Release v19.0.0 (What’s new?).

pyOpenSSL is a rather thin wrapper around (a subset of) the OpenSSL library. With thin wrapper we mean that a lot of the object methods do nothing more than calling a corresponding function in the OpenSSL library.
1.1 Introduction

1.1.1 History

pyOpenSSL was originally created by Martin Sjögren because the SSL support in the standard library in Python 2.1 (the contemporary version of Python when the pyOpenSSL project was begun) was severely limited. Other OpenSSL wrappers for Python at the time were also limited, though in different ways.

Later it was maintained by Jean-Paul Calderone who among other things managed to make pyOpenSSL a pure Python project which the current maintainers are very grateful for.

Over the time the standard library’s ssl module improved, never reaching the completeness of pyOpenSSL’s API coverage. Despite PEP 466 many useful features remain Python 3-only and pyOpenSSL remains the only alternative for full-featured TLS code across all noteworthy Python versions from 2.7 through 3.5 and PyPy.

1.1.2 Development

pyOpenSSL is collaboratively developed by the Python Cryptography Authority (PyCA) that also maintains the low-level bindings called cryptography.

Current maintainer and release manager is Hynek Schlawack.

1.1.3 Contributing

First of all, thank you for your interest in contributing to pyOpenSSL! This project has no company backing its development therefore we’re dependent on help by the community.
Filing bug reports

Bug reports are very welcome. Please file them on the GitHub issue tracker. Good bug reports come with extensive descriptions of the error and how to reproduce it. Reporters are strongly encouraged to include an short, self contained, correct example.

Patches

All patches to pyOpenSSL should be submitted in the form of pull requests to the main pyOpenSSL repository, pyca/pyopenssl. These pull requests should satisfy the following properties:

Code

- The pull request should focus on one particular improvement to pyOpenSSL. Create different pull requests for unrelated features or bugfixes.
- Code should follow PEP 8, especially in the “do what code around you does” sense. Follow OpenSSL naming for callables whenever possible is preferred.
- Pull requests that introduce code must test all new behavior they introduce as well as for previously untested or poorly tested behavior that they touch.
- Pull requests are not allowed to break existing tests. We usually don’t comment on pull requests that are breaking the CI because we consider them work in progress. Please note that not having 100% code coverage for the code you wrote/touched also causes our CI to fail.

Documentation

When introducing new functionality, please remember to write documentation.

- New functions and methods should have a docstring describing what they do, what parameters they takes, what types those parameters are, and what they return.

```python
def dump_publickey(type, pkey):
    """
    Dump a public key to a buffer.

    :param type: The file type (one of :data:`FILETYPE_PEM` or :data:`FILETYPE_ASN1`).
    :param PKey pkey: The PKey to dump.

    :return: The buffer with the dumped key in it.
    :rtype: bytes
    """
```

Don’t forget to add an .. auto(function|class|method):: statement to the relevant API document found in doc/api/ to actually add your function to the Sphinx documentation.

- Do not use :py: prefixes when cross-linking (Python is default). Do not use the generic :data: or :obj:. Instead use more specific types like :class:, :func: or :meth: if applicable.

- Pull requests that introduce features or fix bugs should note those changes in the CHANGELOG.rst file. Please add new entries to the top of the current Changes section followed by a line linking to the relevant pull request:
- Added `OpenSSL.crypto.some_func()` to do something awesome.
  [¹#1 <https://github.com/pyca/pyopenssl/pull/1>`_]

- Use semantic newlines in reStructuredText files (files ending in .rst).

## Review

Finally, pull requests must be reviewed before merging. This process mirrors the cryptography code review process. Everyone can perform reviews; this is a very valuable way to contribute, and is highly encouraged.

Pull requests are merged by members of PyCA. They should, of course, keep all the requirements detailed in this document as well as the pyca/cryptography merge requirements in mind.

The final responsibility for the reviewing of merged code lies with the person merging it. Since pyOpenSSL is a sensitive project from a security perspective, reviewers are strongly encouraged to take this review and merge process very seriously.

## Finding Help

If you need any help with the contribution process, you’ll find us hanging out at #cryptography-dev on Freenode IRC. You can also ask questions on our mailing list.

Please note that this project is released with a Contributor Code of Conduct. By participating in this project you agree to abide by its terms.

## Security

If you feel that you found a security-relevant bug that you would prefer to discuss in private, please send us a GPG-encrypted e-mail.

The maintainer can be reached at hs@ox.cx and his GPG key ID is 0xAE2536227F69F181 (Fingerprint: C2A0 4F86 ACE2 8ADC F817 DBB7 AE25 3622 7F69 F181). Feel free to cross-check this information with Keybase.

## 1.2 Installation

To install pyOpenSSL:

```
$ pip install pyopenssl
```

If you are installing in order to develop on pyOpenSSL, move to the root directory of a pyOpenSSL checkout, and run:

```
$ pip install -e .
```

**Warning:** As of 0.14, pyOpenSSL is a pure-Python project. That means that if you encounter any kind of compiler errors, pyOpenSSL’s bugtracker is the wrong place to report them because we cannot help you.

Please take the time to read the errors and report them/ask help from the appropriate project. The most likely culprit being cryptography that contains OpenSSL’s library bindings.
1.2.1 Supported OpenSSL Versions

pyOpenSSL supports the same platforms and releases as the upstream cryptography project does. Currently that means:

- 1.0.1
- 1.0.2
- 1.1.0

If you need support for older releases, the following pinned versions will work:

- OpenSSL 0.9.8: 'pyOpenSSL<17.0' 'cryptography<1.4'
- OpenSSL 1.0.0: 'pyOpenSSL<17.1' 'cryptography<1.7'

You can always find out the versions of pyOpenSSL, cryptography, and the linked OpenSSL by running `python -m OpenSSL.debug`.

1.2.2 Documentation

The documentation is written in reStructuredText and built using Sphinx:

```
$ cd doc
$ make html
```

1.3 OpenSSL — Python interface to OpenSSL

This package provides a high-level interface to the functions in the OpenSSL library. The following modules are defined:

1.3.1 crypto — Generic cryptographic module

**Note:** `pyca/cryptography` is likely a better choice than using this module. It contains a complete set of cryptographic primitives as well as a significantly better and more powerful X509 API. If necessary you can convert to and from cryptography objects using the `to_cryptography` and `from_cryptography` methods on X509, X509Req, CRL, and PKey.

**Elliptic curves**

`OpenSSL.crypto.get_elliptic_curves()`  
Return a set of objects representing the elliptic curves supported in the OpenSSL build in use.

The curve objects have a `unicode name` attribute by which they identify themselves.

The curve objects are useful as values for the argument accepted by `Context.set_tmp_ecdh()` to specify which elliptical curve should be used for ECDHE key exchange.

`OpenSSL.crypto.get_elliptic_curve(name)`  
Return a single curve object selected by name.

See `get_elliptic_curves()` for information about curve objects.
Parameters name (unicode) – The OpenSSL short name identifying the curve object to retrieve. If the named curve is not supported then ValueError is raised.

Serialization and deserialization

The following serialization functions take one of these constants to determine the format.

OpenSSL.crypto.FILETYPE_PEM

FILETYPE_PEM serializes data to a Base64-encoded encoded representation of the underlying ASN.1 data structure. This representation includes delimiters that define what data structure is contained within the Base64-encoded block: for example, for a certificate, the delimiters are -----BEGIN CERTIFICATE----- and -----END CERTIFICATE-----.

OpenSSL.crypto.FILETYPE_ASN1

FILETYPE_ASN1 serializes data to the underlying ASN.1 data structure. The format used by FILETYPE_ASN1 is also sometimes referred to as DER.

Certificates

OpenSSL.crypto.dump_certificate(type, cert)

Dump the certificate cert into a buffer string encoded with the type type.

Parameters

• type – The file type (one of FILETYPE_PEM, FILETYPE_ASN1, or FILETYPE_TEXT)
• cert – The certificate to dump

Returns The buffer with the dumped certificate in

OpenSSL.crypto.load_certificate(type, buffer)

Load a certificate (X509) from the string buffer encoded with the type type.

Parameters

• type – The file type (one of FILETYPE_PEM, FILETYPE_ASN1)
• buffer (bytes) – The buffer the certificate is stored in

Returns The X509 object

Certificate signing requests

OpenSSL.crypto.dump_certificate_request(type, req)

Dump the certificate request req into a buffer string encoded with the type type.

Parameters

• type – The file type (one of FILETYPE_PEM, FILETYPE_ASN1)
• req – The certificate request to dump

Returns The buffer with the dumped certificate request in

OpenSSL.crypto.load_certificate_request(type, buffer)

Load a certificate request (X509Req) from the string buffer encoded with the type type.

Parameters
• **type** – The file type (one of FILETYPE_PEM, FILETYPE_ASN1)
• **buffer** – The buffer the certificate request is stored in

**Returns** The X509Req object

### Private keys

**OpenSSL.crypto.dump_privatekey** *(type, pkey, cipher=None, passphrase=None)*

Dump the private key `pkey` into a buffer string encoded with the type `type`. Optionally (if `type` is `FILETYPE_PEM`) encrypting it using `cipher` and `passphrase`.

**Parameters**

• **type** – The file type (one of `FILETYPE_PEM`, `FILETYPE_ASN1`, or `FILETYPE_TEXT`)
• **pkey** (PKey) – The PKey to dump
• **cipher** – (optional) if encrypted PEM format, the cipher to use
• **passphrase** – (optional) if encrypted PEM format, this can be either the passphrase to use, or a callback for providing the passphrase.

**Returns** The buffer with the dumped key in

**Return type** bytes

**OpenSSL.crypto.load_privatekey** *(type, buffer, passphrase=None)*

Load a private key (PKey) from the string `buffer` encoded with the type `type`.

**Parameters**

• **type** – The file type (one of `FILETYPE_PEM`, `FILETYPE_ASN1`)
• **buffer** – The buffer the key is stored in
• **passphrase** – (optional) if encrypted PEM format, this can be either the passphrase to use, or a callback for providing the passphrase.

**Returns** The PKey object

### Public keys

**OpenSSL.crypto.dump_publickey** *(type, pkey)*

Dump a public key to a buffer.

**Parameters**

• **type** – The file type (one of `FILETYPE_PEM` or `FILETYPE_ASN1`).
• **pkey** (PKey) – The public key to dump

**Returns** The buffer with the dumped key in it.

**Return type** bytes

**OpenSSL.crypto.load_publickey** *(type, buffer)*

Load a public key from a buffer.

**Parameters**

• **type** – The file type (one of `FILETYPE_PEM`, `FILETYPE_ASN1`).
• buffer (A Python string object, either unicode or bytestring.)
  – The buffer the key is stored in.

  Returns The PKey object.

  Return type PKey

Certificate revocation lists

OpenSSL.crypto.dump_crl(type, crl)
Dump a certificate revocation list to a buffer.

Parameters
  • type – The file type (one of FILETYPE_PEM, FILETYPE_ASN1, or
           FILETYPE_TEXT).
  • crl (CRL) – The CRL to dump.

  Returns The buffer with the CRL.

  Return type bytes

OpenSSL.crypto.load_crl(type, buffer)
Load Certificate Revocation List (CRL) data from a string buffer. buffer encod
ed with the type type.

Parameters
  • type – The file type (one of FILETYPE_PEM, FILETYPE_ASN1)
  • buffer – The buffer the CRL is stored in

  Returns The PKey object

OpenSSL.crypto.load_pkcs7_data(type, buffer)
Load pkcs7 data from the string buffer encoded with the type type.

Parameters
  • type – The file type (one of FILETYPE_PEM or FILETYPE_ASN1)
  • buffer – The buffer with the pkcs7 data.

  Returns The PKCS7 object

OpenSSL.crypto.load_pkcs12(buffer, passphrase=None)
Load pkcs12 data from the string buffer. If the pkcs12 structure is encryp
ted, a passphrase must be included. The MAC is always checked and thus required.

  See also the man page for the C function PKCS12_parse().

Parameters
  • buffer – The buffer the certificate is stored in
  • passphrase – (Optional) The password to decrypt the PKCS12 lump

  Returns The PKCS12 object

Signing and verifying signatures

OpenSSL.crypto.sign(pkey, data, digest)
Sign a data string using the given key and message digest.
Parameters

- **pkey** – PKey to sign with
- **data** – data to be signed
- **digest** – message digest to use

Returns signature

New in version 0.11.

Python crypto.verify(cert, signature, data, digest)

Verify the signature for a data string.

Parameters

- **cert** – signing certificate (X509 object) corresponding to the private key which generated the signature.
- **signature** – signature returned by sign function
- **data** – data to be verified
- **digest** – message digest to use

Returns None if the signature is correct, raise exception otherwise.

New in version 0.11.

**X509 objects**

class OpenSSL.crypto.X509

An X.509 certificate.

**add_extensions(extensions)**

Add extensions to the certificate.

Parameters **extensions** (An iterable of X509Extension objects.) – The extensions to add.

Returns None

**digest(digest_name)**

Return the digest of the X509 object.

Parameters **digest_name**(bytes) – The name of the digest algorithm to use.

Returns The digest of the object, formatted as b":"-delimited hex pairs.

Return type bytes

classmethod from_cryptography(crypto_cert)

Construct based on a cryptography crypto_cert.

Parameters **crypto_key**(crypto�y.x509.Certificate) – A cryptography X.509 certificate.

Return type X509

New in version 17.1.0.

**get_extension(index)**

Get a specific extension of the certificate by index.

Extensions on a certificate are kept in order. The index parameter selects which extension will be returned.
Parameters **index** (*int*) – The index of the extension to retrieve.

**Returns** The extension at the specified index.

**Return type** `X509Extension`

**Raises** `IndexError` – If the extension index was out of bounds.

New in version 0.12.

**get_extension_count ()**

Get the number of extensions on this certificate.

**Returns** The number of extensions.

**Return type** `int`

New in version 0.12.

**get_issuer ()**

Return the issuer of this certificate.

This creates a new `X509Name` that wraps the underlying issuer name field on the certificate. Modifying it will modify the underlying certificate, and will have the effect of modifying any other `X509Name` that refers to this issuer.

**Returns** The issuer of this certificate.

**Return type** `X509Name`

**get_notAfter ()**

Get the timestamp at which the certificate stops being valid.

The timestamp is formatted as an ASN.1 TIME:

```
YYYYMMDDhhmmsZ
```

**Returns** A timestamp string, or `None` if there is none.

**Return type** `bytes` or `NoneType`

**get_notBefore ()**

Get the timestamp at which the certificate starts being valid.

The timestamp is formatted as an ASN.1 TIME:

```
YYYYMMDDhhmmsZ
```

**Returns** A timestamp string, or `None` if there is none.

**Return type** `bytes` or `NoneType`

**get_pubkey ()**

Get the public key of the certificate.

**Returns** The public key.

**Return type** `PKey`

**get_serial_number ()**

Return the serial number of this certificate.

**Returns** The serial number.
Return type int

get_signature_algorithm()
  Return the signature algorithm used in the certificate.

  Returns The name of the algorithm.

  Return type bytes

  Raises ValueError – If the signature algorithm is undefined.

New in version 0.13.

get_subject()
  Return the subject of this certificate.

  This creates a new X509Name that wraps the underlying subject name field on the certificate. Modifying it will modify the underlying certificate, and will have the effect of modifying any other X509Name that refers to this subject.

  Returns The subject of this certificate.

  Return type X509Name

get_version()
  Return the version number of the certificate.

  Returns The version number of the certificate.

  Return type int

gmtime_adj_notAfter(amount)
  Adjust the time stamp on which the certificate stops being valid.

  Parameters amount (int) – The number of seconds by which to adjust the timestamp.

  Returns None

gmtime_adj_notBefore(amount)
  Adjust the timestamp on which the certificate starts being valid.

  Parameters amount – The number of seconds by which to adjust the timestamp.

  Returns None

hasExpired()
  Check whether the certificate has expired.

  Returns True if the certificate has expired, False otherwise.

  Return type bool

set_issuer(issuer)
  Set the issuer of this certificate.

  Parameters issuer (X509Name) – The issuer.

  Returns None

set_notAfter(when)
  Set the timestamp at which the certificate stops being valid.

  The timestamp is formatted as an ASN.1 TIME:

  YYYYMDDhhmmssZ

  Parameters when (bytes) – A timestamp string.
Returns None

**set_notBefore** *(when)*
Set the timestamp at which the certificate starts being valid.
The timestamp is formatted as an ASN.1 TIME:

```
YYYYMMDDhhmmssZ
```

Parameters *when* *(bytes)* – A timestamp string.

Returns None

**set_pubkey** *(pkey)*
Set the public key of the certificate.

Parameters *pkey* *(PKey)* – The public key.

Returns None

**set_serial_number** *(serial)*
Set the serial number of the certificate.

Parameters *serial* *(int)* – The new serial number.

Returns :py:data:`None`

**set_subject** *(subject)*
Set the subject of this certificate.

Parameters *subject* *(X509Name)* – The subject.

Returns None

**set_version** *(version)*
Set the version number of the certificate. Note that the version value is zero-based, eg. a value of 0 is V1.

Parameters *version* *(int)* – The version number of the certificate.

Returns None

**sign** *(pkey, digest)*
Sign the certificate with this key and digest type.

Parameters

- *pkey* *(PKey)* – The key to sign with.
- *digest* *(bytes)* – The name of the message digest to use.

Returns None

**subject_name_hash** ()
Return the hash of the X509 subject.

Returns The hash of the subject.

Return type bytes

**to_cryptography** ()
Export as a cryptography certificate.

Return type cryptography.x509.Certificate

New in version 17.1.0.
X509Name objects

class OpenSSL.crypto.X509Name(name)
An X.509 Distinguished Name.

Variables

• countryName – The country of the entity.
• C – Alias for countryName.
• stateOrProvinceName – The state or province of the entity.
• ST – Alias for stateOrProvinceName.
• localityName – The locality of the entity.
• L – Alias for localityName.
• organizationName – The organization name of the entity.
• O – Alias for organizationName.
• organizationalUnitName – The organizational unit of the entity.
• OU – Alias for organizationalUnitName
• commonName – The common name of the entity.
• CN – Alias for commonName.
• emailAddress – The e-mail address of the entity.

__init__(name)
Create a new X509Name, copying the given X509Name instance.

Parameters name (X509Name) – The name to copy.

__setattr__(name, value)
x.__setattr__('name', value) <==> x.name = value

der()
Return the DER encoding of this name.

Returns The DER encoded form of this name.

Return type bytes

get_components()
Returns the components of this name, as a sequence of 2-tuples.

Returns The components of this name.

Return type list of name, value tuples.

hash()
Return an integer representation of the first four bytes of the MD5 digest of the DER representation of the name.

This is the Python equivalent of OpenSSL’s X509_NAME_hash.

Returns The (integer) hash of this name.

Return type int
X509Req objects

class OpenSSL.crypto.X509Req
An X.509 certificate signing requests.

__init__()
  x.__init__(...) initializes x; see help(type(x)) for signature

add_extensions(extensions)
  Add extensions to the certificate signing request.
  Parameters extensions (iterable of X509Extension) – The X.509 extensions to add.
  Returns None

classmethod from_cryptography(crypto_req)
  Construct based on a cryptography crypto_req.
  Parameters crypto_req (cryptography.x509.CertificateSigningRequest) – A cryptography X.509 certificate signing request
  Return type X509Req
  New in version 17.1.0.

get_extensions()
  Get X.509 extensions in the certificate signing request.
  Returns The X.509 extensions in this request.
  Return type list of X509Extension objects.
  New in version 0.15.

get_pubkey()
  Get the public key of the certificate signing request.
  Returns The public key.
  Return type PKey

get_subject()
  Return the subject of this certificate signing request.
  This creates a new X509Name that wraps the underlying subject name field on the certificate signing request. Modifying it will modify the underlying signing request, and will have the effect of modifying any other X509Name that refers to this subject.
  Returns The subject of this certificate signing request.
  Return type X509Name

get_version()
  Get the version subfield (RFC 2459, section 4.1.2.1) of the certificate request.
  Returns The value of the version subfield.
  Return type int

set_pubkey(pkey)
  Set the public key of the certificate signing request.
  Parameters pkey (PKey) – The public key to use.
  Returns None
**set_version (version)**

Set the version subfield (RFC 2459, section 4.1.2.1) of the certificate request.

- **Parameters:** `version (int)` – The version number.
- **Returns:** None

**sign (pkey, digest)**

Sign the certificate signing request with this key and digest type.

- **Parameters:**
  - `pkey (PKey)` – The key pair to sign with.
  - `digest (bytes)` – The name of the message digest to use for the signature, e.g. `b"sha256"`.
- **Returns:** None

**to_cryptography ()**

Export as a cryptography certificate signing request.

- **Return type:** `cryptography.x509.CertificateSigningRequest`
- **New in version 17.1.0.**

**verify (pkey)**

Verifies the signature on this certificate signing request.

- **Parameters:** `key (PKey)` – A public key.
- **Returns:** True if the signature is correct.
- **Return type:** `bool`
- **Raises:** `OpenSSL.crypto.Error` – If the signature is invalid or there is a problem verifying the signature.

---

**X509Store objects**

**class OpenSSL.crypto.X509Store**

An X.509 store.

An X.509 store is used to describe a context in which to verify a certificate. A description of a context may include a set of certificates to trust, a set of certificate revocation lists, verification flags and more.

An X.509 store, being only a description, cannot be used by itself to verify a certificate. To carry out the actual verification process, see `X509StoreContext`.

**add_cert (cert)**

Adds a trusted certificate to this store.

Adding a certificate with this method adds this certificate as a trusted certificate.

- **Parameters:** `cert (X509)` – The certificate to add to this store.
- **Raises:**
  - `TypeError` – If the certificate is not an `X509`.
  - `OpenSSL.crypto.Error` – If OpenSSL was unhappy with your certificate.
- **Returns:** None if the certificate was added successfully.
add_crl (crl)
Add a certificate revocation list to this store.

The certificate revocation lists added to a store will only be used if the associated flags are configured to
check certificate revocation lists.

New in version 16.1.0.

Parameters crl (CRL) – The certificate revocation list to add to this store.

Returns None if the certificate revocation list was added successfully.

set_flags (flags)
Set verification flags to this store.

Verification flags can be combined by oring them together.

Note: Setting a verification flag sometimes requires clients to add additional information to the store,
otherwise a suitable error will be raised.

For example, in setting flags to enable CRL checking a suitable CRL must be added to the store otherwise
an error will be raised.

New in version 16.1.0.

Parameters flags (int) – The verification flags to set on this store. See X509StoreFlags
for available constants.

Returns None if the verification flags were successfully set.

set_time (vfy_time)
Set the time against which the certificates are verified.

Normally the current time is used.

Note: For example, you can determine if a certificate was valid at a given time.

New in version 17.0.0.

Parameters vfy_time (datetime) – The verification time to set on this store.

Returns None if the verification time was successfully set.

X509StoreContextError objects

class OpenSSL.crypto.X509StoreContextError (message, certificate)
An exception raised when an error occurred while verifying a certificate using
OpenSSL.X509StoreContext.verify_certificate.

Variables certificate – The certificate which caused verify failure.

X509StoreContext objects

class OpenSSL.crypto.X509StoreContext (store, certificate)
An X.509 store context.

An X.509 store context is used to carry out the actual verification process of a certificate in a described context.
For describing such a context, see X509Store.
Variables

- **_store_ctx** – The underlying X509_STORE_CTX structure used by this instance. It is dynamically allocated and automatically garbage collected.

- **_store** – See the store __init__ parameter.

- **_cert** – See the certificate __init__ parameter.

Parameters

- **store** (**X509Store**) – The certificates which will be trusted for the purposes of any verifications.

- **certificate** (**X509**) – The certificate to be verified.

**set_store** (**store**)  
Set the context’s X.509 store.

New in version 0.15.

**Parameters**  
**store** (**X509Store**) – The store description which will be used for the purposes of any future verifications.

**verify_certificate** ()  
Verify a certificate in a context.

New in version 0.15.

**Raises** **X509StoreContextError** – If an error occurred when validating a certificate in the context. Sets certificate attribute to indicate which certificate caused the error.

**X509StoreFlags constants**

**class** **OpenSSL.crypto.X509StoreFlags**

Flags for X.509 verification, used to change the behavior of **X509Store**.

See OpenSSL Verification Flags for details.

- **CRL_CHECK**
- **CRL_CHECK_ALL**
- **IGNORE_CRITICAL**
- **X509_STRICT**
- **ALLOW_PROXY_CERTS**
- **POLICY_CHECK**
- **EXPlicit_POLICY**
- **INHIBIT_MAP**
- **NOTIFY_POLICY**
- **CHECK_SS_SIGNATURE**
- **CB_ISSUER_CHECK**
PKey objects

```python
class OpenSSL.crypto.PKey
    A class representing an DSA or RSA public key or key pair.

    bits()
        Returns the number of bits of the key
        Returns: The number of bits of the key.

    check()
        Check the consistency of an RSA private key.
        This is the Python equivalent of OpenSSL’s RSA_check_key.
        Returns: True if key is consistent.
        Raises:
        • `OpenSSL.crypto.Error` – if the key is inconsistent.
        • `TypeError` – if the key is of a type which cannot be checked. Only RSA keys can currently be checked.

classmethod from_cryptography_key(crypto_key)
    Construct based on a cryptography crypto_key.
    Parameters crypto_key (One of cryptography’s key interfaces.) – A cryptography key.
    Return type: PKey
    New in version 16.1.0.

generate_key(type, bits)
    Generate a key pair of the given type, with the given number of bits.
    This generates a key “into” the this object.
    Parameters:
    • type (TYPE_RSA or TYPE_DSA) – The key type.
    • bits (int >= 0) – The number of bits.
    Raises:
    • `TypeError` – If `type` or `bits` isn’t of the appropriate type.
    • `ValueError` – If the number of bits isn’t an integer of the appropriate size.
    Returns: None

to_cryptography_key()
    Export as a cryptography key.
    Return type: One of cryptography’s key interfaces.
    New in version 16.1.0.

type()
    Returns the type of the key
    Returns: The type of the key.
```

OpenSSL.crypto.TYPE_RSA
OpenSSL.crypto.TYPE_DSA
Key type constants.

PKCS7 objects
PKCS7 objects have the following methods:

class OpenSSL.crypto.PKCS7

    get_type_name()
    Returns the type name of the PKCS7 structure
    Returns A string with the typename

    type_is_data()
    Check if this NID_pkcs7_data object
    Returns True if the PKCS7 is of type data

    type_is_enveloped()
    Check if this NID_pkcs7_enveloped object
    Returns True if the PKCS7 is of type enveloped

    type_is_signed()
    Check if this NID_pkcs7_signed object
    Returns True if the PKCS7 is of type signed

    type_is_signedAndEnveloped()
    Check if this NID_pkcs7_signedAndEnveloped object
    Returns True if the PKCS7 is of type signedAndEnveloped

PKCS12 objects

class OpenSSL.crypto.PKCS12
A PKCS #12 archive.

    export (passphrase=None, iter=2048, maciter=1)
    Dump a PKCS12 object as a string.
    For more information, see the PKCS12_create() man page.
    Parameters
        • passphrase (bytes) – The passphrase used to encrypt the structure. Unlike some other passphrase arguments, this must be a string, not a callback.
        • iter (int) – Number of times to repeat the encryption step.
        • maciter (int) – Number of times to repeat the MAC step.
    Returns The string representation of the PKCS #12 structure.
    Return type

    get_ca_certificates()
    Get the CA certificates in the PKCS #12 structure.
    Returns A tuple with the CA certificates in the chain, or None if there are none.
    Return type tuple of X509 or None
get_certificate()
Get the certificate in the PKCS #12 structure.

    Returns The certificate, or None if there is none.
    Return type X509 or None

get_friendlyname()
Get the friendly name in the PKCS# 12 structure.

    Returns The friendly name, or None if there is none.
    Return type bytes or None

get_privatekey()
Get the private key in the PKCS #12 structure.

    Returns The private key, or None if there is none.
    Return type PKey

set_ca_certificates(cacerts)
Replace or set the CA certificates within the PKCS12 object.

    Parameters cacerts (An iterable of X509 or None) – The new CA certificates, or None to unset them.
    Returns None

set_certificate(cert)
Set the certificate in the PKCS #12 structure.

    Parameters cert (X509 or None) – The new certificate, or None to unset it.
    Returns None

set_friendlyname(name)
Set the friendly name in the PKCS #12 structure.

    Parameters name (bytes or None) – The new friendly name, or None to unset.
    Returns None

set_privatekey(pkey)
Set the certificate portion of the PKCS #12 structure.

    Parameters pkey (PKey or None) – The new private key, or None to unset it.
    Returns None

X509Extension objects

class OpenSSL.crypto.X509Extension(type_name, critical, value, subject=None, issuer=None)
An X.509 v3 certificate extension.

    __init__(type_name, critical, value, subject=None, issuer=None)
    Initializes an X509 extension.

    Parameters
        • type_name (bytes) – The name of the type of extension to create.
        • critical (bool) – A flag indicating whether this is a critical extension.
        • value (bytes) – The value of the extension.
• **subject** *(X509)* – Optional X509 certificate to use as subject.

• **issuer** *(X509)* – Optional X509 certificate to use as issuer.

```python
__str__()

Returns a nice text representation of the extension
```

```python
get_critical()

Returns the critical field of this X.509 extension.

Returns The critical field.
```

```python
get_data()

Returns the data of the X509 extension, encoded as ASN.1.

Returns The ASN.1 encoded data of this X509 extension.

Return type bytes

New in version 0.12.
```

```python
get_short_name()

Returns the short type name of this X.509 extension.

The result is a byte string such as `b"basicConstraints"`.

Returns The short type name.

Return type bytes

New in version 0.12.
```

### NetscapeSPKI objects

```python
class OpenSSL.crypto.NetscapeSPKI

A Netscape SPKI object.
```

```python
__init__()

x.__init__(...) initializes x; see help(type(x)) for signature
```

```python
b64_encode()

Generate a base64 encoded representation of this SPKI object.

Returns The base64 encoded string.

Return type bytes
```

```python
get_pubkey()

Get the public key of this certificate.

Returns The public key.

Return type PKey
```

```python
set_pubkey(pkey)

Set the public key of the certificate

Parameters pkey – The public key

Returns None
```

```python
sign(pkey, digest)

Sign the certificate request with this key and digest type.

Parameters
```
• **pkey** (*PKey*) – The private key to sign with.

• **digest** (*bytes*) – The message digest to use.

Returns None

**verify(key)**  
Verifies a signature on a certificate request.

Parameters **key** (*PKey*) – The public key that signature is supposedly from.

Returns **True** if the signature is correct.

Return type **bool**

Raises **OpenSSL.crypto.Error** – If the signature is invalid, or there was a problem verifying the signature.

### CRL objects

**class** OpenSSL.crypto.CRL  
A certificate revocation list.

**__init__()**  
x.__init__(...) initializes x; see help(type(x)) for signature

**add_revoked(revoked)**  
Add a revoked (by value not reference) to the CRL structure

This revocation will be added by value, not by reference. That means it’s okay to mutate it after adding: it won’t affect this CRL.

Parameters **revoked** (*Revoked*) – The new revocation.

Returns None

**export(cert, key, type=1, days=100, digest=<object object>)**  
Export the CRL as a string.

Parameters

• **cert** (*X509*) – The certificate used to sign the CRL.

• **key** (*PKey*) – The key used to sign the CRL.

• **type** (*int*) – The export format, either **FILETYPE_PEM**, **FILETYPE_ASN1**, or **FILETYPE_TEXT**.

• **days** (*int*) – The number of days until the next update of this CRL.

• **digest** (*bytes*) – The name of the message digest to use (eg b"sha2566").

Return type **bytes**

**classmethod from_cryptography(crypto_crl)**  
Construct based on a cryptography **crypto_crl**.

Parameters **crypto_crl** (cryptography.x509.CertificateRevocationList) – A cryptography certificate revocation list

Return type **CRL**

New in version 17.1.0.
get_issuer()
Get the CRL’s issuer.
New in version 16.1.0.

Return type: X509Name

get_revoked()
Return the revocations in this certificate revocation list.
These revocations will be provided by value, not by reference. That means it’s okay to mutate them: it
won’t affect this CRL.

Returns: The revocations in this CRL.

Return type: tuple of Revocation

set_lastUpdate(when)
Set when the CRL was last updated.
The timestamp is formatted as an ASN.1 TIME:

YYYYMMDDhhmmssZ

New in version 16.1.0.

Parameters: when (bytes) – A timestamp string.

Returns: None

set_nextUpdate(when)
Set when the CRL will next be updated.
The timestamp is formatted as an ASN.1 TIME:

YYYYMMDDhhmmssZ

New in version 16.1.0.

Parameters: when (bytes) – A timestamp string.

Returns: None

set_version(version)
Set the CRL version.
New in version 16.1.0.

Parameters: version (int) – The version of the CRL.

Returns: None

sign(issuer_cert, issuer_key, digest)
Sign the CRL.
Signing a CRL enables clients to associate the CRL itself with an issuer. Before a CRL is meaningful to
other OpenSSL functions, it must be signed by an issuer.
This method implicitly sets the issuer’s name based on the issuer certificate and private key used to sign
the CRL.

New in version 16.1.0.

Parameters:
* issuer_cert (X509) – The issuer’s certificate.
• **issuer_key** (PKey) – The issuer’s private key.
• **digest (bytes)** – The digest method to sign the CRL with.

**to_cryptography()**
Export as a cryptography CRL.

Return type **cryptography.x509.CertificateRevocationList**
New in version 17.1.0.

## Revoked objects

**class OpenSSL.crypto.Revoked**
A certificate revocation.

**all_reasons()**
Return a list of all the supported reason strings.
This list is a copy; modifying it does not change the supported reason strings.

Returns **A list of reason strings.**

Return type **list of bytes**

**get_reason()**
Get the reason of this revocation.

Returns **The reason, or None if there is none.**

Return type **bytes or NoneType**

See also:

**all_reasons()**, which gives you a list of all supported reasons this method might return.

**get_rev_date()**
Get the revocation timestamp.

Returns **The timestamp of the revocation, as ASN.1 TIME.**

Return type **bytes**

**get_serial()**
Get the serial number.
The serial number is formatted as a hexadecimal number encoded in ASCII.

Returns **The serial number.**

Return type **bytes**

**set_reason (reason)**
Set the reason of this revocation.
If `reason` is `None`, delete the reason instead.

Parameters **reason (bytes or NoneType)** – The reason string.

Returns **None**

See also:

**all_reasons()**, which gives you a list of all supported reasons which you might pass to this method.

**set_rev_date (when)**
Set the revocation timestamp.
Parameters  when (bytes) – The timestamp of the revocation, as ASN.1 TIME.

Returns  None

set_serial (hex_str)
Set the serial number.

The serial number is formatted as a hexadecimal number encoded in ASCII.

Parameters  hex_str (bytes) – The new serial number.

Returns  None

Exceptions

exception  OpenSSL.crypto.Error
Generic exception used in the crypto module.

Digest names

Several of the functions and methods in this module take a digest name. These must be strings describing a digest algorithm supported by OpenSSL (by EVP_get_digestbyname, specifically). For example, b"sha256" or b"sha384".

More information and a list of these digest names can be found in the EVP_DigestInit(3) man page of your OpenSSL installation. This page can be found online for the latest version of OpenSSL: https://www.openssl.org/docs/manmaster/man3/EVP_DigestInit.html

1.3.2 SSL — An interface to the SSL-specific parts of OpenSSL

This module handles things specific to SSL. There are two objects defined: Context, Connection.

OpenSSL.SSL.SSLv2_METHOD
OpenSSL.SSL.SSLv3_METHOD
OpenSSL.SSL.TLSv1_METHOD
OpenSSL.SSL.TLSv1_1_METHOD
OpenSSL.SSL.TLSv1_2_METHOD

These constants represent the different SSL methods to use when creating a context object. If the underlying OpenSSL build is missing support for any of these protocols, constructing a Context using the corresponding *_METHOD will raise an exception.

OpenSSL.SSL VERIFY_NONE
OpenSSL.SSL VERIFY_PEER
OpenSSL.SSL VERIFY_FAIL_IF_NO_PEER_CERT

These constants represent the verification mode used by the Context object’s set_verify() method.

OpenSSL.SSL.FILETYPE_PEM
OpenSSL.SSL.FILETYPE_ASN1

File type constants used with the use_certificate_file() and use_privatekey_file() methods of Context objects.

OpenSSL.SSL.OP_SINGLE_DH_USE
OpenSSL.SSL.OP_SINGLE_ECDH_USE

Constants used with set_options() of Context objects.
When these options are used, a new key will always be created when using ephemeral (Elliptic curve) Diffie-Hellman.

```
OpenSSL.SSL.OP_EPHEMERAL_RSA
```

Constant used with `set_options()` of Context objects.

When this option is used, ephemeral RSA keys will always be used when doing RSA operations.

```
OpenSSL.SSL.OP_NO_TICKET
```

Constant used with `set_options()` of Context objects.

When this option is used, the session ticket extension will not be used.

```
OpenSSL.SSL.OP_NO_COMPRESSION
```

Constant used with `set_options()` of Context objects.

When this option is used, compression will not be used.

```
OpenSSL.SSL.OP_NO_SSLv2
OpenSSL.SSL.OP_NO_SSLv3
OpenSSL.SSL.OP_NO_TLSv1
OpenSSL.SSL.OP_NO_TLSv1_1
OpenSSL.SSL.OP_NO_TLSv1_2
```

Constants used with `set_options()` of Context objects.

Each of these options disables one version of the SSL/TLS protocol. This is interesting if you’re using e.g. `SSLv23_METHOD` to get an SSLv2-compatible handshake, but don’t want to use SSLv2. If the underlying OpenSSL build is missing support for any of these protocols, the `OP_NO_*` constant may be undefined.

```
OpenSSL.SSL.SSLEAY_VERSION
OpenSSL.SSL.SSLEAY_CFLAGS
OpenSSL.SSL.SSLEAY_BUILT_ON
OpenSSL.SSL.SSLEAY_PLATFORM
OpenSSL.SSL.SSLEAY_DIR
```

Constants used with `SSLeay_version()` to specify what OpenSSL version information to retrieve. See the man page for the `SSLeay_version()` C API for details.

```
OpenSSL.SSL.SESSION_CACHE_OFF
OpenSSL.SSL.SESSION_CACHE_CLIENT
OpenSSL.SSL.SESSION_CACHE_SERVER
OpenSSL.SSL.SESSION_CACHE_BOTH
OpenSSL.SSL.SESSION_CACHE_NO_AUTO_CLEAR
OpenSSL.SSL.SESSION_CACHE_NO_INTERNAL_LOOKUP
OpenSSL.SSL.SESSION_CACHE_NO_INTERNAL_STORE
OpenSSL.SSL.SESSION_CACHE_NO_INTERNAL
```

Constants used with `Context.set_session_cache_mode()` to specify the behavior of the session cache and potential session reuse. See the man page for the `SSL_CTX_set_session_cache_mode()` C API for details.

New in version 0.14.

```
OpenSSL.SSL.OPENSSL_VERSION_NUMBER
```

An integer giving the version number of the OpenSSL library used to build this version of pyOpenSSL. See the man page for the `SSLeay_version()` C API for details.

```
OpenSSL.SSL.SSLeay_version(type)
```

Return a string describing the version of OpenSSL in use.

**Parameters** `type` — One of the `SSLEAY_` constants defined in this module.
OpenSSL.SSL.ContextType
See Context.

class OpenSSL.SSL.Context (method)
OpenSSL.SSL.Context instances define the parameters for setting up new SSL connections.

Parameters method — One of SSLv2_METHOD, SSLv3_METHOD, SSLv23_METHOD, or TLSv1_METHOD.

class OpenSSL.SSL.Session
A class representing an SSL session. A session defines certain connection parameters which may be re-used to speed up the setup of subsequent connections.

New in version 0.14.

OpenSSL.SSL.ConnectionType
See Connection.

class OpenSSL.SSL.Connection (context, socket)
A class representing SSL connections.

context should be an instance of Context and socket should be a socket\(^1\) object. socket may be None; in this case, the Connection is created with a memory BIO: see the bio_read(), bio_write(), and bio_shutdown() methods.

exception OpenSSL.SSL.Error
This exception is used as a base class for the other SSL-related exceptions, but may also be raised directly.

Whenever this exception is raised directly, it has a list of error messages from the OpenSSL error queue, where each item is a tuple (lib, function, reason). Here lib, function and reason are all strings, describing where and what the problem is. See err(3) for more information.

exception OpenSSL.SSL.ZeroReturnError
This exception matches the error return code SSL_ERROR_ZERO_RETURN, and is raised when the SSL Connection has been closed. In SSL 3.0 and TLS 1.0, this only occurs if a closure alert has occurred in the protocol, i.e. the connection has been closed cleanly. Note that this does not necessarily mean that the transport layer (e.g. a socket) has been closed.

It may seem a little strange that this is an exception, but it does match an SSL_ERROR code, and is very convenient.

exception OpenSSL.SSL.WantReadError
The operation did not complete; the same I/O method should be called again later, with the same arguments. Any I/O method can lead to this since new handshakes can happen at any time.

The wanted read is for dirty data sent over the network, not the clean data inside the tunnel. For a socket based SSL connection, read means data coming at us over the network. Until that read succeeds, the attempted OpenSSL.SSL.Connection.recv(), OpenSSL.SSL.Connection.send(), or OpenSSL.SSL.Connection.do_handshake() is prevented or incomplete. You probably want to select() on the socket before trying again.

exception OpenSSL.SSL.WantWriteError
See WantReadError. The socket send buffer may be too full to write more data.

exception OpenSSL.SSL.WantX509LookupError
The operation did not complete because an application callback has asked to be called again. The I/O method should be called again later, with the same arguments.

\(^1\) Actually, all that is required is an object that behaves like a socket, you could even use files, even though it’d be tricky to get the handshakes right!
exception OpenSSL.SSL.SysCallError

The SysCallError occurs when there’s an I/O error and OpenSSL’s error queue does not contain any information. This can mean two things: An error in the transport protocol, or an end of file that violates the protocol. The parameter to the exception is always a pair (errnum, errstr).

Context objects

Context objects have the following methods:

class OpenSSL.SSL.Context (method)

OpenSSL.SSL.Context instances define the parameters for setting up new SSL connections.

Parameters method – One of SSLv2_METHOD, SSLv3_METHOD, SSLv23_METHOD, or TLSv1_METHOD.

add_client_ca (certificate_authority)

Add the CA certificate to the list of preferred signers for this context.

The list of certificate authorities will be sent to the client when the server requests a client certificate.

Parameters certificate_authority – certificate authority’s X509 certificate.

Returns None

New in version 0.10.

add_extra_chain_cert (certobj)

Add certificate to chain

Parameters certobj – The X509 certificate object to add to the chain

Returns None

check_privatekey ()

Check if the private key (loaded with use_privatekey()) matches the certificate (loaded with use_certificate())

Returns None (raises Error if something’s wrong)

get_app_data ()

Get the application data (supplied via set_app_data())

Returns The application data

get_cert_store ()

Get the certificate store for the context. This can be used to add “trusted” certificates without using the load_verify_locations() method.

Returns A X509Store object or None if it does not have one.

get_session_cache_mode ()

Get the current session cache mode.

Returns The currently used cache mode.

New in version 0.14.

get_timeout ()

Retrieve session timeout, as set by set_timeout(). The default is 300 seconds.
Returns The session timeout

get_verify_depth()
Retrieve the Context object’s verify depth, as set by set_verify_depth().

Returns The verify depth

get_verify_mode()
Retrieve the Context object’s verify mode, as set by set_verify().

Returns The verify mode

load_client_ca(cafile)
Load the trusted certificates that will be sent to the client. Does not actually imply any of the certificates are trusted; that must be configured separately.

Parameters cafile (bytes) – The path to a certificates file in PEM format.

Returns None

load_tmp_dh(dhfile)
Load parameters for Ephemeral Diffie-Hellman

Parameters dhfile – The file to load EDH parameters from (bytes or unicode).

Returns None

load_verify_locations(cafile, capath=None)
Let SSL know where we can find trusted certificates for the certificate chain. Note that the certificates have to be in PEM format.

If capath is passed, it must be a directory prepared using the c_rehash tool included with OpenSSL. Either, but not both, of pemfile or capath may be None.

Parameters
  • cafile – In which file we can find the certificates (bytes or unicode).
  • capath – In which directory we can find the certificates (bytes or unicode).

Returns None

set_alpn_protos(protos)
Specify the protocols that the client is prepared to speak after the TLS connection has been negotiated using Application Layer Protocol Negotiation.

Parameters protos – A list of the protocols to be offered to the server. This list should be a Python list of bytestrings representing the protocols to offer, e.g. [b'http/1.1', b'spdy/2'].

set_alpn_select_callback(callback)
Specify a callback function that will be called on the server when a client offers protocols using ALPN.

Parameters callback – The callback function. It will be invoked with two arguments: the Connection, and a list of offered protocols as bytestrings, e.g [b'http/1.1', b'spdy/2']. It should return one of those bytestrings, the chosen protocol.

set_app_data(data)
Set the application data (will be returned from get_app_data())

Parameters data – Any Python object

Returns None
**set_cipher_list** *(cipher_list)*
Set the list of ciphers to be used in this context.

See the OpenSSL manual for more information (e.g. `ciphers(1)`).

**Parameters** `cipher_list` *(bytes)* – An OpenSSL cipher string.

**Returns** None

**set_client_ca_list** *(certificateAuthorities)*
Set the list of preferred client certificate signers for this server context.

This list of certificate authorities will be sent to the client when the server requests a client certificate.

**Parameters** `certificateAuthorities` – a sequence of X509Names.

**Returns** None

New in version 0.10.

**set_default_verify_paths** ()
Specify that the platform provided CA certificates are to be used for verification purposes. This method has some caveats related to the binary wheels that cryptography (pyOpenSSL's primary dependency) ships:

- macOS will only load certificates using this method if the user has the openssl@1.1 Homebrew formula installed in the default location.
- Windows will not work.
- manylinux1 cryptography wheels will work on most common Linux distributions in pyOpenSSL 17.1.0 and above. pyOpenSSL detects the manylinux1 wheel and attempts to load roots via a fallback path.

**Returns** None

**set_info_callback** *(callback)*
Set the information callback to `callback`. This function will be called from time to time during SSL handshakes.

**Parameters** `callback` – The Python callback to use. This should take three arguments: a Connection object and two integers. The first integer specifies where in the SSL handshake the function was called, and the other the return code from a (possibly failed) internal function call.

**Returns** None

**set_mode** *(mode)*
Add modes via bitmask. Modes set before are not cleared! This method should be used with the `MODE_*` constants.

**Parameters** mode – The mode to add.

**Returns** The new mode bitmask.

**set_npn_advertise_callback** *(callback)*
Specify a callback function that will be called when offering Next Protocol Negotiation as a server.

**Parameters** `callback` – The callback function. It will be invoked with one argument, the `Connection` instance. It should return a list of bytestrings representing the advertised protocols, like `[b'http/1.1', b'spdy/2']`.

New in version 0.15.
set_npn_select_callback(callback)
Specify a callback function that will be called when a server offers Next Protocol Negotiation options.

Parameters

- **callback** – The callback function. It will be invoked with two arguments: the Connection, and a list of offered protocols as bytestrings, e.g. [b'http/1.1', b'spdy/2']. It should return one of those bytestrings, the chosen protocol.

New in version 0.15.

set_ocsp_client_callback(callback, data=None)
Set a callback to validate OCSP data stapled to the TLS handshake on the client side.

Parameters

- **callback** – The callback function. It will be invoked with three arguments: the Connection, a bytestring containing the stapled OCSP assertion, and the optional arbitrary data you have provided. The callback must return a boolean that indicates the result of validating the OCSP data: True if the OCSP data is valid and the certificate can be trusted, or False if either the OCSP data is invalid or the certificate has been revoked.

- **data** – Some opaque data that will be passed into the callback function when called. This can be used to avoid needing to do complex data lookups or to keep track of what context is being used. This parameter is optional.

set_ocsp_server_callback(callback, data=None)
Set a callback to provide OCSP data to be stapled to the TLS handshake on the server side.

Parameters

- **callback** – The callback function. It will be invoked with two arguments: the Connection, and the optional arbitrary data you have provided. The callback must return a bytestring that contains the OCSP data to staple to the handshake. If no OCSP data is available for this connection, return the empty bytestring.

- **data** – Some opaque data that will be passed into the callback function when called. This can be used to avoid needing to do complex data lookups or to keep track of what context is being used. This parameter is optional.

set_options(options)
Add options. Options set before are not cleared! This method should be used with the OP_* constants.

Parameters

- **options** – The options to add.

Returns

The new option bitmask.

set_passwd_cb(callback, userdata=None)
Set the passphrase callback. This function will be called when a private key with a passphrase is loaded.

Parameters

- **callback** – The Python callback to use. This must accept three positional arguments. First, an integer giving the maximum length of the passphrase it may return. If the returned passphrase is longer than this, it will be truncated. Second, a boolean value which will be true if the user should be prompted for the passphrase twice and the callback should verify that the two values supplied are equal. Third, the value given as theuserdata parameter to set_passwd_cb(). The callback must return a byte string. If an error occurs, callback should return a false value (e.g. an empty string).

- **userdata** – (optional) A Python object which will be given as argument to the callback

Returns

None
**set_session_cache_mode** *(mode)*

Set the behavior of the session cache used by all connections using this Context. The previously set mode is returned. See `SESS_CACHE_*` for details about particular modes.

**Parameters**

- **mode** – One or more of the `SESS_CACHE_*` flags (combine using bitwise or)

**Returns**

The previously set caching mode.

New in version 0.14.

**set_session_id** *(buf)*

Set the session id to `buf` within which a session can be reused for this Context object. This is needed when doing session resumption, because there is no way for a stored session to know which Context object it is associated with.

**Parameters**

- **buf** (*bytes*) – The session id.

**Returns**

None

**set_timeout** *(timeout)*

Set the timeout for newly created sessions for this Context object to `timeout`. The default value is 300 seconds. See the OpenSSL manual for more information (e.g. `SSL_CTX_set_timeout(3)`).

**Parameters**

- **timeout** – The timeout in (whole) seconds

**Returns**

The previous session timeout

**set_tlsext_servername_callback** *(callback)*

Specify a callback function to be called when clients specify a server name.

**Parameters**

- **callback** – The callback function. It will be invoked with one argument, the Connection instance.

New in version 0.13.

**set_tlsext_use_srtp** *(profiles)*

Enable support for negotiating SRTP keying material.

**Parameters**

- **profiles** (*bytes*) – A colon delimited list of protection profile names, like `b"SRTP_AES128_CM_SHA1_80:SRTP_AES128_CM_SHA1_32"`.

**Returns**

None

**set_tmp_ecdh** *(curve)*

Select a curve to use for ECDHE key exchange.

**Parameters**

- **curve** – A curve object to use as returned by either `OpenSSL.crypto.get_elliptic_curve()` or `OpenSSL.crypto.get_elliptic_curves()`.

**Returns**

None

**set_verify** *(mode, callback)*

Set the verification flags for this Context object to `mode` and specify that `callback` should be used for verification callbacks.

**Parameters**

- **mode** – The verify mode, this should be one of `VERIFY_NONE` and `VERIFY_PEER`. If `VERIFY_PEER` is used, `mode` can be OR:ed with `VERIFY_FAIL_IF_NO_PEER_CERT` and `VERIFY_CLIENT_ONCE` to further control the behaviour.

- **callback** – The Python callback to use. This should take five arguments: A Connection object, an X509 object, and three integer variables, which are in turn potential error
number, error depth and return code. callback should return True if verification passes and False otherwise.

**Returns** None

See SSL_CTX_set_verify(3SSL) for further details.

**set_verify_depth**(depth)

Set the maximum depth for the certificate chain verification that shall be allowed for this Context object.

**Parameters** depth – An integer specifying the verify depth

**Returns** None

**use_certificate**(cert)

Load a certificate from a X509 object

**Parameters** cert – The X509 object

**Returns** None

**use_certificate_chain_file**(certfile)

Load a certificate chain from a file.

**Parameters** certfile – The name of the certificate chain file (bytes or unicode). Must be PEM encoded.

**Returns** None

**use_certificate_file**(certfile, filetype=1)

Load a certificate from a file

**Parameters**

• certfile – The name of the certificate file (bytes or unicode).

• filetype – (optional) The encoding of the file, which is either FILETYPE_PEM or FILETYPE_ASN1. The default is FILETYPE_PEM.

**Returns** None

**use_privatekey**(pkey)

Load a private key from a PKey object

**Parameters** pkey – The PKey object

**Returns** None

**use_privatekey_file**(keyfile, filetype=<object object>)

Load a private key from a file

**Parameters**

• keyfile – The name of the key file (bytes or unicode)

• filetype – (optional) The encoding of the file, which is either FILETYPE_PEM or FILETYPE_ASN1. The default is FILETYPE_PEM.

**Returns** None

**Session objects**

Session objects have no methods.
Connection objects

Connection objects have the following methods:

```python
class OpenSSL.SSL.Connection(context, socket=None)
```

- **accept()**
  Call the `accept()` method of the underlying socket and set up SSL on the returned socket, using the Context object supplied to this `Connection` object at creation.
  
  **Returns** A `(conn, addr)` pair where `conn` is the new `Connection` object created, and `address` is as returned by the socket’s `accept()`.

- **bio_read(bufsiz)**
  If the Connection was created with a memory BIO, this method can be used to read bytes from the write end of that memory BIO. Many Connection methods will add bytes which must be read in this manner or the buffer will eventually fill up and the Connection will be able to take no further actions.
  
  **Parameters** `bufsiz` – The maximum number of bytes to read
  
  **Returns** The string read.

- **bio_shutdown()**
  If the Connection was created with a memory BIO, this method can be used to indicate that end of file has been reached on the read end of that memory BIO.
  
  **Returns** None

- **bio_write(buf)**
  If the Connection was created with a memory BIO, this method can be used to add bytes to the read end of that memory BIO. The Connection can then read the bytes (for example, in response to a call to `recv()`).
  
  **Parameters** `buf` – The string to put into the memory BIO.
  
  **Returns** The number of bytes written

- **client_random()**
  Retrieve the random value used with the client hello message.
  
  **Returns** A string representing the state

- **connect(addr)**
  Call the `connect()` method of the underlying socket and set up SSL on the socket, using the Context object supplied to this `Connection` object at creation.
  
  **Parameters** `addr` – A remote address
  
  **Returns** What the socket’s connect method returns

- **connect_ex(addr)**
  Call the `connect_ex()` method of the underlying socket and set up SSL on the socket, using the Context object supplied to this `Connection` object at creation. Note that if the `connect_ex()` method of the socket doesn’t return 0, SSL won’t be initialized.
  
  **Parameters** `addr` – A remote address
  
  **Returns** What the socket’s connect_ex method returns

- **do_handshake()**
  Perform an SSL handshake (usually called after `renegotiate()` or one of `set_accept_state()` or `set_accept_state()`). This can raise the same exceptions as `send()` and `recv()`.
  
  **Returns** None.
**export_keying_material** *(label, olen, context=None)*

Obtain keying material for application use.

- **Param** `label` - a disambiguation label string as described in RFC 5705
- **Param** `olen` - the length of the exported key material in bytes
- **Param** `context` - a per-association context value

**Returns** the exported key material bytes or None

**get_alpn_proto_negotiated** ()

Get the protocol that was negotiated by ALPN.

**Returns** A bytestring of the protocol name. If no protocol has been negotiated yet, returns an empty string.

**get_app_data** ()

Retrieve application data as set by `set_app_data()`.

**Returns** The application data

**get_certificate** ()

Retrieve the local certificate (if any)

**Returns** The local certificate

**get_cipher_bits** ()

Obtain the number of secret bits of the currently used cipher.

**Returns** The number of secret bits of the currently used cipher or *None* if no connection has been established.

**Return type** `int` or NoneType

New in version 0.15.

**get_cipher_list** ()

Retrieve the list of ciphers used by the Connection object.

**Returns** A list of native cipher strings.

**get_cipher_name** ()

Obtain the name of the currently used cipher.

**Returns** The name of the currently used cipher or *None* if no connection has been established.

**Return type** unicode or NoneType

New in version 0.15.

**get_cipher_version** ()

Obtain the protocol version of the currently used cipher.

**Returns** The protocol name of the currently used cipher or *None* if no connection has been established.

**Return type** unicode or NoneType

New in version 0.15.

**get_client_ca_list** ()

Get CAs whose certificates are suggested for client authentication.

**Returns**
If this is a server connection, the list of certificate authorities that will be sent or has been sent to the client, as controlled by this `Connection`'s `Context`.

If this is a client connection, the list will be empty until the connection with the server is established.

New in version 0.10.

`get_context()`
Retrieve the `Context` object associated with this `Connection`.

`get_finished()`
Obtain the latest TLS Finished message that we sent.

    Returns  The contents of the message or `None` if the TLS handshake has not yet completed.

    Return type  `bytes` or `NoneType`

New in version 0.15.

`get_next_proto_negotiated()`
Get the protocol that was negotiated by NPN.

    Returns  A bytestring of the protocol name. If no protocol has been negotiated yet, returns an empty string.

New in version 0.15.

`get_peer_cert_chain()`
Retrieve the other side’s certificate (if any)

    Returns  A list of X509 instances giving the peer’s certificate chain, or `None` if it does not have one.

`get_peer_certificate()`
Retrieve the other side’s certificate (if any)

    Returns  The peer’s certificate

`get_peer_finished()`
Obtain the latest TLS Finished message that we received from the peer.

    Returns  The contents of the message or `None` if the TLS handshake has not yet completed.

    Return type  `bytes` or `NoneType`

New in version 0.15.

`get_protocol_version()`
Retrieve the SSL or TLS protocol version of the current connection.

    Returns  The TLS version of the current connection. For example, it will return `0x769` for connections made over TLS version 1.

    Return type  `int`

`get_protocol_version_name()`
Retrieve the protocol version of the current connection.

    Returns  The TLS version of the current connection, for example the value for TLS 1.2 would be `TLSv1.2` or `Unknown` for connections that were not successfully established.

    Return type  `unicode`
get_servername()
Retrieve the servername extension value if provided in the client hello message, or None if there wasn’t
one.

Returns: A byte string giving the server name or None.
New in version 0.13.

get_session()
Returns the Session currently used.

Returns: An instance of OpenSSL.SSL.Session or None if no session exists.
New in version 0.14.

get_shutdown()
Get the shutdown state of the Connection.

Returns: The shutdown state, a bitvector of SENT_SHUTDOWN, RECEIVED_SHUTDOWN.

get_state_string()
Retrieve a verbose string detailing the state of the Connection.

Returns: A string representing the state
Return type: bytes

makefile(*args, **kwargs)
The makefile() method is not implemented, since there is no dup semantics for SSL connections
Raise: NotImplementedError

master_key()
Retrieve the value of the master key for this session.

Returns: A string representing the state

pending()
Get the number of bytes that can be safely read from the SSL buffer (not the underlying transport buffer).

Returns: The number of bytes available in the receive buffer.

read(bufsiz, flags=None)
Receive data on the connection.

Parameters:
- bufsiz – The maximum number of bytes to read
- flags – (optional) The only supported flag is MSG_PEEK, all other flags are ignored.

Returns: The string read from the Connection

recv(bufsiz, flags=None)
Receive data on the connection.

Parameters:
- bufsiz – The maximum number of bytes to read
- flags – (optional) The only supported flag is MSG_PEEK, all other flags are ignored.

Returns: The string read from the Connection

recv_into(buffer, nbytes=None, flags=None)
Receive data on the connection and copy it directly into the provided buffer, rather than creating a new string.
Parameters

• **buffer** – The buffer to copy into.

• **nbytes** – (optional) The maximum number of bytes to read into the buffer. If not present, defaults to the size of the buffer. If larger than the size of the buffer, is reduced to the size of the buffer.

• **flags** – (optional) The only supported flag is MSG_PEEK, all other flags are ignored.

Returns  The number of bytes read into the buffer.

**renegotiate()**

Renegotiate the session.

Returns  True if the renegotiation can be started, False otherwise

Return type  bool

**renegotiate_pending()**

Check if there’s a renegotiation in progress, it will return False once a renegotiation is finished.

Returns  Whether there’s a renegotiation in progress

Return type  bool

**request_ocsp()**

Called to request that the server sends stapled OCSP data, if available. If this is not called on the client side then the server will not send OCSP data. Should be used in conjunction with Context.set_ocsp_client_callback().

**send(buf, flags=0)**

Send data on the connection. NOTE: If you get one of the WantRead, WantWrite or WantX509Lookup exceptions on this, you have to call the method again with the SAME buffer.

Parameters

• **buf** – The string, buffer or memoryview to send

• **flags** – (optional) Included for compatibility with the socket API, the value is ignored

Returns  The number of bytes written

**sendall(buf, flags=0)**

Send “all” data on the connection. This calls send() repeatedly until all data is sent. If an error occurs, it’s impossible to tell how much data has been sent.

Parameters

• **buf** – The string, buffer or memoryview to send

• **flags** – (optional) Included for compatibility with the socket API, the value is ignored

Returns  The number of bytes written

**server_random()**

Retrieve the random value used with the server hello message.

Returns  A string representing the state

**set_accept_state()**

Set the connection to work in server mode. The handshake will be handled automatically by read/write.

Returns  None
**set_alpn_protos** (*protos*)

Specify the client's ALPN protocol list.

These protocols are offered to the server during protocol negotiation.

**Parameters protos** – A list of the protocols to be offered to the server. This list should be a Python list of bytestrings representing the protocols to offer, e.g. `[b'http/1.1', b'spdy/2']`.

**set_app_data** (*data*)

Set application data

**Parameters data** – The application data

**Returns** None

**set_connect_state** ()

Set the connection to work in client mode. The handshake will be handled automatically by read/write.

**Returns** None

**set_context** (*context*)

Switch this connection to a new session context.

**Parameters context** – A `Context` instance giving the new session context to use.

**set_session** (*session*)

Set the session to be used when the TLS/SSL connection is established.

**Parameters session** – A Session instance representing the session to use.

**Returns** None

New in version 0.14.

**set_shutdown** (*state*)

Set the shutdown state of the Connection.

**Parameters state** – bitvector of SENT_SHUTDOWN, RECEIVED_SHUTDOWN.

**Returns** None

**set_tlsext_host_name** (*name*)

Set the value of the servername extension to send in the client hello.

**Parameters name** – A byte string giving the name.

New in version 0.13.

**shutdown** ()

Send the shutdown message to the Connection.

**Returns** True if the shutdown completed successfully (i.e. both sides have sent closure alerts), False otherwise (in which case you call `recv()` or `send()` when the connection becomes readable/writeable).

**sock_shutdown** (*args, **kwargs*)

Call the `shutdown()` method of the underlying socket. See `shutdown(2)`.

**Returns** What the socket's `shutdown()` method returns

**total_renegotiations** ()

Find out the total number of renegotiations.

**Returns** The number of renegotiations.

**Return type** int
want_read()
    Checks if more data has to be read from the transport layer to complete an operation.

    Returns True iff more data has to be read

want_write()
    Checks if there is data to write to the transport layer to complete an operation.

    Returns True iff there is data to write

write(buf, flags=0)
    Send data on the connection. NOTE: If you get one of the WantRead, WantWrite or WantX509Lookup exceptions on this, you have to call the method again with the SAME buffer.

    Parameters
      • buf – The string, buffer or memoryview to send
      • flags – (optional) Included for compatibility with the socket API, the value is ignored

    Returns The number of bytes written

1.4 Internals

We ran into three main problems developing this: Exceptions, callbacks and accessing socket methods. This is what this chapter is about.

1.4.1 Exceptions

We realized early that most of the exceptions would be raised by the I/O functions of OpenSSL, so it felt natural to mimic OpenSSL’s error code system, translating them into Python exceptions. This naturally gives us the exceptions SSL.ZeroReturnError, SSL.WantReadError, SSL.WantWriteError, SSL.WantX509LookupError and SSL.SysCallError.

For more information about this, see section SSL — An interface to the SSL-specific parts of OpenSSL.

1.4.2 Callbacks

Callbacks were more of a problem when pyOpenSSL was written in C. Having switched to being written in Python using cffi, callbacks are now straightforward. The problems that originally existed no longer do (if you are interested in the details you can find descriptions of those problems in the version control history for this document).

1.4.3 Accessing Socket Methods

We quickly saw the benefit of wrapping socket methods in the SSL.Connection class, for an easy transition into using SSL. The problem here is that the socket module lacks a C API, and all the methods are declared static. One approach would be to have OpenSSL as a submodule to the socket module, placing all the code in socketmodule.c, but this is obviously not a good solution, since you might not want to import tonnes of extra stuff you’re not going to use when importing the socket module. The other approach is to somehow get a pointer to the method to be called, either the C function, or a callable Python object. This is not really a good solution either, since there’s a lot of lookups involved.

The way it works is that you have to supply a socket- like transport object to the SSL.Connection. The only requirement of this object is that it has a fileno() method that returns a file descriptor that’s valid at the C level (i.e. you can use the system calls read and write). If you want to use the connect() or accept() methods of
the `SSL.Connection` object, the transport object has to supply such methods too. Apart from them, any method lookups in the `SSL.Connection` object that fail are passed on to the underlying transport object.

Future changes might be to allow Python-level transport objects, that instead of having `fileno()` methods, have `read()` and `write()` methods, so more advanced features of Python can be used. This would probably entail some sort of OpenSSL BIOs, but converting Python strings back and forth is expensive, so this shouldn’t be used unless necessary. Other nice things would be to be able to pass in different transport objects for reading and writing, but then the `fileno()` method of `SSL.Connection` becomes virtually useless. Also, should the method resolution be used on the read-transport or the write-transport?

There are also examples in the pyOpenSSL repository that may help you getting started.

## 1.5 Meta

### 1.5.1 Backward Compatibility

pyOpenSSL has a very strong backward compatibility policy. Generally speaking, you shouldn’t ever be afraid of updating.

If breaking changes are needed do be done, they are:

1. ...announced in the *Changelog*.
2. ...the old behavior raises a `DeprecationWarning` for a year.
3. ...are done with another announcement in the *Changelog*.

### 1.5.2 Changelog

Versions are year-based with a strict backward-compatibility policy. The third digit is only for regressions.

#### 19.0.0 (2019-01-21)

**Backward-incompatible changes:**

- `X509Store.add_cert` no longer raises an error if you add a duplicate cert. #787

**Deprecations:**

*none*

**Changes:**

- pyOpenSSL now works with OpenSSL 1.1.1. #805
- pyOpenSSL now handles NUL bytes in `X509Name.get_components()` #804
18.0.0 (2018-05-16)

Backward-incompatible changes:

- The minimum cryptography version is now 2.2.1.
- Support for Python 2.6 has been dropped.

Deprecations:

none

Changes:

- Added `Connection.get_certificate` to retrieve the local certificate. #733
- `OpenSSL.SSL.Connection` now sets `SSL_MODE_AUTO_RETRY` by default. #753
- Added `Context.set_tlsext_use_srtp` to enable negotiation of SRTP keying material. #734

17.5.0 (2017-11-30)

Backward-incompatible changes:

- The minimum cryptography version is now 2.1.4.

Deprecations:

none

Changes:

- Fixed a potential use-after-free in the verify callback and resolved a memory leak when loading PKCS12 files with `cacerts`. #723
- Added `Connection.export_keying_material` for RFC 5705 compatible export of keying material. #725

17.4.0 (2017-11-21)

Backward-incompatible changes:

none
Deprecations:

none

Changes:

- Re-added a subset of the OpenSSL.rand module. This subset allows conscientious users to reseed the OpenSSL CSPRNG after fork. #708
- Corrected a use-after-free when reusing an issuer or subject from an X509 object after the underlying object has been mutated. #709

17.3.0 (2017-09-14)

Backward-incompatible changes:

- Dropped support for Python 3.3. #677
- Removed the deprecated OpenSSL.rand module. This is being done ahead of our normal deprecation schedule due to its lack of use and the fact that it was becoming a maintenance burden. os.urandom() should be used instead. #675

Deprecations:

- Deprecated OpenSSL.tsafe. #673

Changes:

- Fixed a memory leak in OpenSSL.crypto.CRL. #690
- Fixed a memory leak when verifying certificates with OpenSSL.crypto.X509StoreContext. #691

17.2.0 (2017-07-20)

Backward-incompatible changes:

none

Deprecations:

- Deprecated OpenSSL.rand-callers should use os.urandom() instead. #658
Changes:

- Fixed a bug causing `Context.set_default_verify_paths()` to not work with cryptography `manylinux1` wheels on Python 3.x. #665
- Fixed a crash with (EC)DSA signatures in some cases. #670

17.1.0 (2017-06-30)

Backward-incompatible changes:

- Removed the deprecated `OpenSSL.rand.egd()` function. Applications should prefer `os.urandom()` for random number generation. #630
- Removed the deprecated default `digest` argument to `OpenSSL.crypto.CRL.export()`. Callers must now always pass an explicit `digest`. #652
- Fixed a bug with `ASN1_TIME` casting in `X509.set_notBefore()`, `X509.set_notAfter()`, `Revoked.set_rev_date()`, `Revoked.set_nextUpdate()`, and `Revoked.set_lastUpdate()`. You must now pass times in the form `YYYYMMDDhhmmsZ`. `YYYYMMDDhhmss+hhmm` and `YYYYMMDDhhmss-hhmm` will no longer work. #612

Deprecations:

- Deprecated the legacy “Type” aliases: `ContextType`, `ConnectionType`, `PKeyType`, `X509NameType`, `X509ExtensionType`, `X509ReqType`, `X509Type`, `X509StoreType`, `CRLType`, `PKCS7Type`, `PKCS12Type`, `NetscapeSPKIType`. The names without the “Type”-suffix should be used instead.

Changes:

- Added `OpenSSL.crypto.X509.from_cryptography()` and `OpenSSL.crypto.X509.to_cryptography()` for converting X.509 certificate to and from pyca/cryptography objects. #640
- Added `OpenSSL.crypto.X509Req.from_cryptography()`, `OpenSSL.crypto.X509Req.to_cryptography()`, `OpenSSL.crypto.CRL.from_cryptography()`, and `OpenSSL.crypto.CRL.to_cryptography()` for converting X.509 CSRs and CRLs to and from pyca/cryptography objects. #645
- Added `OpenSSL.debug` that allows to get an overview of used library versions (including linked OpenSSL) and other useful runtime information using `python -m OpenSSL.debug`. #620
- Added a fallback path to `Context.set_default_verify_paths()` to accommodate the upcoming release of cryptography `manylinux1` wheels. #633

17.0.0 (2017-04-20)

Backward-incompatible changes:

`none`
Deprecations:

none

Changes:

- Added `OpenSSL.X509Store.set_time()` to set a custom verification time when verifying certificate chains. #567
- Added a collection of functions for working with OCSP stapling. None of these functions make it possible to validate OCSP assertions, only to staple them into the handshake and to retrieve the stapled assertion if provided. Users will need to write their own code to handle OCSP assertions. We specifically added: `Context.set_ocsp_server_callback()`, `Context.set_ocsp_client_callback()`, and `Connection.request_ocsp()`. #580
- Changed the SSL module’s memory allocation policy to avoid zeroing memory it allocates when unnecessary. This reduces CPU usage and memory allocation time by an amount proportional to the size of the allocation. For applications that process a lot of TLS data or that use very large allocations this can provide considerable performance improvements. #578
- Automatically set `SSL_CTX_set_ecdh_auto()` on `OpenSSL.SSL.Context`. #575
- Fix empty exceptions from `OpenSSL.crypto.load_privatekey()`. #581

16.2.0 (2016-10-15)

Backward-incompatible changes:

none

Deprecations:

none

Changes:

- Fixed compatibility errors with OpenSSL 1.1.0.
  - Fixed an issue that caused failures with subinterpreters and embedded Pythons. #552

16.1.0 (2016-08-26)

Backward-incompatible changes:

none
Deprecations:

- Dropped support for OpenSSL 0.9.8.

Changes:

- Fix memory leak in `OpenSSL.crypto.dump_privatekey()` with FILETYPE_TEXT. #496
- Enable use of CRL (and more) in verify context. #483
- `OpenSSL.crypto.PKey` can now be constructed from `cryptography` objects and also exported as such. #439
- Support newer versions of `cryptography` which use opaque structs for OpenSSL 1.1.0 compatibility.

16.0.0 (2016-03-19)

This is the first release under full stewardship of PyCA. We have made many changes to make local development more pleasing. The test suite now passes both on Linux and OS X with OpenSSL 0.9.8, 1.0.1, and 1.0.2. It has been moved to `pytest`, all CI test runs are part of `tox` and the source code has been made fully `flake8` compliant.

We hope to have lowered the barrier for contributions significantly but are open to hear about any remaining frustrations.

Backward-incompatible changes:

- Python 3.2 support has been dropped. It never had significant real world usage and has been dropped by our main dependency `cryptography`. Affected users should upgrade to Python 3.3 or later.

Deprecations:

- The support for EGD has been removed. The only affected function `OpenSSL.rand.egd()` now uses `os.urandom()` to seed the internal PRNG instead. Please see `pyca/cryptography#1636` for more background information on this decision. In accordance with our backward compatibility policy `OpenSSL.rand.egd()` will be removed no sooner than a year from the release of 16.0.0.

Please note that you should use `urandom` for all your secure random number needs.

- Python 2.6 support has been deprecated. Our main dependency `cryptography` deprecated 2.6 in version 0.9 (2015-05-14) with no time table for actually dropping it. pyOpenSSL will drop Python 2.6 support once `cryptography` does.

Changes:

- Fixed `OpenSSL.SSL.Context.set_session_id`, `OpenSSL.SSL.Connection.renegotiate`, `OpenSSL.SSL.Connection.renegotiate_pending`, and `OpenSSL.SSL.Context.load_client_ca`. They were lacking an implementation since 0.14. #422
- Fixed segmentation fault when using keys larger than 4096-bit to sign data. #428
• Fixed AttributeError when OpenSSL.SSL.Connection.get_app_data() was called before setting any app data. #304

• Added OpenSSL.crypto.dump_publickey() to dump OpenSSL.crypto.PKey objects that represent public keys, and OpenSSL.crypto.load_publickey() to load such objects from serialized representations. #382

• Added OpenSSL.crypto.dump_crl() to dump a certificate revocation list out to a string buffer. #368

• Added OpenSSL.SSL.Connection.get_state_string() using the OpenSSL binding state_string_long. #358

• Added support for the socket.MSG_PEEK flag to OpenSSL.SSL.Connection.recv() and OpenSSL.SSL.Connection.recv_into(). #294

• Added OpenSSL.SSL.Connection.get_protocol_version() and OpenSSL.SSL.Connection.get_protocol_version_name(). #244

• Switched to utf8string mask by default. OpenSSL formerly defaulted to a T61String if there were UTF-8 characters present. This was changed to default to UTF8String in the config around 2005, but the actual code didn’t change it until late last year. This will default us to the setting that actually works. To revert this you can call OpenSSL.crypto._lib.ASN1_STRING_set_default_mask_asc(b"default"). #234

Older Changelog Entries

The changes from before release 16.0.0 are preserved in the repository.
Indices and tables

- genindex
- modindex
- search
Python Module Index

O
OpenSSL, 6
OpenSSL.crypto, 6
OpenSSL.SSL, 26
from cryptography() (OpenSSL.crypto.X509 class method), 10
from cryptography() (OpenSSL.crypto.X509Req class method), 15
from cryptography_key() (OpenSSL.crypto.PKey class method), 19

generate_key() (OpenSSL.crypto.PKey method), 19
get_alpn_proto_negotiated() (OpenSSL.SSL.Connection method), 36
get_app_data() (OpenSSL.SSL.Connection method), 36
get_app_data() (OpenSSL.SSL.Context method), 29
get_ca_certificates() (OpenSSL.crypto.PKCS12 method), 20
get_certificate() (OpenSSL.crypto.PKCS12 method), 21
get_certificate() (OpenSSL.crypto.PKCS12 method), 21
generate_key() (OpenSSL.crypto.PKey method), 19
get_alpn_proto_negotiated() (OpenSSL.SSL.Connection method), 36
get_app_data() (OpenSSL.SSL.Connection method), 36
get_app_data() (OpenSSL.SSL.Connection method), 36
get_components() (OpenSSL.crypto.X509Name method), 14
get_context() (OpenSSL.SSL.Connection method), 37
get_critical() (OpenSSL.crypto.X509Extension method), 22
get_data() (OpenSSL.crypto.X509Extension method), 22
get_elliptic_curve() (OpenSSL.crypto.X509Extension method), 22
get_elliptic_curve() (in module OpenSSL.crypto), 6
get_elliptic_curves() (in module OpenSSL.crypto), 6
get_extension() (OpenSSL.crypto.X509 method), 10
get_extension_count() (OpenSSL.crypto.X509 method), 11
get_extensions() (OpenSSL.crypto.X509Req method), 15
get_finished() (OpenSSL.SSL.Connection method), 37
get_friendlyname() (OpenSSL.crypto.PKCS12 method), 21
get_issuer() (OpenSSL.crypto.CRL method), 23
get_issuer() (OpenSSL.crypto.X509 method), 11
generate_key() (OpenSSL.crypto.PKey method), 19
get_alpn_proto_negotiated() (OpenSSL.SSL.Connection method), 37
get_notAfter() (OpenSSL.crypto.X509 method), 11
get_notBefore() (OpenSSL.crypto.X509 method), 11
get_peer_cert_chain() (OpenSSL.SSL.Connection method), 37
get_peer_certificate() (OpenSSL.SSL.Connection method), 37
get_peer_finished() (OpenSSL.SSL.Connection method), 37
get_privatekey() (OpenSSL.crypto.PKCS12 method), 21
get_protocol_version() (OpenSSL.SSL.Connection method), 37
get_protocol_version_name() (OpenSSL.SSL.Connection method), 37
get_pubkey() (OpenSSL.crypto.NetssapKI method), 22
get_pubkey() (OpenSSL.crypto.X509 method), 11
get_pubkey() (OpenSSL.crypto.X509Req method), 15
get_reason() (OpenSSL.crypto.Revoked method), 25
get_rev_date() (OpenSSL.crypto.Revoked method), 25
get_revoked() (OpenSSL.crypto.CRL method), 24
get_serial() (OpenSSL.crypto.Revoked method), 25
get_serial_number() (OpenSSL.crypto.X509 method), 11
get_servername() (OpenSSL.SSL.Connection method), 37
get_session() (OpenSSL.SSL.Connection method), 38
get_session_cache_mode() (OpenSSL.SSL.Context method), 29
get_short_name() (OpenSSL.crypto.X509Extension method), 22
get_shutdown() (OpenSSL.SSL.Connection method), 38
get_signature_algorithm() (OpenSSL.crypto.X509 method), 12
get_state_string() (OpenSSL.SSL.Connection method), 38
get_subject() (OpenSSL.crypto.X509 method), 12
get_subject() (OpenSSL.crypto.X509Req method), 15
get_timeout() (OpenSSL.SSL.Context method), 29
get_type_name() (OpenSSL.crypto.PKCS7 method), 20
get_verify_depth() (OpenSSL.SSL.Context method), 30
get_verify_mode() (OpenSSL.SSL.Context method), 30

Index
method), 30
get_version() (OpenSSL.crypto.X509 method), 12
get_version() (OpenSSL.crypto.X509Req method), 15
gmtime_adj_notAfter() (OpenSSL.crypto.X509 method), 12
gmtime_adj_notBefore() (OpenSSL.crypto.X509 method), 12

H
has_expired() (OpenSSL.crypto.X509 method), 12
hash() (OpenSSL.crypto.X509Name method), 14

L
load_certificate() (in module OpenSSL.crypto), 7
load_certificate_request() (in module OpenSSL.crypto), 7
load_client_ca() (OpenSSL.SSL.Context method), 30
load_crl() (in module OpenSSL.crypto), 9
load_pkcs12() (in module OpenSSL.crypto), 9
load_privatekey() (in module OpenSSL.crypto), 8
load_publickey() (in module OpenSSL.crypto), 8
load_tmp_dh() (OpenSSL.SSL.Context method), 30
load_verify_locations() (OpenSSL.SSL.Context method), 30

M
makefile() (OpenSSL.SSL.Connection method), 38
master_key() (OpenSSL.SSL.Connection method), 38

N
NetscapeSPKI (class in OpenSSL.crypto), 22

O
OP_EPHEMERAL_RSA (in module OpenSSL.SSL), 27
OP_NO_COMPRESSION (in module OpenSSL.SSL), 27
OP_NO_SSLv2 (in module OpenSSL.SSL), 27
OP_NO_SSLv3 (in module OpenSSL.SSL), 27
OP_NO_TICKET (in module OpenSSL.SSL), 27
OP_NO_TLSv1 (in module OpenSSL.SSL), 27
OP_NO_TLSv1_1 (in module OpenSSL.SSL), 27
OP_NO_TLSv1_2 (in module OpenSSL.SSL), 27
OP_SINGLE_DH_USE (in module OpenSSL.SSL), 26
OP_SINGLE_ECDH_USE (in module OpenSSL.SSL), 26
OpenSSL (module), 6
OpenSSL.crypto (module), 6
OpenSSL.SSL (module), 26
OPENSSL_VERSION_NUMBER (in module OpenSSL.SSL), 27

P
pending() (OpenSSL.SSL.Connection method), 38
PKCS12 (class in OpenSSL.crypto), 20
PKCS7 (class in OpenSSL.crypto), 20
PKKey (class in OpenSSL.crypto), 19

R
read() (OpenSSL.SSL.Connection method), 38
recv() (OpenSSL.SSL.Connection method), 38
recv_into() (OpenSSL.SSL.Connection method), 38
renegotiate() (OpenSSL.SSL.Connection method), 39
renegotiate_pending() (OpenSSL.SSL.Connection method), 39
request_ocsp() (OpenSSL.SSL.Connection method), 39
Revoked (class in OpenSSL.crypto), 25

S
send() (OpenSSL.SSL.Connection method), 39
sendall() (OpenSSL.SSL.Connection method), 39
server_random() (OpenSSL.SSL.Connection method), 39
SESS_CACHE_BOTH (in module OpenSSL.SSL), 27
SESS_CACHE_CLIENT (in module OpenSSL.SSL), 27
SESS_CACHE_NO_AUTO_CLEAR (in module OpenSSL.SSL), 27
SESS_CACHE_NO_INTERNAL (in module OpenSSL.SSL), 27
SESS_CACHE_NO_INTERNAL_LOOKUP (in module OpenSSL.SSL), 27
SESS_CACHE_NO_INTERNAL_STORE (in module OpenSSL.SSL), 27
SESS_CACHE_OFF (in module OpenSSL.SSL), 27
SESS_CACHE_SERVER (in module OpenSSL.SSL), 27
Session (class in OpenSSL.SSL), 28
set_accept_state() (OpenSSL.SSL.Connection method), 39
set_alpn_protos() (OpenSSL.SSL.Connection method), 39
set_alpn_protos() (OpenSSL.SSL.Context method), 30
set_alpn_select_callback() (OpenSSL.SSL.Context method), 30
set_app_data() (OpenSSL.SSL.Connection method), 40
set_app_data() (OpenSSL.SSL.Context method), 30
set_ca_certificates() (OpenSSL.crypto.PKCS12 method), 21
set_cipher_list() (OpenSSL.crypto.PKCS12 method), 21
set_cipher_list() (OpenSSL.SSL.Context method), 30
set_client_ca_list()  (OpenSSL.SSL.Context method), 31
set_connect_state()  (OpenSSL.SSL.Connection method), 40
set_context()  (OpenSSL.SSL.Connection method), 40
set_default_verify_paths()  (OpenSSL.SSL.Context method), 31
set_flags()  (OpenSSL.crypto.X509Store method), 17
set_friendlyname()  (OpenSSL.crypto.PKCS12 method), 21
set_info_callback()  (OpenSSL.SSL.Context method), 31
set_issuer()  (OpenSSL.crypto.X509 method), 12
set_lastUpdate()  (OpenSSL.crypto.CRL method), 24
set_mode()  (OpenSSL.SSL.Context method), 31
set_nextUpdate()  (OpenSSL.crypto.CRL method), 24
set_notAfter()  (OpenSSL.crypto.X509 method), 12
set_notBefore()  (OpenSSL.crypto.X509 method), 13
set_npn_advertise_callback()  (OpenSSL.SSL.Context method), 31
set_npn_select_callback()  (OpenSSL.SSL.Context method), 31
set_ocsp_client_callback()  (OpenSSL.SSL.Context method), 32
set_ocsp_server_callback()  (OpenSSL.SSL.Context method), 32
set_options()  (OpenSSL.SSL.Context method), 32
set_passwd_cb()  (OpenSSL.SSL.Context method), 32
set_privatekey()  (OpenSSL.crypto.PKCS12 method), 21
set_pubkey()  (OpenSSL.crypto.X509 method), 13
set_pubkey()  (OpenSSL.crypto.X509Req method), 15
set_reason()  (OpenSSL.crypto.Revoked method), 25
set_rev_date()  (OpenSSL.crypto.Revoked method), 25
set_serial()  (OpenSSL.crypto.Revoked method), 26
set_serial_number()  (OpenSSL.crypto.X509 method), 13
set_session()  (OpenSSL.SSL.Connection method), 40
set_session_cache_mode()  (OpenSSL.SSL.Context method), 32
set_session_id()  (OpenSSL.SSL.Context method), 33
set_shutdown()  (OpenSSL.SSL.Connection method), 40
set_store()  (OpenSSL.crypto.X509StoreContext method), 18
set_subject()  (OpenSSL.crypto.X509 method), 13
set_time()  (OpenSSL.crypto.X509Store method), 17
set_timeout()  (OpenSSL.SSL.Context method), 33
set_tlsext_host_name()  (OpenSSL.SSL.Connection method), 40
set_tlsext_servername_callback()  (OpenSSL.SSL.Context method), 33
set_tlsext_use_srtp()  (OpenSSL.SSL.Context method), 33
set_tmp_ecdh()  (OpenSSL.SSL.Context method), 33
set_verify()  (OpenSSL.SSL.Context method), 33
set_verify_depth()  (OpenSSL.SSL.Context method), 34
set_version()  (OpenSSL.crypto.CRL method), 24
set_version()  (OpenSSL.crypto.X509 method), 13
set_version()  (OpenSSL.crypto.X509Req method), 15
shutdown()  (OpenSSL.SSL.Connection method), 40
sign()  (in module OpenSSL.crypto), 9
sign()  (OpenSSL.crypto.CRL method), 24
sign()  (OpenSSL.crypto.NetscapeSPKI method), 22
sign()  (OpenSSL.crypto.X509 method), 13
sign()  (OpenSSL.crypto.X509Req method), 16
sock_shutdown()  (OpenSSL.SSL.Connection method), 40
SSLEAY_BUILT_ON  (in module OpenSSL.SSL), 27
SSLEAY_CFLAGS  (in module OpenSSL.SSL), 27
SSLEAY_DIR  (in module OpenSSL.SSL), 27
SSLEAY_PLATFORM  (in module OpenSSL.SSL), 27
SSLEAY_VERSION  (in module OpenSSL.SSL), 27
SSLeay_version()  (in module OpenSSL.SSL), 27
SSLv23_METHOD  (in module OpenSSL.SSL), 26
SSLv2_METHOD  (in module OpenSSL.SSL), 26
SSLv3_METHOD  (in module OpenSSL.SSL), 26
subject_name_hash()  (OpenSSL.crypto.X509 method), 13
SysCallError, 29

T

TLSv1_1_METHOD  (in module OpenSSL.SSL), 26
TLSv1_2_METHOD  (in module OpenSSL.SSL), 26
TLSv1_METHOD  (in module OpenSSL.SSL), 26
to_cryptography()  (OpenSSL.crypto.CRL method), 25
to_cryptography()  (OpenSSL.crypto.X509 method), 13
to_cryptography()  (OpenSSL.crypto.X509Req method), 16
to_cryptography_key()  (OpenSSL.crypto.PKey method), 19
total_renegotiations()
    (OpenSSL.SSL.Connection method), 40
type()
    (OpenSSL.crypto.Pkey method), 19
TYPE_DSA
    (in module OpenSSL.crypto), 19
type_is_data()
    (OpenSSL.crypto.PKCS7 method), 20
type_is_enveloped()
    (OpenSSL.crypto.PKCS7 method), 20
type_is_signed()
    (OpenSSL.crypto.PKCS7 method), 20
type_is_signedAndEnveloped()
    (OpenSSL.crypto.PKCS7 method), 20
TYPE_RSA
    (in module OpenSSL.crypto), 19

U
use_certificate()
    (OpenSSL.SSL.Context method), 34
use_certificate_chain_file()
    (OpenSSL.SSL.Context method), 34
use_certificate_file()
    (OpenSSL.SSL.Context method), 34
use_privatekey()
    (OpenSSL.SSL.Context method), 34
use_privatekey_file()
    (OpenSSL.SSL.Context method), 34

V
verify()
    (in module OpenSSL.crypto), 10
verify()
    (OpenSSL.crypto.NetscapeSPKI method), 23
verify()
    (OpenSSL.crypto.X509Req method), 16
verify_certificate()
    (OpenSSL.crypto.X509StoreContext method), 18
VERIFY_FAIL_IF_NO_PEER_CERT
    (in module OpenSSL.SSL), 26
VERIFY_NONE
    (in module OpenSSL.SSL), 26
VERIFY_PEER
    (in module OpenSSL.SSL), 26

W
want_read()
    (OpenSSL.SSL.Connection method), 40
want_write()
    (OpenSSL.SSL.Connection method), 41
WantReadError, 28
WantWriteError, 28
WantX509LookupError, 28
write()
    (OpenSSL.SSL.Connection method), 41

X
X509
    (class in OpenSSL.crypto), 10
X509Extension
    (class in OpenSSL.crypto), 21
X509Name
    (class in OpenSSL.crypto), 14
X509Req
    (class in OpenSSL.crypto), 15
X509Store
    (class in OpenSSL.crypto), 16