                 b'\x4E\x6F\x77\x20\x69\x73\x20\x74'
...                 b'\x68\x65\x20\x74\x69\x6D\x65\x20'
...                 b'\x66\x6F\x72\x20\x00'
...             )
...         },
...         cryptographic_length=256
...     )
...
'456'
```

Deriving a new key using NIST 800 108-C would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     512,
...     (
...         b'\xdd\x5d\xbd\x45\x59\x3e\xe2\xac'
...         b'\x13\x97\x48\xe7\x64\x5b\x45\x0f'
...         b'\x22\x3d\x2f\xf2\x97\xb7\x3f\xd7'
...         b'\x1c\xbc\xeb\xe7\x1d\x41\x65\x3c'
...         b'\x95\x0b\x88\x50\x0d\xe5\x32\x2d'
```

(continues on next page)

(continued from previous page)

```
...         b'\x99\xef\x18\xdf\xdd\x30\x42\x82'
...         b'\x94\xc4\xb3\x09\x4f\x4c\x95\x43'
...         b'\x34\xe5\x93\xbd\x98\x2e\xc6\x14'
...     ),
...     masks=[
...         enums.CryptographicUsageMask.DERIVE_KEY
...     ]
... )
>>> with c:
...     key_id = c.register(key)
...     c.activate(key_id)
...     c.derive_key(
...         enums.ObjectType.SYMMETRIC_KEY,
...         [key_id],
...         enums.DerivationMethod.NIST800_108_C,
...         {
...             'cryptographic_parameters': {
...                 'hashing_algorithm':
...                     enums.HashingAlgorithm.SHA_512
...             },
...             'derivation_data': (
...                 b'\xb5\x0b\x0c\x96\x3c\x6b\x30\x34'
...                 b'\xb8\xcf\x19\xcd\x3f\x5c\x4e\xbe'
...                 b'\x4f\x49\x85\xaf\x0c\x03\xe5\x75'
...                 b'\xdb\x62\xe6\xfd\xf1\xec\xfe\x4f'
...                 b'\x28\xb9\x5d\x7c\xe1\x6d\xf8\x58'
...                 b'\x43\x24\x6e\x15\x57\xce\x95\xbb'
...                 b'\x26\xcc\x9a\x21\x97\x4b\xbd\x2e'
...                 b'\xb6\x9e\x83\x55'
...             )
...         },
...         cryptographic_length=128,
...         cryptographic_algorithm=enums.CryptographicAlgorithm.AES
...     )
...
'458'
```

Deriving a new secret using HMAC would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> secret = objects.SecretData(
...     (
...         b'\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c'
...         b'\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c'
...         b'\x0c\x0c\x0c\x0c\x0c\x0c\x0c\x0c'
...     ),
...     enums.SecretDataType.SEED,
...     masks=[
...         enums.CryptographicUsageMask.DERIVE_KEY
...     ]
... )
>>> with c:
...     secret_id = c.register(secret)
...     c.activate(secret_id)
```

(continues on next page)

(continued from previous page)

```
...     c.derive_key(
...         enums.ObjectType.SECRET_DATA,
...         [secret_id],
...         enums.DerivationMethod.HMAC,
...         {
...             'cryptographic_parameters': {
...                 'hashing_algorithm':
...                     enums.HashingAlgorithm.SHA_1
...             },
...             'derivation_data': b'',
...             'salt': b''
...         },
...         cryptographic_length=336
...     )
...
'460'
```

destroy(*uid=None*)

Destroy a managed object stored by a KMIP appliance.

Parameters **uid** (*string*) – The unique ID of the managed object to destroy.

Returns None

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Destroying a symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     key_id = c.register(symmetric_key)
...     c.destroy(key_id)
```

encrypt(*data*, *uid=None*, *cryptographic_parameters=None*, *iv_counter_nonce=None*)

Encrypt data using the specified encryption key and parameters.

Parameters

- **data** (*bytes*) – The bytes to encrypt. Required.
- **uid** (*string*) – The unique ID of the encryption key to use. Optional, defaults to None.

- **cryptographic_parameters** (*dict*) – A dictionary containing various cryptographic settings to be used for the encryption. Optional, defaults to None. See *cryptographic_parameters* for more information.
- **iv_counter_nonce** (*bytes*) – The bytes to use for the IV/counter/ nonce, if needed by the encryption algorithm and/or cipher mode. Optional, defaults to None.

Returns The encrypted data bytes.

Returns The IV/counter/nonce bytes used with the encryption algorithm, only if it was autogenerated by the server.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Encrypting plain text with a symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> with c:
...     key_id = c.create(
...         enums.CryptographicAlgorithm.AES,
...         256,
...         cryptographic_usage_mask=[
...             enums.CryptographicUsageMask.ENCRYPT,
...             enums.CryptographicUsageMask.DECRYPT
...         ]
...     )
...     c.activate(key_id)
...     c.encrypt(
...         b'This is a secret message.',
...         uid=key_id,
...         cryptographic_parameters={
...             'cryptographic_algorithm':
...                 enums.CryptographicAlgorithm.AES,
...             'block_cipher_mode': enums.BlockCipherMode.CBC,
...             'padding_method': enums.PaddingMethod.PKCS5
...         },
...         iv_counter_nonce=(
...             b'\x85\x1e\x87\x64\x77\x6e\x67\x96'
...             b'\xaa\xb7\x22\xdb\xb6\x44\xac\xe8'
...         )
...     )
...
(b'...', None)
```

get (*uid=None*, *key_wrapping_specification=None*)
Get a managed object from a KMIP appliance.

Parameters

- **uid** (*string*) – The unique ID of the managed object to retrieve.

- **key_wrapping_specification** (*dict*) – A dictionary containing the settings to use to wrap the object before retrieval. Optional, defaults to None. See *key_wrapping_specification* for more information.

Returns An `kmip.pie.objects.ManagedObject` instance.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Getting a symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     key_id = c.register(symmetric_key)
...     c.get(key_id)
SymmetricKey(...)
```

Getting a wrapped symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> wrapping_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x11\x22\x33\x44\x55\x66\x77'
...         b'\x88\x99\xAA\xBB\xCC\xDD\xEE\xFF'
...     ),
...     [
...         enums.CryptographicUsageMask.WRAP_KEY
...     ]
... )
```

(continues on next page)

(continued from previous page)

```
>>> with c:
...     key_id = c.register(symmetric_key)
...     wrapping_key_id = c.register(wrapping_key)
...     c.activate(wrapping_key_id)
...     c.get(
...         key_id,
...         key_wrapping_specification={
...             'wrapping_method': enums.WrappingMethod.ENCRYPT,
...             'encryption_key_information': {
...                 'unique_identifier': wrapping_key_id,
...                 'cryptographic_parameters': {
...                     'block_cipher_mode':
...                         enums.BlockCipherMode.NIST_KEY_WRAP
...                 }
...             },
...             'encoding_option': enums.EncodingOption.NO_ENCODING
...         }
...     )
SymmetricKey(...)
```

get_attributes(*uid=None*, *attribute_names=None*)

Get the attributes associated with a managed object.

If the uid is not specified, the appliance will use the ID placeholder by default.

If the attribute_names list is not specified, the appliance will return all viable attributes for the managed object.

Parameters

- **uid** (*string*) – The unique ID of the managed object with which the retrieved attributes should be associated. Optional, defaults to None.
- **attribute_names** (*list*) – A list of string attribute names indicating which attributes should be retrieved. Optional, defaults to None.

Returns The string ID of the object the attributes belong to.

Returns A list of `kmip.core.objects.Attribute` instances.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Retrieving all of the attributes for a managed object would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
```

(continues on next page)

(continued from previous page)

```
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     key_id = c.register(symmetric_key)
...     c.get_attributes(key_id)
('458', [Attribute(...), Attribute(...), ...])
```

Retrieving only a specific attribute for a managed object would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     key_id = c.register(symmetric_key)
...     c.get_attributes(key_id, ['Cryptographic Length'])
...
(
    '458',
    [
        Attribute(
            attribute_name=AttributeName(value='Cryptographic Length'),
            attribute_index=None,
            attribute_value=CryptographicLength(value=128)
        )
    ]
)
```

get_attribute_list(*uid=None*)

Get the names of the attributes associated with a managed object.

If the uid is not specified, the appliance will use the ID placeholder by default.

Parameters **uid** (*string*) – The unique ID of the managed object with which the retrieved attribute names should be associated. Optional, defaults to None.

Retrieving the list of attribute names for a symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
```

(continues on next page)

(continued from previous page)

```
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     key_id = c.register(symmetric_key)
...     c.get_attribute_list(key_id)
...
[
    'Cryptographic Algorithm',
    'Cryptographic Length',
    'Cryptographic Usage Mask',
    'Initial Date',
    'Object Type',
    'Operation Policy Name',
    'State',
    'Unique Identifier'
]
```

locate(*maximum_items=None*, *storage_status_mask=None*, *object_group_member=None*, *offset_items=None*, *attributes=None*)

Search for managed objects with specific matching attributes.

Parameters

- **maximum_items** (*int*) – The maximum number of results that should be returned.
- **storage_status_mask** (*int*) – A bit-mask indicating whether online or archived objects should be included in the search. See *storage_status* for more information.
- **object_group_member** (*enum*) – A `kmip.core.enums.ObjectGroupMember` enumeration indicating the object group member type. See *object_group_member* for more information.
- **offset_items** (*int*) – The number of results that should be skipped before results are returned.
- **attributes** (*list*) – A list of `kmip.core.objects.Attribute` objects representing an attribute filter for the search.

Returns A list of string IDs of all matching objects, per the operation parameters.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

```
>>> from kmip.pie import client
>>> from kmip.core import enums
>>> from kmip.core.factories import attributes
>>> f = attributes.AttributeFactory()
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.locate(
...         attributes=[
...             f.create_attribute(
```

(continues on next page)

(continued from previous page)

```
...                 enums.AttributeType.OBJECT_TYPE,
...                 enums.ObjectType.SYMMETRIC_KEY
...             )
...         ]
...     )
['1', '2', '3']
```

mac(*data*, *uid=None*, *algorithm=None*)
Get the message authentication code for a piece of data.

Parameters

- **data** (*string*) – The data to be MACed.
- **uid** (*string*) – The unique ID of the managed object that is the key to be used in the MAC operation.
- **algorithm** (*enum*) – A `kmip.core.enums.CryptographicAlgorithm` enumeration specifying the algorithm to use in the MAC operation. See *cryptographic_algorithm* for more information.

Returns The string ID of the managed object that is the key used in the MAC operation.

Returns The bytestring representing the MAC of the data.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

```
>>> from kmip.pie import client
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.mac(
...         b'\x01\x02\x03\x04',
...         uid="5",
...         algorithm=enums.CryptographicAlgorithm.HMAC_SHA512
...     )
('5', b'...')
```

modify_attribute(*unique_identifier=None*, ***kwargs*)
Modify an attribute on a managed object.

Parameters

- **unique_identifier** (*string*) – The unique ID of the managed object on which to set the specified attribute.
- ****kwargs** – A placeholder for attribute values used to identify the attribute to set. See the example below for more information.

Returns The string ID of the managed object on which the attribute was set.

Returns A `kmip.core.primitives.Struct` object representing the modified attribute. Only returned for KMIP 1.0 - 1.4 messages.

For KMIP 1.0 - 1.4, the supported *kwargs* values are:

- *attribute* (**struct**): **A `kmip.core.objects.Attribute` object** containing the details required to identify and modify an existing attribute on the specified managed object. Required.

```
>>> from kmip.pie import client
>>> from kmip.core import enums
>>> from kmip.core.factories import attributes
>>> f = attributes.AttributeFactory()
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.modify_attribute(
...         unique_identifier="1",
...         attribute=f.create_attribute(
...             enums.AttributeType.NAME,
...             "New Name",
...             index=0
...         )
...     )
('1', Attribute(...))
```

For KMIP 2.0+, the supported *kwargs* values are:

- *current_attribute* (struct): A `kmip.core.objects.CurrentAttribute` object containing the existing attribute to modify. Required if the attribute is multivalued.
- *new_attribute* (struct): A `kmip.core.objects.NewAttribute` object containing the new attribute. Required.

```
>>> from kmip.pie import client
>>> from kmip import enums
>>> from kmip.core import objects, primitives
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.modify_attribute(
...         unique_identifier="1",
...         current_attribute=objects.CurrentAttribute(
...             attribute=primitives.TextString(
...                 value="Old Object Group",
...                 tag=enums.Tags.OBJECT_GROUP
...             ),
...         ),
...         new_attribute=objects.NewAttribute(
...             attribute=primitives.TextString(
...                 value="New Object Group",
...                 tag=enums.Tags.OBJECT_GROUP
...             )
...         )
...     )
'1'
```

`register`(*managed_object*)

Register a managed object with a KMIP appliance.

Parameters **`managed_object`** – A `kmip.pie.objects.ManagedObject` instance to register with the server.

Returns The string uid of the newly registered managed object.

Raises

- **`kmip.pie.exceptions.ClientConnectionNotOpen`** – This is raised if the

client connection is unusable.

- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Registering an existing 128-bit AES symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     c.register(symmetric_key)
...
'452'
```

rekey(*uid=None*, *offset=None*, ***kwargs*)

Rekey an existing key.

Parameters

- **uid** (*string*) – The unique ID of the managed object that is the key to rekey.
- **offset** (*int*) – The time delta, in seconds, between the new key's initialization date and activation date.
- ****kwargs** – A placeholder for object attributes that should be set on the newly rekeyed key.

Returns The string ID of the newly rekeyed key.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

The current set of supported *kwargs* values are:

- *activation_date* (int): The new key's activation date, in seconds since the epoch.
- *process_start_date* (int): The new key's process start date, in seconds since the epoch.
- *protect_stop_date* (int): The new key's protect stop date, in seconds since the epoch.
- *deactivation_date* (int): The new key's deactivation date, in seconds since the epoch.

```
>>> from kmip.pie import client
>>> c = client.ProxyKmipClient()
>>> with c:
```

(continues on next page)

(continued from previous page)

```
...     c.rekey(
...         uid="1",
...         offset=60
...     )
"2"
```

revoke(*revocation_reason*, *uid=None*, *revocation_message=None*, *compromise_occurrence_date=None*)

Revoke a managed object stored by a KMIP appliance.

Activated objects must be revoked before they can be destroyed.

Parameters

- **revocation_reason** – A `kmip.core.enums.RevocationReasonCode` enumeration indicating the revocation reason. See *revocation_reason_code* for more information.
- **uid** (*string*) – The unique ID of the managed object to revoke. Optional, defaults to None.
- **revocation_message** (*string*) – A message regarding the revocation. Optional, defaults to None.
- **compromise_occurrence_date** (*int*) – An integer, the number of seconds since the epoch, which will be converted to the Datetime when the managed object was first believed to be compromised. Optional, defaults to None.

Returns None

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Revoking an activated symmetric key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> symmetric_key = objects.SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     )
... )
>>> with c:
...     key_id = c.register(symmetric_key)
...     c.activate(key_id)
...     c.revoke(
...         enums.RevocationReasonCode.CESSATION_OF_OPERATION,
...         key_id
...     )
```

set_attribute(*unique_identifier=None*, ***kwargs*)
Set an attribute on a managed object.

Parameters

- **unique_identifier** (*string*) – The unique ID of the managed object on which to set the specified attribute.
- ****kwargs** – A placeholder for attribute values used to identify the attribute to set. See the example below for more information.

Returns The string ID of the managed object on which the attribute was set.

This operation is supported by KMIP 2.0+ only. The supported *kwargs* values are:

- *attribute_name* (string): The name of the attribute to set. Required.
- *attribute_value* (various): The value of the attribute to set. Required.

```
>>> from kmip.pie import client
>>> c = client.ProxyKmipClient()
>>> with c:
...     c.set_attribute(
...         unique_identifier="1",
...         attribute_name="Sensitive",
...         attribute_value=True
...     )
'1'
```

sign(*data*, *uid=None*, *cryptographic_parameters=None*)
Create a digital signature for data using the specified signing key.

Parameters

- **data** (*bytes*) – The bytes of the data to be signed. Required.
- **uid** (*string*) – The unique ID of the signing key to use. Optional, defaults to None.
- **cryptographic_parameters** (*dict*) – A dictionary containing various cryptographic settings to be used for creating the signature (e.g., cryptographic algorithm, hashing algorithm, and/or digital signature algorithm). Optional, defaults to None. See *cryptographic_parameters* for more information.

Returns Bytes representing the signature of the data.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Signing data with a private key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
>>> with c:
...     public_key_id, private_key_id = c.create_key_pair(
```

(continues on next page)

(continued from previous page)

```
...         enums.CryptographicAlgorithm.RSA,
...         2048,
...         public_usage_mask=[
...             enums.CryptographicUsageMask.VERIFY
...         ],
...         private_usage_mask=[
...             enums.CryptographicUsageMask.SIGN
...         ]
...     )
...     c.activate(public_key_id)
...     c.activate(private_key_id)
...     signature = c.sign(
...         b'This is a signed message.',
...         uid=private_key_id,
...         cryptographic_parameters={
...             'padding_method': enums.PaddingMethod.PSS,
...             'cryptographic_algorithm':
...                 enums.CryptographicAlgorithm.RSA,
...             'hashing_algorithm': enums.HashingAlgorithm.SHA_256
...         }
...     )
...
>>> signature
b'...'
```

signature_verify(*message*, *signature*, *uid=None*, *cryptographic_parameters=None*)

Verify a message signature using the specified signing key.

Parameters

- **message** (*bytes*) – The bytes of the signed message. Required.
- **signature** (*bytes*) – The bytes of the message signature. Required.
- **uid** (*string*) – The unique ID of the signing key to use. Optional, defaults to None.
- **cryptographic_parameters** (*dict*) – A dictionary containing various cryptographic settings to be used for signature verification (e.g., cryptographic algorithm, hashing algorithm, and/or digital signature algorithm). Optional, defaults to None. See *cryptographic_parameters* for more information.

Returns A `kmip.core.enums.ValidityIndicator` enumeration indicating whether or not the signature was valid.

Raises

- **kmip.pie.exceptions.ClientConnectionNotOpen** – This is raised if the client connection is unusable.
- **kmip.pie.exceptions.KmipOperationFailure** – This is raised if the operation result is a failure.
- **TypeError** – This is raised if the input argument is invalid.

Verifying a signature with a public key would look like this:

```
>>> from kmip.pie import objects
>>> from kmip.pie import client
>>> from kmip import enums
>>> c = client.ProxyKmipClient()
```

(continues on next page)

(continued from previous page)

```
>>> with c:
...     public_key_id, private_key_id = c.create_key_pair(
...         enums.CryptographicAlgorithm.RSA,
...         2048,
...         public_usage_mask=[
...             enums.CryptographicUsageMask.VERIFY
...         ],
...         private_usage_mask=[
...             enums.CryptographicUsageMask.SIGN
...         ]
...     )
...     c.activate(public_key_id)
...     c.activate(private_key_id)
...     c.signature_verify(
...         b'This is a signed message.',
...         b'...',
...         uid=public_key_id,
...         cryptographic_parameters={
...             'padding_method': enums.PaddingMethod.PSS,
...             'cryptographic_algorithm':
...                 enums.CryptographicAlgorithm.RSA,
...             'hashing_algorithm': enums.HashingAlgorithm.SHA_256
...         }
...     )
...
<ValidityIndicator.VALID: 1>
```

2.7 Server

Warning: The PyKMIP server is intended for testing and demonstration purposes only. It is **not** a replacement for a secure, hardened, hardware-based key management appliance. It should **not** be used in a production-level environment, nor for critical operations.

The PyKMIP server is a software implementation of a KMIP-compliant key management appliance. It supports over a dozen key management operations, including key lifecycle management, object metadata access, and cryptographic functions like encrypting and signing data.

The server is used to test the functionality of the PyKMIP client and library and is primarily intended as a testing and demonstration tool.

2.7.1 Configuration

The server settings can be managed by a configuration file, by default located at `/etc/pykmip/server.conf`. An example server configuration settings block, as found in the configuration file, is shown below:

```
[server]
hostname=127.0.0.1
port=5696
certificate_path=/path/to/certificate/file
key_path=/path/to/certificate/key/file
```

(continues on next page)

(continued from previous page)

```
ca_path=/path/to/ca/certificate/file
auth_suite=Basic
policy_path=/path/to/policy/file
enable_tls_client_auth=True
tls_cipher_suites=
    TLS_RSA_WITH_AES_128_CBC_SHA256
    TLS_RSA_WITH_AES_256_CBC_SHA256
    TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
logging_level=DEBUG
database_path=/tmp/pykmip.db
```

The server can also be configured manually via Python. The following example shows how to create the `KmipServer` in Python code, directly specifying the different configuration values:

```
>>> from kmip.services.server import KmipServer
>>> server = KmipServer(
...     hostname='127.0.0.1',
...     port=5696,
...     certificate_path='/path/to/certificate/file/',
...     key_path='/path/to/certificate/key/file/',
...     ca_path='/path/to/ca/certificate/file/',
...     auth_suite='Basic',
...     config_path='/etc/pykmip/server.conf',
...     log_path='/var/log/pykmip/server.log',
...     policy_path='/etc/pykmip/policies',
...     enable_tls_client_auth=True,
...     tls_cipher_suites=[
...         'TLS_RSA_WITH_AES_128_CBC_SHA256',
...         'TLS_RSA_WITH_AES_256_CBC_SHA256',
...         'TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384'
...     ],
...     logging_level='DEBUG',
...     database_path='/tmp/pykmip.db'
... )
```

The different configuration options are defined below:

- **`hostname`** A string representing either a hostname in Internet domain notation or an IPv4 address.
- **`port`** An integer representing a port number. Recommended to be `5696` according to the KMIP specification.
- **`certificate_path`** A string representing a path to a PEM-encoded server certificate file. For more information, see the ssl documentation.
- **`key_path`** A string representing a path to a PEM-encoded server certificate key file. The private key contained in the file must correspond to the certificate pointed to by `certificate_path`. For more information, see the ssl documentation.
- **`ca_path`** A string representing a path to a PEM-encoded certificate authority certificate file. If using a self-signed certificate, the `ca_path` and the `certificate_path` should be identical. For more information, see the ssl documentation.
- **`auth_suite`** A string representing the type of authentication suite to use when establishing TLS connections. Acceptable values are `Basic` and `TLS1.2`.

  Note: `TLS1.2` can only be used with versions of Python that support TLS 1.2 (e.g,. Python 2.7.9+ or Python 3.4+). If you are running on an older version of Python, you will only be able to use basic TLS 1.0

authentication. For more information, see the ssl documentation.

- **config_path** A string representing a path to a server configuration file, as shown above. Only set via the `KmipServer` constructor. Defaults to `/etc/pykmip/server.conf`.
- **log_path** A string representing a path to a log file. The server will set up a rotating file logger on this file. Only set via the `KmipServer` constructor. Defaults to `/var/log/pykmip/server.log`.
- **policy_path** A string representing a path to the filesystem directory containing PyKMIP server operation policy JSON files.
- **enable_tls_client_auth** A boolean indicating whether or not extension checks should be performed on client certificates to verify that they can be used to derive client identity. This setting is enabled by default for backwards compatibility and must be explicitly disabled if this behavior is not desired.
- **tls_cipher_suites** A list of strings representing the set of cipher suites to use when establishing TLS connections with new clients. Enable debug logging for more information on the cipher suites used by the client and server.
- **logging_level** A string indicating what the base logging level should be for the server. Options include: DEBUG, INFO, WARNING, ERROR, and CRITICAL. The DEBUG log level logs the most information, the CRITICAL log level logs the least.
- **database_path** A string representing a path to a SQLite database file. The server will store all managed objects (e.g., keys, certificates) in this file.

Note: When installing PyKMIP and deploying the server, you must manually set up the server configuration file. It **will not** be placed in `/etc/pykmip` automatically. See `/examples` in the PyKMIP repository for a boilerplate configuration file to get started.

Third-Party Authentication

To configure third-party authentication plugins, separate configuration blocks must be specified in the server configuration file.

Note: Third-party authentication settings can only be set in the server configuration file. There is no way to set them using the `KmipServer` constructor in Python code.

An example authentication plugin configuration settings block is shown below:

```
[auth:slugs]
enabled=False
url=http://127.0.0.1:8080/slugs/
```

All authentication plugin configuration settings blocks must begin with the string `auth:`. For more information on third-party authentication integration, see *Third-Party Integration*.

2.7.2 Usage

The software server can be run using the `bin/run_server.py` startup script. If you are currently in the PyKMIP root directory, use the following command:

```
$ python bin/run_server.py
```

If you need more information about running the startup script, pass `-h` to it:

Note: You may need to run the server as root, depending on the permissions of the configuration, log, and certificate file directories.

If PyKMIP is installed and you are able to `import kmip` in Python, you can copy the startup script and run it from any directory you choose.

PyKMIP also defines a system-wide entry point that can be used to run the PyKMIP server once PyKMIP is installed. You can use the entry point like this:

```
$ pykmip-server
```

2.7.3 Storage

All data storage for the server is managed via SQLAlchemy. The current backend leverages SQLite, storing managed objects in a flat file. The file location can be configured using the `database_path` configuration setting. By default this file will be located at `/tmp/pykmip.database`. If this database file is deleted, the stored objects will be gone for good. If this file is preserved across server restarts, object access will be maintained.

Note: Updates to the server data model will generate errors if the server is run with a `pykmip.database` file adhering to an older data model. There is no upgrade path.

Long term, the intent is to add support for more robust database and storage backends available through `SQLAlchemy`. If you are interested in this work, please see *Development* for more information.

2.7.4 Authentication

Client authentication for the PyKMIP server is currently enforced by the validation of the client certificate used to establish the client/server TLS connection. If the client connects to the server with a certificate that has been signed by a certificate authority recognized by the server, the initial connection is allowed. If the server cannot validate the client's certificate, the connection is blocked and the client cannot access any objects stored on the server.

If client authentication succeeds, the identity of the client is obtained from the client's certificate. The server will extract the common name from the certificate's subject distinguished name and use the common name as the identity of the client. If the `enable_tls_client_auth` configuration setting is set to `True`, the server will check the client's certificate for the extended key usage extension (see RFC 5280). In this case the certificate must have the extension marked for client authentication, which indicates that the certificate can be used to derive client identity. If the extension is not present or is marked incorrectly, the server will not be able to derive the client's identity and will close the connection. If the `enable_tls_client_auth` configuration setting is set to `False`, the certificate extension check is omitted.

Once the client's identity is obtained, the client's request is processed. Any objects created or registered by the client will be marked as owned by the client identity. This identity is then used in conjunction with KMIP operation policies to enforce object access control (see *Access Control*).

Third-Party Integration

Beyond validating the client's certificate and extracting the client identity from the certificate's subject distinguished name, the server also supports a configurable framework for third-party authentication. This allows the server to integrate with existing authentication systems.

For each enabled third-party authentication plugin, the server will query the associated third-party service to verify that the user identified by the client certificate is a valid user. If validation succeeds, the server will also query the service for information pertaining to any groups the user may belong to. This information is leveraged for fine-grained access control (see *Access Control*). No other plugins are queried once a validation success has occurred. If validation fails, the server will attempt to authenticate with the next enabled plugin. If validation fails for all enabled plugins, the server will reject the client's request and close the connection. Validation only needs to succeed for one authentication plugin for client authentication to succeed.

If no third-party authentication plugins are enabled, the server will skip third-party authentication and will rely solely on client certificate validation for client authentication. Note that in this case, no user group information is available for fine-grained access control.

For more information on configuring third-party authentication plugins, see *Third-Party Authentication*.

Supported third-party authentication plugins are discussed below.

SLUGS

The Simple, Lightweight User Group Services (SLUGS) library is an open-source web service that serves user/group membership data over a basic REST interface. It is intended as an easy-to-use stopgap for developers and deployers interested in leveraging third-party authentication with the PyKMIP server.

All SLUGS plugin configuration settings blocks must begin with the string `auth:slugs`. Multiple SLUGS plugins can be configured at once; simply add a unique suffix to the block name to distinguish it from other blocks (e.g., `auth:slugs:primary`, `auth:slugs:secondary`).

The different configuration options supported by the SLUGS plugin are defined below:

- **`enabled`** A boolean indicating whether or not the authentication plugin should be used for authentication.
- **`url`** A string representing the URL at which to access a SLUGS REST interface.

For more information on SLUGS, see SLUGS.

2.7.5 Access Control

Access control for server objects is managed through KMIP operation policies. An operation policy is a set of permissions, indexed by object type and operation. For any KMIP object type and operation pair, the policy defines who is allowed to conduct the operation on the object type.

There are three basic permissions currently supported by KMIP:

- **`Allow All`** This permission indicates that any client authenticated with the server can conduct the corresponding operation on any object of the corresponding type.
- **`Allow Owner`** This permission restricts the operation to any client authenticated and identified as the owner of the object.
- **`Disallow All`** This permission blocks any client from conducting the operation on the object and is usually reserved for static public objects or tasks that only the server itself is allowed to perform.

For example, let's examine a simple use case where a client wants to retrieve a symmetric key from the server.

1. The client submits a `Get` request to the server (see *Get*), including the UUID of the symmetric key it wants to retrieve.
2. The server will derive the client's identity and then lookup the object with the corresponding UUID.
3. If the object is located, the server will check the object's operation policy attribute for the name of the operation policy associated with the object.
4. The server will then use the operation policy, the client's identity, the object's type, the object's owner, and the operation to determine if the client can retrieve the symmetric key.
5. If the operation policy has symmetric keys and the `Get` operation mapped to `Allow All`, the operation is allowed for the client regardless of the client's identity and the symmetric key is returned to the client. If the permission is set to `Allow Owner`, the server will return the symmetric key only if the client's identity matches the object's owner. If the permission is set to `Disallow All`, the server will refuse to return the symmetric key, regardless of the client's identity.

While an operation policy can cover every possible combination of object type and operation, it does not have to. If a policy does not cover a specific object type or operation, the server defaults to the safest option and acts as if the permission was set to `Disallow All`.

Each KMIP object is assigned an operation policy and owner upon creation. If no operation policy is included in the creation request, the server automatically assigns it the `default` operation policy. The `default` operation policy is defined in the KMIP specification and is built into the PyKMIP server; it cannot be redefined or overridden by the user or server administrator. For more information on reserved policies, see *Reserved Operation Policies*.

Policy Files

In addition to the built-in operation policies, the PyKMIP server allows users to define their own operation policies via policy files. A policy file is a basic JSON file that maps names for policies to tables of access controls. The server dynamically loads policy files from the policy directory, which is defined by the `policy_path` configuration setting. The server tracks any changes made to the policy directory, supporting the addition, modification, and/or removal of policy files and/or policies within those files. This allows users and administrators to modify and update their policies while the server is running, without any downtime. Note that it is up to the server administrator to ensure that user-defined policies do not overwrite each other by using identical policy names. Should this occur, the server will cache older policies, dynamically restoring them should the naming collision be corrected.

An example policy file, `policy.json`, is included in the `examples` directory of the PyKMIP repository. Let's take a look at the first few lines from the policy:

```
{
    "example": {
        "preset": {
            "CERTIFICATE": {
                "LOCATE": "ALLOW_ALL",
                "CHECK":  "ALLOW_ALL",
...
```

The first piece of information in the policy file is the name of the policy, in this case `example`. The name maps to collections of operation policies, grouped into two sets. The first set, shown here, is the `preset` collection. The `preset` collection contains rules that are used when user group information is unavailable; this is usually the case when third-party authentication is disabled. The `preset` collection rules consist of a set of object types, which in turn are mapped to a set of operations with associated permissions. In the snippet above, the first object type supported is `CERTIFICATE` followed by two supported operations, `LOCATE` and `CHECK`. Both operations are mapped to the `ALLOW_ALL` permission. Putting this all together, all clients are allowed to use the `LOCATE` and `CHECK` operations with certificate objects under the `example` policy, regardless of who owns the certificate being accessed. If you examine the full example file, you will see more operations listed, along with additional object types.

The second collection of operation policies that can be found in an operation policy file is the `groups` collection. This collection is used to provide group-based access control to objects. The following snippet is similar to the above snippet, reworked to use `groups` instead of `preset`:

```
{
    "example": {
        "groups": {
            "group_A": {
                "CERTIFICATE": {
                    "GET": "ALLOW_ALL",
                    "DESTROY": "ALLOW_ALL",
                    ...
            },
            "group_B": {
                "CERTIFICATE": {
                    "GET": "ALLOW_ALL",
                    "DESTROY": "DISALLOW_ALL",
                    ...
```

Like the prior snippet, the policy name is `example`. However, unlike the `preset` collection shown before, the `groups` collection first maps to a series of group names, in this case `group_A` and `group_B`. Each group maps to a set of object types and then access controls, following the same structure used by `preset`. The controls mapped under each group are distinct. This allows the policy to provide segregated access controls for groups of users, making it easy to share objects managed by the server while retaining fine-grained access control. In this case, any user belonging to `group_A` will be able to retrieve and destroy certificates using the `example` policy. Users in `group_B` will also be able to retrieve these certificates, but they will be unable to destroy them. Users belonging to both groups will receive the most permissive permissions available across the set of controls, meaning these users will be able to retrieve and destroy certificates since the controls under `group_A` are the most permissive.

The `preset` and `groups` collections can be included in the same policy. For example:

```
{
    "example": {
        "preset": {
            "CERTIFICATE": {
                "DESTROY": "DISALLOW_ALL",
                ...
        },
        "groups": {
            "group_A": {
                "CERTIFICATE": {
                    "DESTROY": "ALLOW_ALL",
                ...
            },
            "group_B": {
                "CERTIFICATE": {
                    "DESTROY": "DISALLOW_ALL",
                    ...
            }
        }
    }
}
```

As stated above, the controls belonging to the `groups` collection are only enforced if user group information is available after client authentication. If client authentication succeeds but no group information is available, the controls belonging to the `preset` collection are enforced. This allows users to effectively enable/disable group-level access controls if applicable to their use case. If group information is provided but only `preset` controls are defined, the `preset` controls will be enforced. If group information is not provided but only `groups` controls are defined,

`Disallow All` will be the only enforced control for the policy. This ensures that the policy behaves according to user expectations.

Finally, a single policy file can contain multiple policies:

```
{
    "example_1": {
        "preset": {
            "CERTIFICATE": {
                "DESTROY": "DISALLOW_ALL",
                ...
        }
    },
    "example_2": {
        "groups": {
            "group_A": {
                "CERTIFICATE": {
                    "DESTROY": "ALLOW_ALL",
                ...
            },
            "group_B": {
                "CERTIFICATE": {
                    "DESTROY": "DISALLOW_ALL",
                    ...
            }
        }
    }
}
```

The above snippet shows two policies, `example_1` and `example_2`. Each contains a different set of rules, one leveraging a `preset` collection and the other using the `groups` collection. While defined in the same JSON block, these policies are distinct from one another and are treated as separate entities. All of the previously defined rules and conventions for policies still apply.

Reserved Operation Policies

The PyKMIP server defines two reserved, built-in operation policies: `default` and `public`. Both of these policies are defined in the KMIP specification. Neither can be renamed or overridden by user-defined policies. The `default` policy is used for newly created objects that are not assigned a policy by their creators, though it can be used by creators intentionally. The `public` policy is intended for use with template objects that are public to the entire user-base of the server.

The following tables define the permissions for each of the built-in policies.

`default` policy

Object Type	Operation	Permission
Certificate	Locate	Allow All
Certificate	Check	Allow All
Certificate	Get	Allow All
Certificate	Get Attributes	Allow All
Certificate	Get Attribute List	Allow All
Certificate	Add Attribute	Allow Owner

Continued on next page

Table 1 – continued from previous page

Object Type	Operation	Permission
Certificate	Modify Attribute	Allow Owner
Certificate	Delete Attribute	Allow Owner
Certificate	Obtain Lease	Allow All
Certificate	Activate	Allow Owner
Certificate	Revoke	Allow Owner
Certificate	Destroy	Allow Owner
Certificate	Archive	Allow Owner
Certificate	Recover	Allow Owner
Symmetric Key	Rekey	Allow Owner
Symmetric Key	Rekey Key Pair	Allow Owner
Symmetric Key	Derive Key	Allow Owner
Symmetric Key	Locate	Allow Owner
Symmetric Key	Check	Allow Owner
Symmetric Key	Get	Allow Owner
Symmetric Key	Get Attributes	Allow Owner
Symmetric Key	Get Attribute List	Allow Owner
Symmetric Key	Add Attribute	Allow Owner
Symmetric Key	Modify Attribute	Allow Owner
Symmetric Key	Delete Attribute	Allow Owner
Symmetric Key	Obtain Lease	Allow Owner
Symmetric Key	Get Usage Allocation	Allow Owner
Symmetric Key	Activate	Allow Owner
Symmetric Key	Revoke	Allow Owner
Symmetric Key	Destroy	Allow Owner
Symmetric Key	Archive	Allow Owner
Symmetric Key	Recover	Allow Owner
Public Key	Locate	Allow All
Public Key	Check	Allow All
Public Key	Get	Allow All
Public Key	Get Attributes	Allow All
Public Key	Get Attribute List	Allow All
Public Key	Add Attribute	Allow Owner
Public Key	Modify Attribute	Allow Owner
Public Key	Delete Attribute	Allow Owner
Public Key	Obtain Lease	Allow All
Public Key	Activate	Allow Owner
Public Key	Revoke	Allow Owner
Public Key	Destroy	Allow Owner
Public Key	Archive	Allow Owner
Public Key	Recover	Allow Owner
Private Key	Rekey	Allow Owner
Private Key	Rekey Key Pair	Allow Owner
Private Key	Derive Key	Allow Owner
Private Key	Locate	Allow Owner
Private Key	Check	Allow Owner
Private Key	Get	Allow Owner
Private Key	Get Attributes	Allow Owner
Private Key	Get Attribute List	Allow Owner
Private Key	Add Attribute	Allow Owner

Continued on next page

Table 1 – continued from previous page

Object Type	Operation	Permission
Private Key	Modify Attribute	Allow Owner
Private Key	Delete Attribute	Allow Owner
Private Key	Obtain Lease	Allow Owner
Private Key	Get Usage Allocation	Allow Owner
Private Key	Activate	Allow Owner
Private Key	Revoke	Allow Owner
Private Key	Destroy	Allow Owner
Private Key	Archive	Allow Owner
Private Key	Recover	Allow Owner
Split Key	Rekey	Allow Owner
Split Key	Rekey Key Pair	Allow Owner
Split Key	Derive Key	Allow Owner
Split Key	Locate	Allow Owner
Split Key	Check	Allow Owner
Split Key	Get	Allow Owner
Split Key	Get Attributes	Allow Owner
Split Key	Get Attribute List	Allow Owner
Split Key	Add Attribute	Allow Owner
Split Key	Modify Attribute	Allow Owner
Split Key	Delete Attribute	Allow Owner
Split Key	Obtain Lease	Allow Owner
Split Key	Get Usage Allocation	Allow Owner
Split Key	Activate	Allow Owner
Split Key	Revoke	Allow Owner
Split Key	Destroy	Allow Owner
Split Key	Archive	Allow Owner
Split Key	Recover	Allow Owner
Template	Locate	Allow Owner
Template	Get	Allow Owner
Template	Get Attributes	Allow Owner
Template	Get Attribute List	Allow Owner
Template	Add Attribute	Allow Owner
Template	Modify Attribute	Allow Owner
Template	Delete Attribute	Allow Owner
Template	Destroy	Allow Owner
Secret Data	Rekey	Allow Owner
Secret Data	Rekey Key Pair	Allow Owner
Secret Data	Derive Key	Allow Owner
Secret Data	Locate	Allow Owner
Secret Data	Check	Allow Owner
Secret Data	Get	Allow Owner
Secret Data	Get Attributes	Allow Owner
Secret Data	Get Attribute List	Allow Owner
Secret Data	Add Attribute	Allow Owner
Secret Data	Modify	Allow Owner
Secret Data	Delete Attribute	Allow Owner
Secret Data	Obtain Lease	Allow Owner
Secret Data	Get Usage Allocation	Allow Owner
Secret Data	Activate	Allow Owner

Continued on next page

Table 1 – continued from previous page

Object Type	Operation	Permission
Secret Data	Revoke	Allow Owner
Secret Data	Destroy	Allow Owner
Secret Data	Archive	Allow Owner
Secret Data	Recover	Allow Owner
Opaque Data	Rekey	Allow Owner
Opaque Data	Rekey Key Pair	Allow Owner
Opaque Data	Derive Key	Allow Owner
Opaque Data	Locate	Allow Owner
Opaque Data	Check	Allow Owner
Opaque Data	Get	Allow Owner
Opaque Data	Get Attributes	Allow Owner
Opaque Data	Get Attribute List	Allow Owner
Opaque Data	Add Attribute	Allow Owner
Opaque Data	Modify Attribute	Allow Owner
Opaque Data	Delete Attribute	Allow Owner
Opaque Data	Obtain Lease	Allow Owner
Opaque Data	Get Usage Allocation	Allow Owner
Opaque Data	Activate	Allow Owner
Opaque Data	Revoke	Allow Owner
Opaque Data	Destroy	Allow Owner
Opaque Data	Archive	Allow Owner
Opaque Data	Recover	Allow Owner
PGP Key	Rekey	Allow Owner
PGP Key	Rekey Key Pair	Allow Owner
PGP Key	Derive Key	Allow Owner
PGP Key	Locate	Allow Owner
PGP Key	Check	Allow Owner
PGP Key	Get	Allow Owner
PGP Key	Get Attributes	Allow Owner
PGP Key	Get Attribute List	Allow Owner
PGP Key	Add Attribute	Allow Owner
PGP Key	Modify Attribute	Allow Owner
PGP Key	Delete Attribute	Allow Owner
PGP Key	Obtain Lease	Allow Owner
PGP Key	Get Usage Allocation	Allow Owner
PGP Key	Activate	Allow Owner
PGP Key	Revoke	Allow Owner
PGP Key	Destroy	Allow Owner
PGP Key	Archive	Allow Owner
PGP Key	Recover	Allow Owner

`public` policy

Object Type	Operation	Permission
Template	Locate	Allow All
Template	Get	Allow All
Template	Get Attributes	Allow All
Template	Get Attribute List	Allow All
Template	Add Attribute	Disallow All
Template	Modify Attribute	Disallow All
Template	Delete Attribute	Disallow All
Template	Destroy	Disallow All

2.7.6 Objects

The following is a list of KMIP managed object types supported by the server.

Symmetric Keys

A symmetric key is an encryption key that can be used to both encrypt plain text data and decrypt cipher text.

Creating a symmetric key object would look like this:

```
>>> from kmip import enums
>>> from kmip.pie.objects import SymmetricKey
>>> key = SymmetricKey(
...     enums.CryptographicAlgorithm.AES,
...     128,
...     (
...         b'\x00\x01\x02\x03\x04\x05\x06\x07'
...         b'\x08\x09\x0A\x0B\x0C\x0D\x0E\x0F'
...     ),
...     [
...         enums.CryptographicUsageMask.ENCRYPT,
...         enums.CryptographicUsageMask.DECRYPT
...     ],
...     "Example Symmetric Key"
... )
```

Public Keys

A public key is a cryptographic key that contains the public components of an asymmetric key pair. It is often used to decrypt data encrypted with, or to verify signatures produced by, the corresponding private key.

Creating a public key object would look like this:

```
>>> from kmip import enums
>>> from kmip.pie.objects import PublicKey
>>> key = PublicKey(
...     enums.CryptographicAlgorithm.RSA,
...     2048,
...     (
...         b'\x30\x82\x01\x0A\x02\x82\x01\x01...'
```

(continues on next page)

(continued from previous page)

```
...     ),
...     enums.KeyFormatType.X_509,
...     [
...         enums.CryptographicUsageMask.VERIFY
...     ],
...     "Example Public Key"
... )
```

Private Keys

A private key is a cryptographic key that contains the private components of an asymmetric key pair. It is often used to encrypt data that may be decrypted by, or generate signatures that may be verified by, the corresponding public key.

Creating a private key object would look like this:

```
>>> from kmip import enums
>>> from kmip.pie.objects import PrivateKey
>>> key = PrivateKey(
...     enums.CryptographicAlgorithm.RSA,
...     2048,
...     (
...         b'\x30\x82\x04\xA5\x02\x01\x00\x02...'
...     ),
...     enums.KeyFormatType.PKCS_8,
...     [
...         enums.CryptographicUsageMask.SIGN
...     ],
...     "Example Private Key"
... )
```

Split Keys

A split key is a secret value representing a key composed of multiple parts. The parts of the key can be recombined cryptographically to reconstitute the original key.

Creating a split key object would look like this:

```
>>> from kmip import enums
>>> from kmip.pie.objects import SplitKey
>>> key = SplitKey(
...     cryptographic_algorithm=enums.CryptographicAlgorithm.AES,
...     cryptographic_length=128,
...     key_value=b'\x00\x11\x22\x33\x44\x55\x66\x77\x88\x99\xAA\xBB\xCC\xDD\xEE\xFF',
...     name="Split Key",
...     split_key_parts=3,
...     key_part_identifier=1,
...     split_key_threshold=3,
...     split_key_method=enums.SplitKeyMethod.XOR
... )
```

Certificates

A certificate is a cryptographic object that contains a public key along with additional identifying information. It is often used to secure communication channels or to verify data signatures produced by the corresponding private key.

Creating a certificate object would look like this:

```
>>> from kmip import enums
>>> from kmip.pie.objects import X509Certificate
>>> cert = X509Certificate(
...     (
...         b'\x30\x82\x03\x12\x30\x82\x01\xFA...'
...     ),
...     [
...         enums.CryptographicUsageMask.VERIFY
...     ],
...     "Example X.509 Certificate"
... )
```

Secret Data

A secret data object is a cryptographic object that represents a shared secret value that is not a key or certificate (e.g., a password or passphrase).

Creating a secret data object would look like this:

```
>>> from kmip import enums
>>> from kmip.pie.objects import SecretData
>>> data = SecretData(
...     (
...         b'\x53\x65\x63\x72\x65\x74\x50\x61'
...         b'\x73\x73\x77\x6F\x72\x64'
...     ),
...     enums.SecretDataType.PASSWORD,
...     [
...         enums.CryptographicUsageMask.DERIVE_KEY
...     ],
...     "Example Secret Data Object"
... )
```

Opaque Objects

An opaque data object is a binary blob that the server is unable to interpret into another well-defined object type. It can be used to store any arbitrary data.

Creating an opaque object would look like this:

```
>>> from kmip import enums
>>> from kmip.pie.objects import OpaqueObject
>>> oo = OpaqueObject(
...     (
...         b'\x53\x65\x63\x72\x65\x74\x50\x61'
...         b'\x73\x73\x77\x6F\x72\x64'
...     ),
...     enums.OpaqueDataType.NONE,
...     "Example Opaque Object"
... )
```

2.7.7 Operations

The following is a list of KMIP operations supported by the server. All supported cryptographic functions are currently implemented using the pyca/cryptography library, which in turn leverages OpenSSL. If the underlying backend does not support a specific feature, algorithm, or operation, the PyKMIP server will not be able to support it either.

If you are interested in adding a new cryptographic backend to the PyKMIP server, see *Development* for more information.

Activate

The Activate operation updates the state of a managed object, allowing it to be used for cryptographic operations. Specifically, the object transitions from the pre-active state to the active state (see *state*).

Errors may be generated during the activation of a managed object. These may occur in the following cases:

- the managed object is not activatable (e.g., opaque data object)
- the managed object is not in the pre-active state

Create

The Create operation is used to create symmetric keys for a variety of cryptographic algorithms.

Algorithm	Key Sizes
3DES	64, 128, 192
AES	128, 256, 192
Blowfish	128, 256, 384, and more
Camellia	128, 256, 192
CAST5	64, 96, 128, and more
IDEA	128
ARC4	128, 256, 192, and more

All users are allowed to create symmetric keys. There are no quotas currently enforced by the server.

Various KMIP-defined attributes are set when a symmetric key is created. These include:

- *cryptographic_algorithm*
- *cryptographic_length*
- *cryptographic_usage_mask*
- *initial_date*
- *key_format_type*
- *name*
- *object_type*
- *operation_policy_name*
- *state*
- *unique_identifier*

Errors may be generated during the creation of a symmetric key. These may occur in the following cases:

- the cryptographic algorithm, length, and/or usage mask are not provided

- an unsupported symmetric algorithm is requested
- an invalid cryptographic length is provided for a specific cryptographic algorithm

CreateKeyPair

The CreateKeyPair operation is used to create asymmetric key pairs.

Algorithm	Key Sizes
RSA	512, 1024, 2048

All users are allowed to create asymmetric keys. There are no quotas currently enforced by the server.

Various KMIP-defined attributes are set when an asymmetric key pair is created. For both public and private keys, the following attributes are identical:

- *cryptographic_algorithm*
- *cryptographic_length*
- *initial_date*
- *operation_policy_name*
- *state*

Other attributes will differ between public and private keys. These include:

- *cryptographic_usage_mask*
- *key_format_type*
- *name*
- *object_type*
- *unique_identifier*

Errors may be generated during the creation of an asymmetric key pair. These may occur in the following cases:

- the cryptographic algorithm, length, and/or usage mask are not provided
- an unsupported asymmetric algorithm is requested
- an invalid cryptographic length is provided for a specific cryptographic algorithm

Decrypt

The Decrypt operations allows the client to decrypt data with an existing managed object stored by the server. Both symmetric and asymmetric decryption are supported. See *Encrypt* above for information on supported algorithms and the types of errors to expect from the server.

DeleteAttribute

The DeleteAttribute operation allows the client to delete an attribute from an existing managed object.

Errors may be generated during the attribute deletion process. These may occur in the following cases:

- the specified managed object does not exist
- the specified attribute may not be applicable to the specified managed object

- the specified attribute is not supported by the server
- the specified attribute cannot be deleted by the client
- the specified attribute could not be located for deletion on the specified managed object

DeriveKey

The DeriveKey operation is used to create a new symmetric key or secret data object from an existing managed object stored on the server. The derivation method and the desired length of the new cryptographic object must be specified with the request. If the generated cryptographic object is longer than the requested length, it will be truncated to match the request length.

Various KMIP-defined attributes are set when a new cryptographic object is derived. These include:

- *cryptographic_algorithm*
- *cryptographic_length*
- *cryptographic_usage_mask*
- *initial_date*
- *key_format_type*
- *name*
- *object_type*
- *operation_policy_name*
- *state*
- *unique_identifier*

Errors may be generated during the key derivation process. These may occur in the following cases:

- the base object is not accessible to the user
- the base object is not an object type usable for key derivation
- the base object does not have the DeriveKey bit set in its usage mask
- the cryptographic length is not provided with the request
- the requested cryptographic length is longer than the generated key

Destroy

The Destroy operation deletes a managed object from the server. Once destroyed, the object can no longer be retrieved or used for cryptographic operations. An object can only be destroyed if it is in the pre-active or deactivated states.

Errors may be generated during the destruction of a managed object. These may occur in the following cases:

- the managed object is not destroyable (e.g., the object does not exist)
- the managed object is in the active state

DiscoverVersions

The DiscoverVersions operation allows the client to determine which versions of the KMIP specification are supported by the server.

Encrypt

The Encrypt operation allows the client to encrypt data with an existing managed object stored by the server. Both symmetric and asymmetric encryption are supported:

Symmetric Key Algorithms

- 3DES
- AES
- Blowfish
- Camellia
- CAST5
- IDEA
- RC4

Asymmetric Key Algorithms

- RSA

Errors may be generated during the encryption. These may occur in the following cases:

- the encryption key is not accessible to the user
- the encryption key is not in the active state and must be activated
- the encryption key does not have the Encrypt bit set in its usage mask
- the requested encryption algorithm is not supported
- the specified encryption key is not compatible with the requested algorithm
- the requested encryption algorithm requires a block cipher mode
- the requested block cipher mode is not supported

Get

The Get attribute is used to retrieve a managed object stored on the server. The *unique_identifier* of the object is used to retrieve it.

It is possible to request that the managed object be cryptographically wrapped before it is returned to the client. Right now only encryption-based wrapping is supported.

Errors may be generated during the retrieval of a managed object. These may occur in the following cases:

- the managed object is not accessible to the user
- a desired key format was specified that cannot be converted by the server
- key compression was specified and the server cannot compress objects
- the wrapping key specified is not accessible to the user
- the wrapping key is not applicable to key wrapping
- the wrapping key does not have the WrapKey bit set in its usage mask

- wrapped attributes were specified and the server cannot wrap attributes
- a wrapping encoding was specified and the server does not support it
- incomplete wrapping specifications were provided with the request

GetAttributes

The GetAttributes operation is used to retrieve specific attributes for a specified managed object. Multiple attribute names can be specified in a single request.

The following names should be used to access the corresponding attributes:

Attribute Name	Attribute
Cryptographic Algorithm	*cryptographic_algorithm*
Cryptographic Length	*cryptographic_length*
Cryptographic Usage Mask	*cryptographic_usage_mask*
Initial Date	*initial_date*
Object Type	*object_type*
Operation Policy Name	*operation_policy_name*
State	*state*
Unique Identifier	*unique_identifier*

GetAttributeList

The GetAttributeList operation is used to identify the attributes currently available for a specific managed object. Given the *unique_identifier* of a managed object, the server will return a list of attribute names for attributes that can be accessed using the GetAttributes operation.

Locate

The Locate operation is used to identify managed objects that the user has access to, according to specific filtering criteria. Currently, the server only support object filtering based on the object *name* attribute.

If no filtering values are provided, the server will return a list of *unique_identifier* values corresponding to all of the managed objects the user has access to.

MAC

The MAC operation allows the client to compute a message authentication code on data using an existing managed object stored by the server. Both HMAC and CMAC algorithms are supported:

HMAC Hashing Algorithms

- MD5
- SHA1
- SHA224
- SHA256
- SHA384

- SHA512

CMAC Symmetric Algorithms

- 3DES
- AES
- Blowfish
- Camellia
- CAST5
- IDEA
- RC4

Errors may be generated during the authentication code creation process. These may occur in the following cases:

- the managed object to use is not accessible to the user
- the managed object to use is not in the active state and must be activated
- the managed object does not have the Generate bit set in its usage mask
- the requested algorithm is not supported for HMAC/CMAC generation

ModifyAttribute

The ModifyAttribute operation allows the client to modify an existing attribute on an existing managed object.

Errors may be generated during the attribute modification process. These may occur in the following cases:

- the specified managed object does not exist
- the specified attribute may not be applicable to the specified managed object
- the specified attribute is not supported by the server
- the specified attribute cannot be modified by the client
- the specified attribute is not set on the specified managed object
- the specified attribute is multivalued and the current attribute field must be specified
- the specified attribute index does not correspond to an existing attribute

Query

The Query operation allows the client to determine what KMIP capabilities are supported by the server. This set of information may include the following types of information, depending upon which items the client requests:

- *operation*
- *object_type*
- *vendor_identification*
- *server_information*
- *application_namespace*
- *extension_information*

- *attestation_type*
- *rng_parameters*
- *profile_information*
- *validation_information*
- *capability_information*
- *client_registration_method*

The PyKMIP server currently only includes the supported operations and the server information in Query responses.

Register

The Register operation is used to store an existing KMIP object with the server. For examples of the objects that can be stored, see *Objects*.

All users are allowed to register objects. There are no quotas currently enforced by the server.

Various KMIP-defined attributes may be set when an object is registered. These may include:

- *cryptographic_algorithm*
- *cryptographic_length*
- *cryptographic_usage_mask*
- *initial_date*
- *key_format_type*
- *name*
- *object_type*
- *operation_policy_name*
- *state*
- *unique_identifier*

Revoke

The Revoke operation updates the state of a managed object, effectively deactivating but not destroying it. The client provides a specific *revocation_reason_code* indicating why revocation is occurring.

If revocation is due to a key or CA compromise, the managed object is moved to the compromised state if it is in the pre-active, active, or deactivated states. If the object has already been destroyed, it will be moved to the destroyed compromised state. Otherwise, if revocation is due to any other reason, the managed object is moved to the deactivated state if it is in the active state.

Errors may be generated during the revocation of a managed object. These may occur in the following cases:

- the managed object is not revokable (e.g., opaque data object)
- the managed object is not active when revoked for a non-compromise

SetAttribute

The SetAttribute operation allows the client to set the value of an attribute on an existing managed object.

Errors may be generated during the attribute setting process. These may occur in the following cases:

- the specified managed object does not exist
- the specified attribute may not be applicable to the specified managed object
- the specified attribute is not supported by the server
- the specified attribute cannot be set by the client
- the specified attribute is multivalued and cannot be set with this operation

Sign

The Sign operation allows the client to sign data with an existing private key stored by the server. The following hashing algorithms are supported with RSA for signing support.

Hashing Algorithms

- MD5
- SHA1
- SHA224
- SHA256
- SHA384
- SHA512

Errors may be generated during the encryption. These may occur in the following cases:

- the signing key is not accessible to the user
- the signing key is not a private key
- the signing key is not in the active state and must be activated
- the signing key does not have the Sign bit set in its usage mask
- the requested signing algorithm is not supported
- the signing key is not compatible with the requested signing algorithm
- a padding method is required for the algorithm and was not specified

SignatureVerify

The SignatureVerify operation allows the client to verify a data signature with an existing public key stored by the server. See *Sign* above for information on supported algorithms and the types of errors to expect from the server.

2.8 Community

The PyKMIP community has various forums and resources you can use:

- Source code
- Issue tracker

2.9 Glossary

alternative_name_type (enum) (1.2) An enumeration specifying the type associated with an alternate name value. Used often as part of the alternative name attribute.

```
>>> from kmip import enums
>>> enums.AlternativeNameType.URI
<AlternativeNameType.URI: 2>
```

Name	Value	KMIP Version
UNINTERPRETED_TEXT_STRING	0x00000001	1.2
URI	0x00000002	1.2
OBJECT_SERIAL_NUMBER	0x00000003	1.2
EMAIL_ADDRESS	0x00000004	1.2
DNS_NAME	0x00000005	1.2
X500_DISTINGUISHED_NAME	0x00000006	1.2
IP_ADDRESS	0x00000007	1.2

application_namespace (str) (1.0) A string identifying a specific application namespace supported by the key management server. Often returned as part of the Query operation.

attestation_type (enum) (1.2) An enumeration specifying the type of attestation measurement included in an attestation credential. Used during client identification credential processing.

```
>>> from kmip import enums
>>> enums.AttestationType.TPM_QUOTE
<AttestationType.TPM_QUOTE: 1>
```

Name	Value	KMIP Version
TPM_QUOTE	0x00000001	1.2
TCG_INTEGRITY_REPORT	0x00000002	1.2
SAML_ASSERTION	0x00000003	1.2

batch_error_continuation_option (enum) (1.0) An enumeration used to control operation batch handling.

```
>>> from kmip import enums
>>> enums.BatchErrorContinuationOption.STOP
<BatchErrorContinuationOption.STOP: 2>
```

Name	Value	KMIP Version
CONTINUE	0x00000001	1.0
STOP	0x00000002	1.0
UNDO	0x00000003	1.0

block_cipher_mode (enum) (1.0) An enumeration specifying the block cipher mode to use with a cryptographic algorithm. Used often with sets of cryptographic parameters.

```
>>> from kmip import enums
>>> enums.BlockCipherMode.CTR
<BlockCipherMode.CTR: 6>
```

Name	Value	KMIP Version
CBC	0x00000001	1.0
ECB	0x00000002	1.0
PCBC	0x00000003	1.0
CFB	0x00000004	1.0
OFB	0x00000005	1.0
CTR	0x00000006	1.0
CMAC	0x00000007	1.0
CCM	0x00000008	1.0
GCM	0x00000009	1.0
CBC_MAC	0x0000000A	1.0
XTS	0x0000000B	1.0
AES_KEY_WRAP_PADDING	0x0000000C	1.0
NIST_KEY_WRAP	0x0000000D	1.0
X9_102_AESKW	0x0000000E	1.0
X9_102_TDKW	0x0000000F	1.0
X9_102_AKW1	0x00000010	1.0
X9_102_AKW2	0x00000011	1.0
AEAD	0x00000012	1.4

cancellation_result (enum) (1.0) An enumeration specifying the result of a cancelled operation.

```
>>> from kmip import enums
>>> enums.CancellationResult.FAILED
<CancellationResult.FAILED: 4>
```

Name	Value	KMIP Version
CANCELED	0x00000001	1.0
UNABLE_TO_CANCEL	0x00000002	1.0
COMPLETED	0x00000003	1.0
FAILED	0x00000004	1.0
UNAVAILABLE	0x00000005	1.0

capability_information (dict) (1.3) A dictionary containing information about a set of KMIP server capabilities. Often obtained from the Query operation response.

```
>>> from kmip import enums
>>> capability_information = {
...     'streaming_capability': False,
...     'asynchronous_capability': False,
...     'attestation_capability': False,
...     'unwrap_mode': enums.UnwrapMode.PROCESSED,
...     'destroy_action': enums.DestroyAction.DELETED,
...     'shredding_algorithm': enums.ShreddingAlgorithm.UNSUPPORTED,
...     'rng_mode': enums.RNGMode.SHARED_INSTANTIATION,
```

(continues on next page)

(continued from previous page)

```
...     'batch_undo_capability': False,
...     'batch_continue_capability': False
...     'quantum_safe_capability': False
... }
```

Key	Value	KMIP Version
streaming_capability	bool	1.3
asynchronous_capability	bool	1.3
attestation_capability	bool	1.3
unwrap_mode	enum	1.3
destroy_action	enum	1.3
shredding_algorithm	enum	1.3
rng_mode	enum	1.3
batch_undo_capability	bool	1.4
batch_continue_capability	bool	1.4
quantum_safe_capability	bool	2.0

certificate_request_type (enum) (1.0) An enumeration specifying the type of the certificate request sent with a certify operation request.

```
>>> from kmip import enums
>>> enums.CertificateRequestType.PEM
<CertificateRequestType.PEM: 3>
```

Name	Value	KMIP Version
CRMF	0x00000001	1.0
PKCS10	0x00000002	1.0
PEM	0x00000003	1.0
PGP	0x00000004	1.0

certificate_type (enum) (1.0) An enumeration specifying the type of a certificate object.

```
>>> from kmip import enums
>>> enums.CertificateTypeEnum.X_509
<CertificateTypeEnum.X_509: 1>
```

Name	Value	KMIP Version
X_509	0x00000001	1.0
PGP	0x00000002	1.0

client_registration_method (enum) (1.3) An enumeration specifying a type of registration method utilized by the client or server. Used often as part of the response to a Query request.

```
>>> from kmip import enums
>>> enums.ClientRegistrationMethod.CLIENT_REGISTERED
<ClientRegistrationMethod.CLIENT_REGISTERED: 5>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
SERVER_PREGENERATED	0x00000002	1.3
SERVER_ON_DEMAND	0x00000003	1.3
CLIENT_GENERATED	0x00000004	1.3
CLIENT_REGISTERED	0x00000005	1.3

credential_type (enum) (1.0) An enumeration specifying the type of a credential object. Used often as part of a credential structure.

```
>>> from kmip import enums
>>> enums.CredentialType.USERNAME_AND_PASSWORD
<CredentialType.USERNAME_AND_PASSWORD: 1>
```

Name	Value	KMIP Version
USERNAME_AND_PASSWORD	0x00000001	1.0
DEVICE	0x00000002	1.1
ATTESTATION	0x00000003	1.2
ONE_TIME_PASSWORD	0x00000004	2.0
HASHED_PASSWORD	0x00000005	2.0
TICKET	0x00000006	2.0

cryptographic_algorithm (enum) (1.0) An enumeration specifying the cryptographic algorithm to use for a cryptographic operation. Used often with sets of cryptographic parameters.

```
>>> from kmip import enums
>>> enums.CryptographicAlgorithm.RSA
<CryptographicAlgorithm.RSA: 4>
```

Name	Value	KMIP Version
DES	0x00000001	1.0
TRIPLE_DES	0x00000002	1.0
AES	0x00000003	1.0
RSA	0x00000004	1.0
DSA	0x00000005	1.0
ECDSA	0x00000006	1.0
HMAC_SHA1	0x00000007	1.0
HMAC_SHA224	0x00000008	1.0
HMAC_SHA256	0x00000009	1.0
HMAC_SHA384	0x0000000A	1.0
HMAC_SHA512	0x0000000B	1.0
HMAC_MD5	0x0000000C	1.0
DH	0x0000000D	1.0
ECDH	0x0000000E	1.0
ECMQV	0x0000000F	1.0
BLOWFISH	0x00000010	1.0
CAMELLIA	0x00000011	1.0
CAST5	0x00000012	1.0
IDEA	0x00000013	1.0
MARS	0x00000014	1.0

Continued on next page

Table 2 – continued from previous page

Name	Value	KMIP Version
RC2	0x00000015	1.0
RC4	0x00000016	1.0
RC5	0x00000017	1.0
SKIPJACK	0x00000018	1.0
TWOFISH	0x00000019	1.0
EC	0x0000001A	1.2
ONE_TIME_PAD	0x0000001B	1.3
CHACHA20	0x0000001C	1.4
POLY1305	0x0000001D	1.4
CHACHA20_POLY1305	0x0000001E	1.4
SHA3_224	0x0000001F	1.4
SHA3_256	0x00000020	1.4
SHA3_384	0x00000021	1.4
SHA3_512	0x00000022	1.4
HMAC_SHA3_224	0x00000023	1.4
HMAC_SHA3_256	0x00000024	1.4
HMAC_SHA3_384	0x00000025	1.4
HMAC_SHA3_512	0x00000026	1.4
SHAKE_128	0x00000027	1.4
SHAKE_256	0x00000028	1.4
ARIA	0x00000029	2.0
SEED	0x0000002A	2.0
SM2	0x0000002B	2.0
SM3	0x0000002C	2.0
SM4	0x0000002D	2.0
GOST_R_34_10_2012	0x0000002E	2.0
GOST_R_34_11_2012	0x0000002F	2.0
GOST_R_34_13_2015	0x00000030	2.0
GOST_28147_89	0x00000031	2.0
XMSS	0x00000032	2.0
SPHINCS_256	0x00000033	2.0
MCELIECE	0x00000034	2.0
MCELIECE_6960119	0x00000035	2.0
MCELIECE_8192128	0x00000036	2.0
ED25519	0x00000037	2.0
ED448	0x00000038	2.0

cryptographic_length (int) (1.0) A integer specifying the length of a cryptographic object in bits. Used as a parameter for creating encryption keys of various types and as an object attribute for cryptographic objects.

cryptographic_parameters (dict) (1.0) A dictionary containing key/value pairs representing settings to be used when performing cryptographic operations. Used as a parameter to various KMIP operations but can also be set as an attribute on a KMIP object.

```
>>> from kmip import enums
>>> cryptographic_parameters = {
...     'block_cipher_mode': enums.BlockCipherMode.CTR,
...     'padding_method': enums.PaddingMethod.PKCS5,
...     'random_iv': False,
...     'initial_counter_value': 0
... }
```

Key	Value	KMIP Version
block_cipher_mode	enum	1.0
padding_method	enum	1.0
hashing_algorithm	enum	1.0
key_role_type	enum	1.0
digital_signature_algorithm	enum	1.2
cryptographic_algorithm	enum	1.2
random_iv	bool	1.2
iv_length	int	1.2
tag_length	int	1.2
fixed_field_length	int	1.2
invocation_field_length	int	1.2
counter_length	int	1.2
initial_counter_value	int	1.2
salt_length	int	1.4
mask_generator	enum	1.4
mask_generator_hashing_algorithm	enum	1.4
p_source	bytes	1.4
trailer_field	int	1.4

cryptographic_usage_mask (enum) (1.0) An enumeration specifying a cryptographic capability, usually associated with a managed object. Often used in list form (e.g., [CryptographicUsageMask.SIGN, CryptographicUsageMask.VERIFY]).

```
>>> from kmip import enums
>>> enums.CryptographicUsageMask.ENCRYPT
<CryptographicUsageMask.ENCRYPT: 4>
```

Name	Value	KMIP Version
SIGN	0x00000001	1.0
VERIFY	0x00000002	1.0
ENCRYPT	0x00000004	1.0
DECRYPT	0x00000008	1.0
WRAP_KEY	0x00000010	1.0
UNWRAP_KEY	0x00000020	1.0
EXPORT	0x00000040	1.0
MAC_GENERATE	0x00000080	1.0
MAC_VERIFY	0x00000100	1.0
DERIVE_KEY	0x00000200	1.0
CONTENT_COMMITMENT	0x00000400	1.0
KEY_AGREEMENT	0x00000800	1.0
CERTIFICATE_SIGN	0x00001000	1.0
CRL_SIGN	0x00002000	1.0
GENERATE_CRYPTOGRAM	0x00004000	1.0
VALIDATE_CRYPTOGRAM	0x00008000	1.0
TRANSLATE_ENCRYPT	0x00010000	1.0
TRANSLATE_DECRYPT	0x00020000	1.0
TRANSLATE_WRAP	0x00040000	1.0
TRANSLATE_UNWRAP	0x00080000	1.0
AUTHENTICATE	0x00100000	2.0
UNRESTRICTED	0x00200000	2.0
FPE_ENCRYPT	0x00400000	2.0
FPE_DECRYPT	0x00800000	2.0

derivation_method (enum) (1.0) An enumeration specifying a key derivation method to be used to derive a new key. Used as a parameter to the DeriveKey operation.

```
>>> from kmip import enums
>>> enums.DerivationMethod.PBKDF2
<DerivationMethod.PBKDF2: 1>
```

Name	Value	KMIP Version
PBKDF2	0x00000001	1.0
HASH	0x00000002	1.0
HMAC	0x00000003	1.0
ENCRYPT	0x00000004	1.0
NIST800_108_C	0x00000005	1.0
NIST800_108_F	0x00000006	1.0
NIST800_108_DPI	0x00000007	1.0
ASYMMETRIC_KEY	0x00000008	1.4
AWS_SIGNATURE_VERSION_4	0x00000009	2.0
HKDF	0x0000000A	2.0

derivation_parameters (dict) (1.0) A dictionary containing key/value pairs representing settings to be used when performing key derivation operations. Used as a parameter to the DeriveKey operation.

```
>>> from kmip import enums
>>> derivation_parameters = {
...     'cryptographic_parameters': {...},
```

(continues on next page)

(continued from previous page)

```
...     'initialization_vector': b'\x01\x02\x03\x04',
...     'derivation_data': b'\xFF\xFF\xFF\xFF',
...     'salt': b'\x00\x00\xFF\xFF',
...     'iteration_count': 1000
... }
```

Key	Value	KMIP Version
cryptographic_parameters	dict	1.0
initialization_vector	bytes	1.0
derivation_data	bytes	1.0
salt	bytes	1.0
iteration_count	int	1.0

destroy_action (enum) (1.3) An enumeration specifying methods of data disposal used by a KMIP server. Used often as part of the response to a Query request.

```
>>> from kmip import enums
>>> enums.DestroyAction.SHREDDED
<DestroyAction.SHREDDED: 7>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
KEY_MATERIAL_DELETED	0x00000002	1.3
KEY_MATERIAL_SHREDDED	0x00000003	1.3
METADATA_DELETED	0x00000004	1.3
METADATA_SHREDDED	0x00000005	1.3
DELETED	0x00000006	1.3
SHREDDED	0x00000007	1.3

digital_signature_algorithm (enum) (1.1) An enumeration specifying a digital signature algorithm, usually associated with a signed object. Used often with sets of cryptographic parameters.

```
>>> from kmip import enums
>>> enums.DigitalSignatureAlgorithm.SHA256_WITH_RSA_ENCRYPTION
<DigitalSignatureAlgorithm.SHA256_WITH_RSA_ENCRYPTION: 5>
```

Name	Value	KMIP Version
MD2_WITH_RSA_ENCRYPTION	0x00000001	1.1
MD5_WITH_RSA_ENCRYPTION	0x00000002	1.1
SHA1_WITH_RSA_ENCRYPTION	0x00000003	1.1
SHA224_WITH_RSA_ENCRYPTION	0x00000004	1.1
SHA256_WITH_RSA_ENCRYPTION	0x00000005	1.1
SHA384_WITH_RSA_ENCRYPTION	0x00000006	1.1
SHA512_WITH_RSA_ENCRYPTION	0x00000007	1.1
RSASSA_PSS	0x00000008	1.1
DSA_WITH_SHA1	0x00000009	1.1
DSA_WITH_SHA224	0x0000000A	1.1
DSA_WITH_SHA256	0x0000000B	1.1
ECDSA_WITH_SHA1	0x0000000C	1.1
ECDSA_WITH_SHA224	0x0000000D	1.1
ECDSA_WITH_SHA256	0x0000000E	1.1
ECDSA_WITH_SHA384	0x0000000F	1.1
ECDSA_WITH_SHA512	0x00000010	1.1
SHA3_256_WITH_RSA_ENCRYPTION	0x00000011	1.4
SHA3_384_WITH_RSA_ENCRYPTION	0x00000012	1.4
SHA3_512_WITH_RSA_ENCRYPTION	0x00000013	1.4

drbg_algorithm (enum) (1.3) An enumeration specifying a deterministic random bit generator. Used often to describe a random number generator.

```
>>> from kmip import enums
>>> enums.DRBGAlgorithm.DUAL_EC
<DRBGAlgorithm.DUAL_EC: 2>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
DUAL_EC	0x00000002	1.3
HASH	0x00000003	1.3
HMAC	0x00000004	1.3
CTR	0x00000005	1.3

encoding_option (enum) (1.1) An enumeration specifying the encoding of an object before it is cryptographically wrapped. Used in various key wrapping metadata structures.

```
>>> from kmip import enums
>>> enums.EncodingOption.NO_ENCODING
<EncodingOption.NO_ENCODING: 1>
```

Name	Value	KMIP Version
NO_ENCODING	0x00000001	1.1
TTLV_ENCODING	0x00000002	1.1

encryption_key_information (dict) (1.0) A dictionary containing information on the encryption key used for key wrapping.

```
>>> from kmip import enums
>>> encryption_key_information = {
```

(continues on next page)

(continued from previous page)

```
...     'unique_identifier': '123e4567-e89b-12d3-a456-426655440000',
...     'cryptographic_parameters': {...}
... }
```

Key	Value	KMIP Version
unique_identifier	string	1.0
cryptographic_parameters	dict	1.0

extension_information (dict) (1.1) A dictionary containing information on a specific KMIP specification extension supported by a KMIP server. Often returned as part of a Query operation response.

```
>>> from kmip import enums
>>> extension_information = {
...     'extension_name': 'ACME LOCATION',
...     'extension_tag': 0x0054aa01,
...     'extension_type': 0x00000007
... }
>>> extension_information = {
...     'extension_name': 'ACME LOCATION',
...     'extension_tag': 0x0054aa01,
...     'extension_type': enums.ItemType.TEXT_STRING,
...     'extension_attribute': True,
...     'extension_parent_structure_tag': 0x0054aa02,
...     'extension_description': 'Example description.'
... }
```

Key	Value	KMIP Version
extension_name	string	1.1
extension_tag	int	1.1
extension_type	int / enum	1.1 / 2.0
extension_enumeration	int	2.0
extension_attribute	bool	2.0
extension_parent_structure_tag	int	2.0
extension_description	string	2.0

fips186_variation (enum) (1.3) An enumeration specifying a FIPS 186 variation. Used often to describe a random number generator.

```
>>> from kmip import enums
>>> enums.FIPS186Variation.K_CHANGE_NOTICE
<FIPS186Variation.K_CHANGE_NOTICE: 7>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
GP_X_ORIGINAL	0x00000002	1.3
GP_X_CHANGE_NOTICE	0x00000003	1.3
X_ORIGINAL	0x00000004	1.3
X_CHANGE_NOTICE	0x00000005	1.3
K_ORIGINAL	0x00000006	1.3
K_CHANGE_NOTICE	0x00000007	1.3

hashing_algorithm (enum) (1.0) An enumeration specifying the hashing method to use with a cryptographic algorithm. Used often with sets of cryptographic parameters.

```
>>> from kmip import enums
>>> enums.HashingAlgorithm.MD5
<HashingAlgorithm.MD5: 3>
```

Name	Value	KMIP Version
MD2	0x00000001	1.0
MD4	0x00000002	1.0
MD5	0x00000003	1.0
SHA_1	0x00000004	1.0
SHA_224	0x00000005	1.0
SHA_256	0x00000006	1.0
SHA_384	0x00000007	1.0
SHA_512	0x00000008	1.0
RIPEMD_160	0x00000009	1.0
TIGER	0x0000000A	1.0
WHIRLPOOL	0x0000000B	1.0
SHA_512_224	0x0000000C	1.2
SHA_512_256	0x0000000D	1.2
SHA3_224	0x0000000E	1.4
SHA3_256	0x0000000F	1.4
SHA3_384	0x00000010	1.4
SHA3_512	0x00000011	1.4

initial_date (int) (1.0) An integer specifying, in seconds since the Epoch, the date and time when a managed object first entered the pre-active state. This occurs when the object is first created or registered with the key management appliance. This value is set by the server on every managed object and cannot be changed.

item_type (enum) (2.0) An enumeration specifying the type of an object. Only the least significant byte of the enumeration value is used in KMIP object encodings.

```
>>> from kmip import enums
>>> enums.ItemType.STRUCTURE
<ItemType.STRUCTURE: 1>
```

Name	Value	KMIP Version
STRUCTURE	0x00000001	2.0
INTEGER	0x00000002	2.0
LONG_INTEGER	0x00000003	2.0
BIG_INTEGER	0x00000004	2.0
ENUMERATION	0x00000005	2.0
BOOLEAN	0x00000006	2.0
TEXT_STRING	0x00000007	2.0
BYTE_STRING	0x00000008	2.0
DATE_TIME	0x00000009	2.0
INTERVAL	0x0000000A	2.0
DATE_TIME_EXTENDED	0x0000000B	2.0

key_compression_type (enum) (1.0) An enumeration specifying the key compression used for an elliptic curve public key. Used as a key value attribute and as a parameter for the Get operation.

```
>>> from kmip import enums
>>> enums.KeyCompressionType.EC_PUBLIC_KEY_TYPE_UNCOMPRESSED
<KeyCompressionType.EC_PUBLIC_KEY_TYPE_UNCOMPRESSED: 1>
```

Name	Value	KMIP Version
EC_PUBLIC_KEY_TYPE_UNCOMPRESSED	0x00000001	1.0
EC_PUBLIC_KEY_TYPE_X9_62_COMPRESSED_PRIME	0x00000002	1.0
EC_PUBLIC_KEY_TYPE_X9_62_COMPRESSED_CHAR2	0x00000003	1.0
EC_PUBLIC_KEY_TYPE_X9_62_HYBRID	0x00000004	1.0

key_format_type (enum) (1.0) An enumeration specifying the format of key material. Used in various ways as a key value attribute, as well as a parameter to the Get operation.

```
>>> from kmip import enums
>>> enums.KeyFormatType.RAW
<KeyFormatType.RAW: 1>
```

Name	Value	KMIP Version
RAW	0x00000001	1.0
OPAQUE	0x00000002	1.0
PKCS_1	0x00000003	1.0
PKCS_8	0x00000004	1.0
X_509	0x00000005	1.0
EC_PRIVATE_KEY	0x00000006	1.0
TRANSPARENT_SYMMETRIC_KEY	0x00000007	1.0
TRANSPARENT_DSA_PRIVATE_KEY	0x00000008	1.0
TRANSPARENT_DSA_PUBLIC_KEY	0x00000009	1.0
TRANSPARENT_RSA_PRIVATE_KEY	0x0000000a	1.0
TRANSPARENT_RSA_PUBLIC_KEY	0x0000000b	1.0
TRANSPARENT_DH_PRIVATE_KEY	0x0000000c	1.0
TRANSPARENT_DH_PUBLIC_KEY	0x0000000d	1.0
TRANSPARENT_ECDSA_PRIVATE_KEY	0x0000000e	1.0
TRANSPARENT_ECDSA_PUBLIC_KEY	0x0000000f	1.0
TRANSPARENT_ECDH_PRIVATE_KEY	0x00000010	1.0
TRANSPARENT_ECDH_PUBLIC_KEY	0x00000011	1.0
TRANSPARENT_ECMQV_PRIVATE_KEY	0x00000012	1.0
TRANSPARENT_ECMQV_PUBLIC_KEY	0x00000013	1.0
TRANSPARENT_EC_PRIVATE_KEY	0x00000014	1.3
TRANSPARENT_EC_PUBLIC_KEY	0x00000015	1.3
PKCS_12	0x00000016	1.4

key_role_type (enum) (1.0) An enumeration specifying the key role type of the associated cryptographic key. Used often with sets of cryptographic parameters.

```
>>> from kmip import enums
>>> enums.KeyRoleType.KEK
<KeyRoleType.KEK: 11>
```

Name	Value	KMIP Version
BDK	0x00000001	1.0
CVK	0x00000002	1.0
DEK	0x00000003	1.0
MKAC	0x00000004	1.0
MKSMC	0x00000005	1.0
MKSMI	0x00000006	1.0
MKDAC	0x00000007	1.0
MKDN	0x00000008	1.0
MKCP	0x00000009	1.0
MKOTH	0x0000000A	1.0
KEK	0x0000000B	1.0
MAC_16609	0x0000000C	1.0
MAC_97971	0x0000000D	1.0
MAC_97972	0x0000000E	1.0
MAC_97973	0x0000000F	1.0
MAC_97974	0x00000010	1.0
MAC_97975	0x00000011	1.0
ZPK	0x00000012	1.0
PVKIBM	0x00000013	1.0
PVKPVV	0x00000014	1.0
PVKOTH	0x00000015	1.0
DUKPT	0x00000016	1.4
IV	0x00000017	1.4
TRKBK	0x00000018	1.4

key_value_location_type (enum) (1.2) An enumeration specifying the type of key value location identifier. Used in cases where a key value is stored outside a key server.

```
>>> from kmip import enums
>>> enums.KeyValueLocationType.URI
<KeyValueLocationType.URI: 2>
```

Name	Value	KMIP Version
UNINTERPRETED_TEXT_STRING	0x00000001	1.2
URI	0x00000002	1.2

key_wrap_type (enum) (1.4) An enumeration specifying the type of key wrap used to access a managed object. Used to specify key wrapping in Get and Export operations.

```
>>> from kmip import enums
>>> enums.KeyWrapType.NOT_WRAPPED
<KeyWrapType.NOT_WRAPPED: 1>
```

Name	Value	KMIP Version
NOT_WRAPPED	0x00000001	1.4
AS_REGISTERED	0x00000002	1.4

key_wrapping_data (dict) (1.0) A dictionary containing information on a cryptographic key wrapping mechanism used to wrap a key value.

```
>>> from kmip import enums
>>> key_wrapping_data = {
...     'wrapping_method': enums.WrappingMethod.ENCRYPT,
...     'encryption_key_information': {...},
...     'iv_counter_nonce': b'\x01\x02\x03\x04',
...     'encoding_option': enums.EncodingOption.NO_ENCODING
... }
```

Key	Value	KMIP Version
wrapping_method	enum	1.0
encryption_key_information	dict	1.0
mac_signature_key_information	dict	1.0
mac_signature	bytes	1.0
iv_counter_nonce	bytes	1.0
encoding_option	enum	1.1

key_wrapping_specification (dict) (1.0) A dictionary containing settings defining how an object should be cryptographically wrapped. Used as a parameter for the Get operation to retrieve cryptographically wrapped objects.

```
>>> from kmip import enums
>>> key_wrapping_specification = {
...     'wrapping_method': enums.WrappingMethod.ENCRYPT,
...     'encryption_key_information': {...},
...     'attribute_names': [
...         'Cryptographic Algorithm',
...         'Cryptographic Length'
...     ]
... }
```

Key	Value	KMIP Version
wrapping_method	enum	1.0
encryption_key_information	dict	1.0
mac_signature_key_information	dict	1.0
attribute_names	list	1.0
encoding_option	enum	1.1

kmip_version (enum) (-) An enumeration specifying the KMIP version to use for the client and/or server. Defined independently of any individual KMIP specification version.

```
>>> from kmip import enums
>>> enums.KMIPVersion.KMIP_1_1
<KMIPVersion.KMIP_1_1: 1.1>
```

Name	Value
KMIP_1_0	1.0
KMIP_1_1	1.1
KMIP_1_2	1.2
KMIP_1_3	1.3
KMIP_1_4	1.4
KMIP_2_0	2.0

link_type (enum) (1.0) An enumeration specifying the type of link connecting two managed objects. Used often as

an object attribute.

```
>>> from kmip import enums
>>> enums.LinkType.PUBLIC_KEY_LINK
<LinkType.PUBLIC_KEY_LINK: 258>
```

Name	Value	KMIP Version
CERTIFICATE_LINK	0x00000101	1.0
PUBLIC_KEY_LINK	0x00000102	1.0
PRIVATE_KEY_LINK	0x00000103	1.0
DERIVATION_BASE_OBJECT_LINK	0x00000104	1.0
DERIVED_KEY_LINK	0x00000105	1.0
REPLACEMENT_OBJECT_LINK	0x00000106	1.0
REPLACED_OBJECT_LINK	0x00000107	1.0
PARENT_LINK	0x00000108	1.2
CHILD_LINK	0x00000109	1.2
PREVIOUS_LINK	0x0000010a	1.2
NEXT_LINK	0x0000010b	1.2
PKCS12_CERTIFICATE_LINK	0x0000010c	1.4
PKCS12_PASSWORD_LINK	0x0000010d	1.4
WRAPPING_KEY_LINK	0x0000010E	2.0

mac_signature_key_information (dict) (1.0) A dictionary containing information on the MAC/signature key used for key wrapping.

```
>>> from kmip import enums
>>> mac_signature_key_information = {
...     'unique_identifier': '123e4567-e89b-12d3-a456-426655440000',
...     'cryptographic_parameters': {...}
... }
```

Key	Value	KMIP Version
unique_identifier	string	1.0
cryptographic_parameters	dict	1.0

mask_generator (enum) (1.4) An enumeration specifying the mask generation function to use for a cryptographic operation. Used often with sets of cryptographic parameters.

```
>>> from kmip import enums
>>> enums.MaskGenerator.MGF1
<MaskGenerator.MGF1: 1>
```

Name	Value	KMIP Version
MGF1	0x00000001	1.4

mask_generator_hashing_algorithm (enum) (1.4) Another name for a hash algorithm. See hashing_algorithm.

name (str) (1.0) A string specifying the name of a managed object stored by the server. It can be used in addition to the *unique_identifier* to identify an object and can be used as a filter with the Locate operation.

name_type (enum) (1.0) An enumeration specifying the type of name value used in a name attribute structure.

```
>>> from kmip import enums
>>> enums.NameType.URI
<NameType.URI: 2>
```

Name	Value	KMIP Version
UNINTERPRETED_TEXT_STRING	0x00000001	1.0
URI	0x00000002	1.0

object_group_member (enum) (1.1) An enumeration specifying whether or not a group object has been returned to a client before the current request. Used as a filtering flag for the Locate operation.

```
>>> from kmip import enums
>>> enums.ObjectGroupMember.GROUP_MEMBER_FRESH
<ObjectGroupMember.GROUP_MEMBER_FRESH: 1>
```

Name	Value	KMIP Version
GROUP_MEMBER_FRESH	0x00000001	1.1
GROUP_MEMBER_DEFAULT	0x00000002	1.1

object_type (enum) (1.0) An enumeration specifying the type of a managed object. Used as an attribute for every managed object on a key server.

```
>>> from kmip import enums
>>> enums.ObjectType.SYMMETRIC_KEY
<ObjectType.SYMMETRIC_KEY: 2>
```

Name	Value	KMIP Version
CERTIFICATE	0x00000001	1.0
SYMMETRIC_KEY	0x00000002	1.0
PUBLIC_KEY	0x00000003	1.0
PRIVATE_KEY	0x00000004	1.0
SPLIT_KEY	0x00000005	1.0
TEMPLATE	0x00000006	1.0
SECRET_DATA	0x00000007	1.0
OPAQUE_DATA	0x00000008	1.0
PGP_KEY	0x00000009	1.2
CERTIFICATE_REQUEST	0x0000000A	2.0

opaque_data_type (enum) (1.0) An enumeration specifying the type of the associated opaque data object. Note that no values have ever been specified by the KMIP specification. A custom NONE value is included in PyKMIP as a default. This value will only be recognized by the PyKMIP server.

```
>>> from kmip import enums
>>> enums.OpaqueDataType.NONE
<OpaqueDataType.NONE: 2147483648>
```

Name	Value	KMIP Version
NONE	0x80000000	–

operation (enum) (1.0) An enumeration specifying a KMIP operation. Used in KMIP requests.

```
>>> from kmip import enums
>>> enums.Operation.GET
<Operation.GET: 10>
```

Name	Value	KMIP Version
CREATE	0x00000001	1.0
CREATE_KEY_PAIR	0x00000002	1.0
REGISTER	0x00000003	1.0
REKEY	0x00000004	1.0
DERIVE_KEY	0x00000005	1.0
CERTIFY	0x00000006	1.0
RECERTIFY	0x00000007	1.0
LOCATE	0x00000008	1.0
CHECK	0x00000009	1.0
GET	0x0000000a	1.0
GET_ATTRIBUTES	0x0000000b	1.0
GET_ATTRIBUTE_LIST	0x0000000c	1.0
ADD_ATTRIBUTE	0x0000000d	1.0
MODIFY_ATTRIBUTE	0x0000000e	1.0
DELETE_ATTRIBUTE	0x0000000f	1.0
OBTAIN_LEASE	0x00000010	1.0
GET_USAGE_ALLOCATION	0x00000011	1.0
ACTIVATE	0x00000012	1.0
REVOKE	0x00000013	1.0
DESTROY	0x00000014	1.0
ARCHIVE	0x00000015	1.0
RECOVER	0x00000016	1.0
VALIDATE	0x00000017	1.0
QUERY	0x00000018	1.0
CANCEL	0x00000019	1.0
POLL	0x0000001a	1.0
NOTIFY	0x0000001b	1.0
PUT	0x0000001c	1.0
REKEY_KEY_PAIR	0x0000001d	1.1
DISCOVER_VERSIONS	0x0000001e	1.1
ENCRYPT	0x0000001f	1.2
DECRYPT	0x00000020	1.2
SIGN	0x00000021	1.2
SIGNATURE_VERIFY	0x00000022	1.2
MAC	0x00000023	1.2
MAC_VERIFY	0x00000024	1.2
RNG_RETRIEVE	0x00000025	1.2
RNG_SEED	0x00000026	1.2
HASH	0x00000027	1.2
CREATE_SPLIT_KEY	0x00000028	1.2
JOIN_SPLIT_KEY	0x00000029	1.2
IMPORT	0x0000002a	1.4
EXPORT	0x0000002b	1.4
LOG	0x0000002C	2.0
LOGIN	0x0000002D	2.0

Continued on next page

Table 3 – continued from previous page

Name	Value	KMIP Version
LOGOUT	0x0000002E	2.0
DELEGATED_LOGIN	0x0000002F	2.0
ADJUST_ATTRIBUTE	0x00000030	2.0
SET_ATTRIBUTE	0x00000031	2.0
SET_ENDPOINT_ROLE	0x00000032	2.0
PKCS_11	0x00000033	2.0
INTEROP	0x00000034	2.0
REPROVISION	0x00000035	2.0

operation_policy_name (str) (1.0) A string specifying the name of the operation policy that should be used for access control decisions for a managed object. One operation policy name attribute can be set per managed object by the server. Once set it cannot be changed by the client.

padding_method (enum) (1.0) An enumeration specifying the padding method to use to pad data during cryptographic operations. Used often with sets of cryptographic parameters.

```
>>> from kmip import enums
>>> enums.PaddingMethod.PKCS5
<PaddingMethod.PKCS5: 3>
```

Name	Value	KMIP Version
NONE	0x00000001	1.0
OAEP	0x00000002	1.0
PKCS5	0x00000003	1.0
SSL3	0x00000004	1.0
ZEROS	0x00000005	1.0
ANSI_X923	0x00000006	1.0
ISO_10126	0x00000007	1.0
PKCS1v15	0x00000008	1.0
X931	0x00000009	1.0
PSS	0x0000000A	1.0

profile_information (dict) (1.3) A dictionary containing information about a KMIP profile supported by a KMIP server. Often obtained from the Query operation response.

```
>>> from kmip import enums
>>> profile_information = {
...     'profile_name': enums.ProfileName.BASELINE_SERVER_BASIC_KMIPv12,
...     'server_uri': 'https://127.0.0.1',
...     'server_port': 5696,
...     'profile_version': {
...         'profile_version_major': 1,
...         'profile_version_minor': 0
...     }
... }
```

Key	Value	KMIP Version
profile_name	enum	1.3
server_uri	string	1.3
server_port	int	1.3
profile_version	dict	2.0

profile_name (enum) (1.3) An enumeration specifying a profile supported by the client or server. Used often as part of the response to a Query request.

```
>>> from kmip import enums
>>> enums.ProfileName.BASELINE_SERVER_BASIC_KMIPv12
<ProfileName.BASELINE_SERVER_BASIC_KMIPv12: 1>
```

Name	Value	KMIP Version
BASELINE_SERVER_BASIC_KMIPv12	0x00000001	1.3
BASELINE_SERVER_TLSv12_KMIPv12	0x00000002	1.3
BASELINE_CLIENT_BASIC_KMIPv12	0x00000003	1.3
BASELINE_CLIENT_TLSv12_KMIPv12	0x00000004	1.3
COMPLETE_SERVER_BASIC_KMIPv12	0x00000005	1.3
COMPLETE_SERVER_TLSv12_KMIPv12	0x00000006	1.3
TAPE_LIBRARY_CLIENT_KMIPv10	0x00000007	1.3
TAPE_LIBRARY_CLIENT_KMIPv11	0x00000008	1.3
TAPE_LIBRARY_CLIENT_KMIPv12	0x00000009	1.3
TAPE_LIBRARY_SERVER_KMIPv10	0x0000000a	1.3
TAPE_LIBRARY_SERVER_KMIPv11	0x0000000b	1.3
TAPE_LIBRARY_SERVER_KMIPv12	0x0000000c	1.3
SYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv10	0x0000000d	1.3
SYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv11	0x0000000e	1.3
SYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv12	0x0000000f	1.3
SYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv10	0x00000010	1.3
SYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv11	0x00000011	1.3
SYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv12	0x00000012	1.3
ASYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv10	0x00000013	1.3
ASYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv11	0x00000014	1.3
ASYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv12	0x00000015	1.3
ASYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv10	0x00000016	1.3
ASYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv11	0x00000017	1.3
ASYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv12	0x00000018	1.3
BASIC_CRYPTOGRAPHIC_CLIENT_KMIPv12	0x00000019	1.3
BASIC_CRYPTOGRAPHIC_SERVER_KMIPv12	0x0000001a	1.3
ADVANCED_CRYPTOGRAPHIC_CLIENT_KMIPv12	0x0000001b	1.3
ADVANCED_CRYPTOGRAPHIC_SERVER_KMIPv12	0x0000001c	1.3
RNG_CRYPTOGRAPHIC_CLIENT_KMIPv12	0x0000001d	1.3
RNG_CRYPTOGRAPHIC_SERVER_KMIPv12	0x0000001e	1.3
BASIC_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv10	0x0000001f	1.3
INTERMEDIATE_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv10	0x00000020	1.3
ADVANCED_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv10	0x00000021	1.3
BASIC_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv11	0x00000022	1.3
INTERMEDIATE_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv11	0x00000023	1.3
ADVANCED_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv11	0x00000024	1.3
BASIC_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv12	0x00000025	1.3
INTERMEDIATE_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv12	0x00000026	1.3
ADVANCED_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv12	0x00000027	1.3
SYMMETRIC_KEY_FOUNDRY_SERVER_KMIPv10	0x00000028	1.3
SYMMETRIC_KEY_FOUNDRY_SERVER_KMIPv11	0x00000029	1.3
SYMMETRIC_KEY_FOUNDRY_SERVER_KMIPv12	0x0000002a	1.3
OPAQUE_MANAGED_OBJECT_STORE_CLIENT_KMIPv10	0x0000002b	1.3

Continued on next page

Table 4 – continued from previous page

Name	Value	KMIP Version
OPAQUE_MANAGED_OBJECT_STORE_CLIENT_KMIPv11	0x0000002c	1.3
OPAQUE_MANAGED_OBJECT_STORE_CLIENT_KMIPv12	0x0000002d	1.3
OPAQUE_MANAGED_OBJECT_STORE_SERVER_KMIPv10	0x0000002e	1.3
OPAQUE_MANAGED_OBJECT_STORE_SERVER_KMIPv11	0x0000002f	1.3
OPAQUE_MANAGED_OBJECT_STORE_SERVER_KMIPv12	0x00000030	1.3
SUITE_B_MINLOS_128_CLIENT_KMIPv10	0x00000031	1.3
SUITE_B_MINLOS_128_CLIENT_KMIPv11	0x00000032	1.3
SUITE_B_MINLOS_128_CLIENT_KMIPv12	0x00000033	1.3
SUITE_B_MINLOS_128_SERVER_KMIPv10	0x00000034	1.3
SUITE_B_MINLOS_128_SERVER_KMIPv11	0x00000035	1.3
SUITE_B_MINLOS_128_SERVER_KMIPv12	0x00000036	1.3
SUITE_B_MINLOS_192_CLIENT_KMIPv10	0x00000037	1.3
SUITE_B_MINLOS_192_CLIENT_KMIPv11	0x00000038	1.3
SUITE_B_MINLOS_192_CLIENT_KMIPv12	0x00000039	1.3
SUITE_B_MINLOS_192_SERVER_KMIPv10	0x0000003a	1.3
SUITE_B_MINLOS_192_SERVER_KMIPv11	0x0000003b	1.3
SUITE_B_MINLOS_192_SERVER_KMIPv12	0x0000003c	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_CLIENT_KMIPv10	0x0000003d	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_CLIENT_KMIPv11	0x0000003e	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_CLIENT_KMIPv12	0x0000003f	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_SERVER_KMIPv10	0x00000040	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_SERVER_KMIPv11	0x00000041	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_SERVER_KMIPv12	0x00000042	1.3
HTTPS_CLIENT_KMIPv10	0x00000043	1.3
HTTPS_CLIENT_KMIPv11	0x00000044	1.3
HTTPS_CLIENT_KMIPv12	0x00000045	1.3
HTTPS_SERVER_KMIPv10	0x00000046	1.3
HTTPS_SERVER_KMIPv11	0x00000047	1.3
HTTPS_SERVER_KMIPv12	0x00000048	1.3
JSON_CLIENT_KMIPv10	0x00000049	1.3
JSON_CLIENT_KMIPv11	0x0000004a	1.3
JSON_CLIENT_KMIPv12	0x0000004b	1.3
JSON_SERVER_KMIPv10	0x0000004c	1.3
JSON_SERVER_KMIPv11	0x0000004d	1.3
JSON_SERVER_KMIPv12	0x0000004e	1.3
XML_CLIENT_KMIPv10	0x0000004f	1.3
XML_CLIENT_KMIPv11	0x00000050	1.3
XML_CLIENT_KMIPv12	0x00000051	1.3
XML_SERVER_KMIPv10	0x00000052	1.3
XML_SERVER_KMIPv11	0x00000053	1.3
XML_SERVER_KMIPv12	0x00000054	1.3
BASELINE_SERVER_BASIC_KMIPv13	0x00000055	1.3
BASELINE_SERVER_TLSv12_KMIPv13	0x00000056	1.3
BASELINE_CLIENT_BASIC_KMIPv13	0x00000057	1.3
BASELINE_CLIENT_TLSv12_KMIPv13	0x00000058	1.3
COMPLETE_SERVER_BASIC_KMIPv13	0x00000059	1.3
COMPLETE_SERVER_TLSv12_KMIPv13	0x0000005a	1.3
TAPE_LIBRARY_CLIENT_KMIPv13	0x0000005b	1.3
TAPE_LIBRARY_SERVER_KMIPv13	0x0000005c	1.3

Continued on next page

Table 4 – continued from previous page

Name	Value	KMIP Version
SYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv13	0x0000005d	1.3
SYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv13	0x0000005e	1.3
ASYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv13	0x0000005f	1.3
ASYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv13	0x00000060	1.3
BASIC_CRYPTOGRAPHIC_CLIENT_KMIPv13	0x00000061	1.3
BASIC_CRYPTOGRAPHIC_SERVER_KMIPv13	0x00000062	1.3
ADVANCED_CRYPTOGRAPHIC_CLIENT_KMIPv13	0x00000063	1.3
ADVANCED_CRYPTOGRAPHIC_SERVER_KMIPv13	0x00000064	1.3
RNG_CRYPTOGRAPHIC_CLIENT_KMIPv13	0x00000065	1.3
RNG_CRYPTOGRAPHIC_SERVER_KMIPv13	0x00000066	1.3
BASIC_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv13	0x00000067	1.3
INTERMEDIATE_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv13	0x00000068	1.3
ADVANCED_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv13	0x00000069	1.3
SYMMETRIC_KEY_FOUNDRY_SERVER_KMIPv13	0x0000006a	1.3
OPAQUE_MANAGED_OBJECT_STORE_CLIENT_KMIPv13	0x0000006b	1.3
OPAQUE_MANAGED_OBJECT_STORE_SERVER_KMIPv13	0x0000006c	1.3
SUITE_B_MINLOS_128_CLIENT_KMIPv13	0x0000006d	1.3
SUITE_B_MINLOS_128_SERVER_KMIPv13	0x0000006e	1.3
SUITE_B_MINLOS_192_CLIENT_KMIPv13	0x0000006f	1.3
SUITE_B_MINLOS_192_SERVER_KMIPv13	0x00000070	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_CLIENT_KMIPv13	0x00000071	1.3
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_SERVER_KMIPv13	0x00000072	1.3
HTTPS_CLIENT_KMIPv13	0x00000073	1.3
HTTPS_SERVER_KMIPv13	0x00000074	1.3
JSON_CLIENT_KMIPv13	0x00000075	1.3
JSON_SERVER_KMIPv13	0x00000076	1.3
XML_CLIENT_KMIPv13	0x00000077	1.3
XML_SERVER_KMIPv13	0x00000078	1.3
BASELINE_SERVER_BASIC_KMIPv14	0x00000079	1.4
BASELINE_SERVER_TLSv12_KMIPv14	0x0000007a	1.4
BASELINE_CLIENT_BASIC_KMIPv14	0x0000007b	1.4
BASELINE_CLIENT_TLSv12_KMIPv14	0x0000007c	1.4
COMPLETE_SERVER_BASIC_KMIPv14	0x0000007d	1.4
COMPLETE_SERVER_TLSv12_KMIPv14	0x0000007e	1.4
TAPE_LIBRARY_CLIENT_KMIPv14	0x0000007f	1.4
TAPE_LIBRARY_SERVER_KMIPv14	0x00000080	1.4
SYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv14	0x00000081	1.4
SYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv14	0x00000082	1.4
ASYMMETRIC_KEY_LIFECYCLE_CLIENT_KMIPv14	0x00000083	1.4
ASYMMETRIC_KEY_LIFECYCLE_SERVER_KMIPv14	0x00000084	1.4
BASIC_CRYPTOGRAPHIC_CLIENT_KMIPv14	0x00000085	1.4
BASIC_CRYPTOGRAPHIC_SERVER_KMIPv14	0x00000086	1.4
ADVANCED_CRYPTOGRAPHIC_CLIENT_KMIPv14	0x00000087	1.4
ADVANCED_CRYPTOGRAPHIC_SERVER_KMIPv14	0x00000088	1.4
RNG_CRYPTOGRAPHIC_CLIENT_KMIPv14	0x00000089	1.4
RNG_CRYPTOGRAPHIC_SERVER_KMIPv14	0x0000008a	1.4
BASIC_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv14	0x0000008b	1.4
INTERMEDIATE_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv14	0x0000008c	1.4
ADVANCED_SYMMETRIC_KEY_FOUNDRY_CLIENT_KMIPv14	0x0000008d	1.4

Continued on next page

Table 4 – continued from previous page

Name	Value	KMIP Version
SYMMETRIC_KEY_FOUNDRY_SERVER_KMIPv14	0x0000008e	1.4
OPAQUE_MANAGED_OBJECT_STORE_CLIENT_KMIPv14	0x0000008f	1.4
OPAQUE_MANAGED_OBJECT_STORE_SERVER_KMIPv14	0x00000090	1.4
SUITE_B_MINLOS_128_CLIENT_KMIPv14	0x00000091	1.4
SUITE_B_MINLOS_128_SERVER_KMIPv14	0x00000092	1.4
SUITE_B_MINLOS_192_CLIENT_KMIPv14	0x00000093	1.4
SUITE_B_MINLOS_192_SERVER_KMIPv14	0x00000094	1.4
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_CLIENT_KMIPv14	0x00000095	1.4
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_SERVER_KMIPv14	0x00000096	1.4
HTTPS_CLIENT_KMIPv14	0x00000097	1.4
HTTPS_SERVER_KMIPv14	0x00000098	1.4
JSON_CLIENT_KMIPv14	0x00000099	1.4
JSON_SERVER_KMIPv14	0x0000009a	1.4
XML_CLIENT_KMIPv14	0x0000009b	1.4
XML_SERVER_KMIPv14	0x0000009c	1.4
COMPLETE_SERVER_BASIC	0x00000104	2.0
COMPLETE_SERVER_TLSv12	0x00000105	2.0
TAPE_LIBRARY_CLIENT	0x00000106	2.0
TAPE_LIBRARY_SERVER	0x00000107	2.0
SYMMETRIC_KEY_LIFECYCLE_CLIENT	0x00000108	2.0
SYMMETRIC_KEY_LIFECYCLE_SERVER	0x00000109	2.0
ASYMMETRIC_KEY_LIFECYCLE_CLIENT	0x0000010A	2.0
ASYMMETRIC_KEY_LIFECYCLE_SERVER	0x0000010B	2.0
BASIC_CRYPTOGRAPHIC_CLIENT	0x0000010C	2.0
BASIC_CRYPTOGRAPHIC_SERVER	0x0000010D	2.0
ADVANCED_CRYPTOGRAPHIC_CLIENT	0x0000010E	2.0
ADVANCED_CRYPTOGRAPHIC_SERVER	0x0000010F	2.0
RNG_CRYPTOGRAPHIC_CLIENT	0x00000110	2.0
RNG_CRYPTOGRAPHIC_SERVER	0x00000111	2.0
BASIC_SYMMETRIC_KEY_FOUNDRY_CLIENT	0x00000112	2.0
INTERMEDIATE_SYMMETRIC_KEY_FOUNDRY_CLIENT	0x00000113	2.0
ADVANCED_SYMMETRIC_KEY_FOUNDRY_CLIENT	0x00000114	2.0
SYMMETRIC_KEY_FOUNDRY_SERVER	0x00000115	2.0
OPAQUE_MANAGED_OBJECT_STORE_CLIENT	0x00000116	2.0
OPAQUE_MANAGED_OBJECT_STORE_SERVER	0x00000117	2.0
SUITE_B_MINLOS_128_CLIENT	0x00000118	2.0
SUITE_B_MINLOS_128_SERVER	0x00000119	2.0
SUITE_B_MINLOS_192_CLIENT	0x0000011A	2.0
SUITE_B_MINLOS_192_SERVER	0x0000011B	2.0
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_CLIENT	0x0000011C	2.0
STORAGE_ARRAY_WITH_SELF_ENCRYPTING_DRIVE_SERVER	0x0000011D	2.0
HTTPS_CLIENT	0x0000011E	2.0
HTTPS_SERVER	0x0000011F	2.0
JSON_CLIENT	0x00000120	2.0
JSON_SERVER	0x00000121	2.0
XML_CLIENT	0x00000122	2.0
XML_SERVER	0x00000123	2.0
AES_XTS_CLIENT	0x00000124	2.0
AES_XTS_SERVER	0x00000125	2.0

Continued on next page

Table 4 – continued from previous page

Name	Value	KMIP Version
QUANTUM_SAFE_CLIENT	0x00000126	2.0
QUANTUM_SAFE_SERVER	0x00000127	2.0
PKCS11_CLIENT	0x00000128	2.0
PKCS11_SERVER	0x00000129	2.0
BASELINE_CLIENT	0x0000012A	2.0
BASELINE_SERVER	0x0000012B	2.0
COMPLETE_SERVER	0x0000012C	2.0

profile_version (dict) (2.0) A dictionary containing the major and minor version numbers of a KMIP profile. Often used with the *profile_information* structure.

```
>>> profile_version = {
...     'profile_version_major': 1,
...     'profile_version_minor': 0
... }
```

Key	Value	KMIP Version
profile_version_major	int	2.0
profile_version_minor	int	2.0

put_function (enum) (1.0) An enumeration specifying the state of an object being pushed by the Put operation.

```
>>> from kmip import enums
>>> enums.PutFunction.NEW
<PutFunction.NEW: 1>
```

Name	Value	KMIP Version
NEW	0x00000001	1.0
REPLACE	0x00000002	1.0

query_function (enum) (1.0) An enumeration specifying the information to include in a Query operation response.

```
>>> from kmip import enums
>>> enums.QueryFunction.QUERY_OPERATIONS
<QueryFunction.QUERY_OPERATIONS: 1>
```

Name	Value	KMIP Version
QUERY_OPERATIONS	0x00000001	1.0
QUERY_OBJECTS	0x00000002	1.0
QUERY_SERVER_INFORMATION	0x00000003	1.0
QUERY_APPLICATION_NAMESPACES	0x00000004	1.0
QUERY_EXTENSION_LIST	0x00000005	1.1
QUERY_EXTENSION_MAP	0x00000006	1.1
QUERY_ATTESTATION_TYPES	0x00000007	1.2
QUERY_RNGS	0x00000008	1.3
QUERY_VALIDATIONS	0x00000009	1.3
QUERY_PROFILES	0x0000000a	1.3
QUERY_CAPABILITIES	0x0000000b	1.3
QUERY_CLIENT_REGISTRATION_METHODS	0x0000000c	1.3
QUERY_DEFAULTS_INFORMATION	0x0000000D	2.0
QUERY_STORAGE_PROTECTION_MASKS	0x0000000E	2.0

recommended_curve (enum) (1.0) An enumeration specifying a recommended curve for an elliptic curve algorithm. Used often as an asymmetric key value attribute.

```
>>> from kmip import enums
>>> enums.RecommendedCurve.P_192
<RecommendedCurve.P_192: 1>
```

Name	Value	KMIP Version
P_192	0x00000001	1.0
K_163	0x00000002	1.0
B_163	0x00000003	1.0
P_224	0x00000004	1.0
K_233	0x00000005	1.0
B_233	0x00000006	1.0
P_256	0x00000007	1.0
K_283	0x00000008	1.0
B_283	0x00000009	1.0
P_384	0x0000000a	1.0
K_409	0x0000000b	1.0
B_409	0x0000000c	1.0
P_521	0x0000000d	1.0
K_571	0x0000000e	1.0
B_571	0x0000000f	1.0
SECP112R1	0x00000010	1.2
SECP112R2	0x00000011	1.2
SECP128R1	0x00000012	1.2
SECP128R2	0x00000013	1.2
SECP160K1	0x00000014	1.2
SECP160R1	0x00000015	1.2
SECP160R2	0x00000016	1.2
SECP191K1	0x00000017	1.2
SECP224K1	0x00000018	1.2
SECP256K1	0x00000019	1.2
SECT113R1	0x0000001a	1.2

Continued on next page

Table 5 – continued from previous page

Name	Value	KMIP Version
SECT113R2	0x0000001b	1.2
SECT131R1	0x0000001c	1.2
SECT131R2	0x0000001d	1.2
SECT163R1	0x0000001e	1.2
SECT193R1	0x0000001f	1.2
SECT193R2	0x00000020	1.2
SECT239K1	0x00000021	1.2
ANSIX9P192V2	0x00000022	1.2
ANSIX9P192V3	0x00000023	1.2
ANSIX9P239V1	0x00000024	1.2
ANSIX9P239V2	0x00000025	1.2
ANSIX9P239V3	0x00000026	1.2
ANSIX9C2PNB163V1	0x00000027	1.2
ANSIX9C2PNB163V2	0x00000028	1.2
ANSIX9C2PNB163V3	0x00000029	1.2
ANSIX9C2PNB176V1	0x0000002a	1.2
ANSIX9C2TNB191V1	0x0000002b	1.2
ANSIX9C2TNB191V2	0x0000002c	1.2
ANSIX9C2TNB191V3	0x0000002d	1.2
ANSIX9C2PNB208W1	0x0000002e	1.2
ANSIX9C2TNB239V1	0x0000002f	1.2
ANSIX9C2TNB239V2	0x00000030	1.2
ANSIX9C2TNB239V3	0x00000031	1.2
ANSIX9C2PNB272W1	0x00000032	1.2
ANSIX9C2PNB304W1	0x00000033	1.2
ANSIX9C2TNB359V1	0x00000034	1.2
ANSIX9C2PNB368W1	0x00000035	1.2
ANSIX9C2TNB431R1	0x00000036	1.2
BRAINPOOLP160R1	0x00000037	1.2
BRAINPOOLP160T1	0x00000038	1.2
BRAINPOOLP192R1	0x00000039	1.2
BRAINPOOLP192T1	0x0000003a	1.2
BRAINPOOLP224R1	0x0000003b	1.2
BRAINPOOLP224T1	0x0000003c	1.2
BRAINPOOLP256R1	0x0000003d	1.2
BRAINPOOLP256T1	0x0000003e	1.2
BRAINPOOLP320R1	0x0000003f	1.2
BRAINPOOLP320T1	0x00000040	1.2
BRAINPOOLP384R1	0x00000041	1.2
BRAINPOOLP384T1	0x00000042	1.2
BRAINPOOLP512R1	0x00000043	1.2
BRAINPOOLP512T1	0x00000044	1.2
CURVE25519	0x00000045	2.0
CURVE448	0x00000046	2.0

result_reason (enum) (1.0) An enumeration specifying the reason for the result status of an operation. Used usually if an operation results in a failure.

```
>>> from kmip import enums
>>> enums.ResultReason.ITEM_NOT_FOUND
```

(continues on next page)

(continued from previous page)

```
<ResultReason.ITEM_NOT_FOUND: 1>
```

Name	Value	KMIP Version
ITEM_NOT_FOUND	0x00000001	1.0
RESPONSE_TOO_LARGE	0x00000002	1.0
AUTHENTICATION_NOT_SUCCESSFUL	0x00000003	1.0
INVALID_MESSAGE	0x00000004	1.0
OPERATION_NOT_SUPPORTED	0x00000005	1.0
MISSING_DATA	0x00000006	1.0
INVALID_FIELD	0x00000007	1.0
FEATURE_NOT_SUPPORTED	0x00000008	1.0
OPERATION_CANCELED_BY_REQUESTER	0x00000009	1.0
CRYPTOGRAPHIC_FAILURE	0x0000000a	1.0
ILLEGAL_OPERATION	0x0000000b	1.0
PERMISSION_DENIED	0x0000000c	1.0
OBJECT_ARCHIVED	0x0000000d	1.0
INDEX_OUT_OF_BOUNDS	0x0000000e	1.0
APPLICATION_NAMESPACE_NOT_SUPPORTED	0x0000000f	1.0
KEY_FORMAT_TYPE_NOT_SUPPORTED	0x00000010	1.0
KEY_COMPRESSION_TYPE_NOT_SUPPORTED	0x00000011	1.0
ENCODING_OPTION_ERROR	0x00000012	1.1
KEY_VALUE_NOT_PRESENT	0x00000013	1.2
ATTESTATION_REQUIRED	0x00000014	1.2
ATTESTATION_FAILED	0x00000015	1.2
SENSITIVE	0x00000016	1.4
NOT_EXTRACTABLE	0x00000017	1.4
OBJECT_ALREADY_EXISTS	0x00000018	1.4
INVALID_TICKET	0x00000019	2.0
USAGE_LIMIT_EXCEEDED	0x0000001A	2.0
NUMERIC_RANGE	0x0000001B	2.0
INVALID_DATA_TYPE	0x0000001C	2.0
READ_ONLY_ATTRIBUTE	0x0000001D	2.0
MULTI_VALUED_ATTRIBUTE	0x0000001E	2.0
UNSUPPORTED_ATTRIBUTE	0x0000001F	2.0
ATTRIBUTE_INSTANCE_NOT_FOUND	0x00000020	2.0
ATTRIBUTE_NOT_FOUND	0x00000021	2.0
ATTRIBUTE_READ_ONLY	0x00000022	2.0
ATTRIBUTE_SINGLE_VALUED	0x00000023	2.0
BAD_CRYPTOGRAPHIC_PARAMETERS	0x00000024	2.0
BAD_PASSWORD	0x00000025	2.0
CODEC_ERROR	0x00000026	2.0
ILLEGAL_OBJECT_TYPE	0x00000028	2.0
INCOMPATIBLE_CRYPTOGRAPHIC_USAGE_MASK	0x00000029	2.0
INTERNAL_SERVER_ERROR	0x0000002A	2.0
INVALID_ASYNCHRONOUS_CORRELATION_VALUE	0x0000002B	2.0
INVALID_ATTRIBUTE	0x0000002C	2.0
INVALID_ATTRIBUTE_VALUE	0x0000002D	2.0
INVALID_CORRELATION_VALUE	0x0000002E	2.0
INVALID_CSR	0x0000002F	2.0

Continued on next page

Table 6 – continued from previous page

Name	Value	KMIP Version
INVALID_OBJECT_TYPE	0x00000030	2.0
KEY_WRAP_TYPE_NOT_SUPPORTED	0x00000032	2.0
MISSING_INITIALIZATION_VECTOR	0x00000034	2.0
NON_UNIQUE_NAME_ATTRIBUTE	0x00000035	2.0
OBJECT_DESTROYED	0x00000036	2.0
OBJECT_NOT_FOUND	0x00000037	2.0
NOT_AUTHORISED	0x00000039	2.0
SERVER_LIMIT_EXCEEDED	0x0000003A	2.0
UNKNOWN_ENUMERATION	0x0000003B	2.0
UNKNOWN_MESSAGE_EXTENSION	0x0000003C	2.0
UNKNOWN_TAG	0x0000003D	2.0
UNSUPPORTED_CRYPTOGRAPHIC_PARAMETERS	0x0000003E	2.0
UNSUPPORTED_PROTOCOL_VERSION	0x0000003F	2.0
WRAPPING_OBJECT_ARCHIVED	0x00000040	2.0
WRAPPING_OBJECT_DESTROYED	0x00000041	2.0
WRAPPING_OBJECT_NOT_FOUND	0x00000042	2.0
WRONG_KEY_LIFECYCLE_STATE	0x00000043	2.0
PROTECTION_STORAGE_UNAVAILABLE	0x00000044	2.0
PKCS11_CODEC_ERROR	0x00000045	2.0
PKCS11_INVALID_FUNCTION	0x00000046	2.0
PKCS11_INVALID_INTERFACE	0x00000047	2.0
GENERAL_FAILURE	0x00000100	1.0

result_status (enum) (1.0) An enumeration specifying the result of an operation. Used in every operation response.

```
>>> from kmip import enums
>>> enums.ResultStatus.OPERATION_FAILED
<ResultStatus.OPERATION_FAILED: 1>
```

Name	Value	KMIP Version
SUCCESS	0x00000000	1.0
OPERATION_FAILED	0x00000001	1.0
OPERATION_PENDING	0x00000002	1.0
OPERATION_UNDONE	0x00000003	1.0

revocation_reason_code (enum) (1.0) An enumeration specifying the reason for the revocation of a managed object.

```
>>> from kmip import enums
>>> enums.RevocationReasonCode.KEY_COMPROMISE
<RevocationReasonCode.KEY_COMPROMISE: 2>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.0
KEY_COMPROMISE	0x00000002	1.0
CA_COMPROMISE	0x00000003	1.0
AFFILIATION_CHANGED	0x00000004	1.0
SUPERSEDED	0x00000005	1.0
CESSATION_OF_OPERATION	0x00000006	1.0
PRIVILEGE_WITHDRAWN	0x00000007	1.0

rng_algorithm (enum) (1.3) An enumeration specifying an algorithm for random number generation. Used often to describe a random number generator.

```
>>> from kmip import enums
>>> enums.RNGAlgorithm.DRBG
<RNGAlgorithm.DRBG: 3>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
FIPS186_2	0x00000002	1.3
DRBG	0x00000003	1.3
NRBG	0x00000004	1.3
ANSI_X931	0x00000005	1.3
ANSI_X962	0x00000006	1.3

rng_mode (enum) (1.3) An enumeration specifying the mode for random number generation. Used often to describe a random number generator.

```
>>> from kmip import enums
>>> enums.RNGMode.SHARED_INSTANTIATION
<RNGMode.SHARED_INSTANTIATION: 2>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
SHARED_INSTANTIATION	0x00000002	1.3
NON_SHARED_INSTANTIATION	0x00000003	1.3

rng_parameters (dict) (1.3) A dictionary containing information about a random number generator supported by a KMIP server. Often obtained from the Query operation response.

```
>>> from kmip import enums
>>> rng_parameters = {
...     'rng_algorithm': enums.RNGAlgorithm.ANSI_X931,
...     'cryptographic_algorithm': enums.CryptographicAlgorithm.AES,
...     'cryptographic_length': 256,
...     'hashing_algorithm': enums.HashingAlgorithm.SHA_256,
...     'drbg_algorithm': enums.DRBGAlgorithm.HASH,
...     'recommended_curve': enums.RecommendedCurve.B_163,
...     'fips186_variation': enums.FIPS186Variation.X_ORIGINAL,
...     'prediction_resistance': True
... }
```

Key	Value	KMIP Version
rng_algorithm	enum	1.3
cryptographic_algorithm	enum	1.3
cryptographic_length	int	1.3
hashing_algorithm	enum	1.3
drbg_algorithm	enum	1.3
recommended_curve	enum	1.3
fips186_variation	enum	1.3
prediction_resistance	bool	1.3

secret_data_type (enum) (1.0) An enumeration specifying the type of a secret data object.

```
>>> from kmip import enums
>>> enums.SecretDataType.PASSWORD
<SecretDataType.PASSWORD: 1>
```

Name	Value	KMIP Version
PASSWORD	0x00000001	1.0
SEED	0x00000002	1.0

server_information (str) (1.0) A string containing additional information on the vendor associated with a KMIP appliance. Often obtained with the Query operation.

shredding_algorithm (enum) (1.3) An enumeration specifying the type of shredding algorithm supported by a key server. Used often as part of the response to a Query request.

```
>>> from kmip import enums
>>> enums.ShreddingAlgorithm.CRYPTOGRAPHIC
<ShreddingAlgorithm.CRYPTOGRAPHIC: 2>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
CRYPTOGRAPHIC	0x00000002	1.3
UNSUPPORTED	0x00000003	1.3

split_key_method (enum) (1.0) An enumeration specifying the method used to split a key. Used as an attribute for split key objects and as a parameter to the CreateSplitKey operation.

```
>>> from kmip import enums
>>> enums.SplitKeyMethod.XOR
<SplitKeyMethod.XOR: 1>
```

Name	Value	KMIP Version
XOR	0x00000001	1.0
POLYNOMIAL_SHARING_GF_2_16	0x00000002	1.0
POLYNOMIAL_SHARING_PRIME_FIELD	0x00000003	1.0
POLYNOMIAL_SHARING_GF_2_8	0x00000004	1.2

state (enum) (1.0) An enumeration specifying the state of a managed object. Used as an attribute for every managed object on a key server.

```
>>> from kmip import enums
>>> enums.State.ACTIVE
<State.ACTIVE: 2>
```

Name	Value	KMIP Version
PRE_ACTIVE	0x00000001	1.0
ACTIVE	0x00000002	1.0
DEACTIVATED	0x00000003	1.0
COMPROMISED	0x00000004	1.0
DESTROYED	0x00000005	1.0
DESTROYED_COMPROMISED	0x00000006	1.0

storage_status (enum) (1.0) An enumeration specifying the state of a stored object. Used as a filter for the Locate operation.

```
>>> from kmip import enums
>>> enums.StorageStatus.ARCHIVAL_STORAGE
<StorageStatus.ARCHIVAL_STORAGE: 2>
```

Name	Value	KMIP Version
ONLINE_STORAGE	0x00000001	1.0
ARCHIVAL_STORAGE	0x00000002	1.0
DESTROYED_STORAGE	0x00000004	2.0

unique_identifier (str) (1.0) A string representing a unique, global identifier for a managed object created or registered with a key management appliance. Each managed object is represented by one unique identifier, which can be used in a variety of operations to access the object or the object metadata. This identifier is assigned when the object is first created or registered and cannot be changed.

unwrap_mode (enum) (1.3) An enumeration specifying an unwrapping mode supported by the server. Used often as part of the response to a Query request.

```
>>> from kmip import enums
>>> enums.UnwrapMode.PROCESSED
<UnwrapMode.PROCESSED: 2>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
PROCESSED	0x00000002	1.3
NOT_PROCESSED	0x00000003	1.3

usage_limits_unit (enum) (1.0) An enumeration specifying the units for a usage limit on a managed object.

```
>>> from kmip import enums
>>> enums.UsageLimitsUnit.BYTE
<UsageLimitsUnit.BYTE: 1>
```

Name	Value	KMIP Version
BYTE	0x00000001	1.0
OBJECT	0x00000002	1.0

validation_authority_type (enum) (1.3) An enumeration specifying a validation authority type supported by the server. Used often as part of the response to a Query request.

```
>>> from kmip import enums
>>> enums.ValidationAuthorityType.COMMON_CRITERIA
<ValidationAuthorityType.COMMON_CRITERIA: 3>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
NIST_CMVP	0x00000002	1.3
COMMON_CRITERIA	0x00000003	1.3

validation_information (dict) (1.3) A dictionary containing information about a formal validation. Often obtained from the Query operation response.

```
>>> from kmip import enums
>>> validation_information = {
...     'validation_authority_type': enums.ValidationAuthorityType.COMMON_
↪CRITERIA,
...     'validation_authority_country': 'US',
...     'validation_profile': [
...         'Example Profile 1',
...         'Example Profile 2'
...     ]
... }
```

Key	Value	KMIP Version
validation_authority_type	enum	1.3
validation_authority_country	string	1.3
validation_authority_uri	string	1.3
validation_version_major	int	1.3
validation_version_minor	int	1.3
validation_type	enum	1.3
validation_level	int	1.3
validation_certificate_identifier	string	1.3
validation_certificate_uri	string	1.3
validation_vendor_uri	string	1.3
validation_profile	list	1.3

validation_type (enum) (1.3) An enumeration specifying a validation type supported by the server. Used often as part of the response to a Query request.

```
>>> from kmip import enums
>>> enums.ValidationType.HARDWARE
<ValidationType.HARDWARE: 2>
```

Name	Value	KMIP Version
UNSPECIFIED	0x00000001	1.3
HARDWARE	0x00000002	1.3
SOFTWARE	0x00000003	1.3
FIRMWARE	0x00000004	1.3
HYBRID	0x00000005	1.3

validity_indicator (enum) (1.0) An enumeration specifying the validity of an operation or object. Used as a return value for various operations.

```
>>> from kmip import enums
>>> enums.ValidityIndicator.VALID
<ValidityIndicator.VALID: 1>
```

Name	Value	KMIP Version
VALID	0x00000001	1.0
INVALID	0x00000002	1.0
UNKNOWN	0x00000003	1.0

vendor_identification (str) (1.0) A string containing identification information on the vendor associated with a KMIP appliance. Often obtained with the Query operation.

wrapping_method (enum) (1.0) An enumeration representing a key wrapping mechanism. Used in various key wrapping metadata structures.

```
>>> from kmip import enums
>>> enums.WrappingMethod.ENCRYPT
<WrappingMethod.ENCRYPT: 1>
```

Name	Value	KMIP Version
ENCRYPT	0x00000001	1.0
MAC_SIGN	0x00000002	1.0
ENCRYPT_THEN_MAC_SIGN	0x00000003	1.0
MAC_SIGN_THEN_ENCRYPT	0x00000004	1.0
TR_31	0x00000005	1.0

Python Module Index

k

Index

L

M

N

O

P

Q

R

S

U

V

W