josepy Documentation

Release 1.1.0

Let's Encrypt Project

Apr 12, 2022

CONTENTS:

Javascript Object Signing and Encryption (JOSE).

This package is a Python implementation of the standards developed by IETF Javascript Object Signing and Encryption (Active WG), in particular the following RFCs:

- JSON Web Algorithms (JWA)
- JSON Web Key (JWK)
- JSON Web Signature (JWS)

Originally developed as part of the ACME protocol implementation.

CHAPTER

ONE

JOSE BASE64

JOSE Base64 is defined as:

- URL-safe Base64
- padding stripped

`josepy.b64.b64encode(data)`

JOSE Base64 encode.

Parameters **data** (*bytes*) – Data to be encoded.

Returns JOSE Base64 string.

Return type bytes

Raises **TypeError** – if `data` is of incorrect type

`josepy.b64.b64decode(data)`

JOSE Base64 decode.

Parameters **data** (*bytes or unicode*) – Base64 string to be decoded. If it's unicode, then only ASCII characters are allowed.

Returns Decoded data.

Return type bytes

Raises

- **TypeError** – if input is of incorrect type
- **ValueError** – if input is unicode with non-ASCII characters

CHAPTER

TWO

ERRORS

JOSE errors.

exception josepy.errors.**Error**

Generic JOSE Error.

exception josepy.errors.**DeserializationError**

JSON deserialization error.

exception josepy.errors.**SerializationError**

JSON serialization error.

exception josepy.errors.**UnrecognizedTypeError**(*typ*, *jobj*)

Unrecognized type error.

Variables

- **typ** (*str*) – The unrecognized type of the JSON object.
- **jobj** – Full JSON object.

CHAPTER

THREE

INTERFACES

JOSE interfaces.

class josepy.interfaces.**JSONDeSerializable**

Interface for (de)serializable JSON objects.

Please recall, that standard Python library implements `json.JSONEncoder` and `json.JSONDecoder` that perform translations based on respective *conversion tables* that look pretty much like the one below (for complete tables see relevant Python documentation):

JSON	Python
object	dict
...	...

While the above **conversion table** is about translation of JSON documents to/from the basic Python types only, *JSONDeSerializable* introduces the following two concepts:

serialization Turning an arbitrary Python object into Python object that can be encoded into a JSON document. **Full serialization** produces a Python object composed of only basic types as required by the *conversion table*. **Partial serialization** (accomplished by *to_partial_json()*) produces a Python object that might also be built from other *JSONDeSerializable* objects.

deserialization Turning a decoded Python object (necessarily one of the basic types as required by the *conversion table*) into an arbitrary Python object.

Serialization produces **serialized object** ("partially serialized object" or "fully serialized object" for partial and full serialization respectively) and deserialization produces **deserialized object**, both usually denoted in the source code as `jobj`.

Wording in the official Python documentation might be confusing after reading the above, but in the light of those definitions, one can view `json.JSONDecoder.decode()` as decoder and deserializer of basic types, `json.JSONEncoder.default()` as serializer of basic types, `json.JSONEncoder.encode()` as serializer and encoder of basic types.

One could extend `json` to support arbitrary object (de)serialization either by:

- overriding `json.JSONDecoder.decode()` and `json.JSONEncoder.default()` in subclasses
- or passing `object_hook` argument (or `object_hook_pairs`) to `json.load()`/`json.loads()` or `default` argument for `json.dump()`/`json.dumps()`.

Interestingly, `default` is required to perform only partial serialization, as `json.dumps()` applies `default` recursively. This is the idea behind making *to_partial_json()* produce only partial serialization, while providing custom *json_dumps()* that dumps with `default` set to *json_dump_default()*.

To make further documentation a bit more concrete, please, consider the following imaginatory implementation example:

```
class Foo(JSONDeSerializable):
    def to_partial_json(self):
        return 'foo'

    @classmethod
    def from_json(cls, jobj):
        return Foo()

class Bar(JSONDeSerializable):
    def to_partial_json(self):
        return [Foo(), Foo()]

    @classmethod
    def from_json(cls, jobj):
        return Bar()
```

abstract to_partial_json()

Partially serialize.

Following the example, **partial serialization** means the following:

```
assert isinstance(Bar().to_partial_json()[0], Foo)
assert isinstance(Bar().to_partial_json()[1], Foo)

# in particular...
assert Bar().to_partial_json() != ['foo', 'foo']
```

Raises *josepy.errors.SerializationError* – in case of any serialization error.

Returns Partially serializable object.

to_json()

Fully serialize.

Again, following the example from before, **full serialization** means the following:

```
assert Bar().to_json() == ['foo', 'foo']
```

Raises *josepy.errors.SerializationError* – in case of any serialization error.

Returns Fully serialized object.

abstract classmethod from_json(*jobj*)

Deserialize a decoded JSON document.

Parameters **jobj** – Python object, composed of only other basic data types, as decoded from JSON document. Not necessarily `dict` (as decoded from "JSON object" document).

Raises *josepy.errors.DeserializationError* – if decoding was unsuccessful, e.g. in case of unparseable X509 certificate, or wrong padding in JOSE base64 encoded string, etc.

classmethod json_loads(*json_string*)

Deserialize from JSON document string.

json_dumps(***kwargs*)

Dump to JSON string using proper serializer.

Returns JSON document string.

Return type str

json_dumps_pretty()

Dump the object to pretty JSON document string.

Return type str

classmethod json_dump_default(*python_object*)

Serialize Python object.

This function is meant to be passed as `default` to `json.dump()` or `json.dumps()`. They call `default(python_object)` only for non-basic Python types, so this function necessarily raises `TypeError` if `python_object` is not an instance of `IJSONSerializable`.

Please read the class docstring for more information.

CHAPTER

FOUR

JSON UTILITIES

JSON (de)serialization framework.

The framework presented here is somewhat based on Go's "json" package (especially the `omitempty` functionality).

class josepy.json_util.**Field**(*json_name*, *default=None*, *omitempty=False*, *decoder=None*, *encoder=None*)

JSON object field.

Field is meant to be used together with *JSONObjectWithFields*.

`encoder` (`decoder`) is a callable that accepts a single parameter, i.e. a value to be encoded (decoded), and returns the serialized (deserialized) value. In case of errors it should raise *SerializationError* (*DeserializationError*).

Note, that `decoder` should perform partial serialization only.

Variables

- **json_name** (*str*) – Name of the field when encoded to JSON.
- **default** – Default value (used when not present in JSON object).
- **omitempty** (*bool*) – If `True` and the field value is empty, then it will not be included in the serialized JSON object, and `default` will be used for deserialization. Otherwise, if `False`, field is considered as required, value will always be included in the serialized JSON objected, and it must also be present when deserializing.

omit(*value*)

Omit the value in output?

decoder(*fdec*)

Descriptor to change the decoder on JSON object field.

encoder(*fenc*)

Descriptor to change the encoder on JSON object field.

decode(*value*)

Decode a value, optionally with context JSON object.

encode(*value*)

Encode a value, optionally with context JSON object.

classmethod default_decoder(*value*)

Default decoder.

Recursively deserialize into immutable types (*josepy.util.frozendict* instead of `dict()`, `tuple()` instead of `list()`).

classmethod default_encoder(*value*)

Default (passthrough) encoder.

class josepy.json_util.**JSONObjectWithFieldsMeta**(*name*, *bases*, *dikt*)

Metaclass for *JSONObjectWithFields* and its subclasses.

It makes sure that, for any class `cls` with `__metaclass__` set to `JSONObjectWithFieldsMeta`:

1. All fields (attributes of type *Field*) in the class definition are moved to the `cls._fields` dictionary, where keys are field attribute names and values are fields themselves.
2. `cls.__slots__` is extended by all field attribute names (i.e. not `Field.json_name`). Original `cls.__slots__` are stored in `cls._orig_slots`.

In a consequence, for a field attribute name `some_field`, `cls.some_field` will be a slot descriptor and not an instance of *Field*. For example:

```
some_field = Field('someField', default=())

class Foo(object):
    __metaclass__ = JSONObjectWithFieldsMeta
    __slots__ = ('baz',)
    some_field = some_field

assert Foo.__slots__ == ('some_field', 'baz')
assert Foo._orig_slots == ()
assert Foo.some_field is not Field

assert Foo._fields.keys() == ['some_field']
assert Foo._fields['some_field'] is some_field
```

As an implementation note, this metaclass inherits from `abc.ABCMeta` (and not the usual `type`) to mitigate the metaclass conflict (`ImmutableMap` and `JSONDeSerializable`, parents of *JSONObjectWithFields*, use `abc.ABCMeta` as its metaclass).

class josepy.json_util.**JSONObjectWithFields**(***kwargs*)

JSON object with fields.

Example:

```
class Foo(JSONObjectWithFields):
    bar = Field('Bar')
    empty = Field('Empty', omitempty=True)

    @bar.encoder
    def bar(value):
        return value + 'bar'

    @bar.decoder
    def bar(value):
        if not value.endswith('bar'):
            raise errors.DeserializationError('No bar suffix!')
        return value[:-3]

assert Foo(bar='baz').to_partial_json() == {'Bar': 'bazbar'}
assert Foo.from_json({'Bar': 'bazbar'}) == Foo(bar='baz')
```

(continues on next page)

(continued from previous page)

```
assert (Foo.from_json({'Bar': 'bazbar', 'Empty': '!'})
        == Foo(bar='baz', empty='!'))
assert Foo(bar='baz').bar == 'baz'
```

encode(*name*)

Encode a single field.

Parameters **name** (*str*) – Name of the field to be encoded.

Raises

- ***errors.SerializationError*** – if field cannot be serialized
- ***errors.Error*** – if field could not be found

fields_to_partial_json()

Serialize fields to JSON.

to_partial_json()

Partially serialize.

Following the example, **partial serialization** means the following:

```
assert isinstance(Bar().to_partial_json()[0], Foo)
assert isinstance(Bar().to_partial_json()[1], Foo)

# in particular...
assert Bar().to_partial_json() != ['foo', 'foo']
```

Raises ***josepy.errors.SerializationError*** – in case of any serialization error.

Returns Partially serializable object.

classmethod fields_from_json(*jobj*)

Deserialize fields from JSON.

classmethod from_json(*jobj*)

Deserialize a decoded JSON document.

Parameters **jobj** – Python object, composed of only other basic data types, as decoded from JSON document. Not necessarily `dict` (as decoded from "JSON object" document).

Raises ***josepy.errors.DeserializationError*** – if decoding was unsuccessful, e.g. in case of unparseable X509 certificate, or wrong padding in JOSE base64 encoded string, etc.

josepy.json_util.**encode_b64jose**(*data*)

Encode JOSE Base-64 field.

Parameters **data** (*bytes*) –

Return type unicode

josepy.json_util.**decode_b64jose**(*data*, *size=None*, *minimum=False*)

Decode JOSE Base-64 field.

Parameters

- **data** (*unicode*) –
- **size** (*int*) – Required length (after decoding).

- **minimum** (*bool*) – If `True`, then `size` will be treated as minimum required length, as opposed to exact equality.

Return type bytes

`josepy.json_util.`**encode_hex16**(*value*)

Hexlify.

Parameters **value** (*bytes*) –

Return type unicode

`josepy.json_util.`**decode_hex16**(*value*, *size=None*, *minimum=False*)

Decode hexlified field.

Parameters

- **value** (*unicode*) –
- **size** (*int*) – Required length (after decoding).
- **minimum** (*bool*) – If `True`, then `size` will be treated as minimum required length, as opposed to exact equality.

Return type bytes

`josepy.json_util.`**encode_cert**(*cert*)

Encode certificate as JOSE Base-64 DER.

Return type unicode

`josepy.json_util.`**decode_cert**(*b64der*)

Decode JOSE Base-64 DER-encoded certificate.

Parameters **b64der** (*unicode*) –

Return type `OpenSSL.crypto.X509` wrapped in *ComparableX509*

`josepy.json_util.`**encode_csr**(*csr*)

Encode CSR as JOSE Base-64 DER.

Return type unicode

`josepy.json_util.`**decode_csr**(*b64der*)

Decode JOSE Base-64 DER-encoded CSR.

Parameters **b64der** (*unicode*) –

Return type `OpenSSL.crypto.X509Req` wrapped in *ComparableX509*

class `josepy.json_util.`**TypedJSONObjectWithFields**(***kwargs*)

JSON object with type.

typ = NotImplemented

Type of the object. Subclasses must override.

type_field_name = 'type'

Field name used to distinguish different object types.

Subclasses will probably have to override this.

TYPES = NotImplemented

Types registered for JSON deserialization

classmethod register(*type_cls*, *typ=None*)

Register class for JSON deserialization.

classmethod get_type_cls(*jobj*)

Get the registered class for `jobj`.

to_partial_json()

Get JSON serializable object.

Returns Serializable JSON object representing ACME typed object. `validate()` will almost certainly not work, due to reasons explained in `josepy.interfaces.IJSONSerializable`.

Return type dict

classmethod from_json(*jobj*)

Deserialize ACME object from valid JSON object.

Raises *josepy.errors.UnrecognizedTypeError* – if type of the ACME object has not been registered.

CHAPTER

FIVE

JSON WEB ALGORITHMS

JSON Web Algorithms.

https://tools.ietf.org/html/draft-ietf-jose-json-web-algorithms-40

class josepy.jwa.**JWA**

JSON Web Algorithm.

class josepy.jwa.**JWASignature**(*name*)

Base class for JSON Web Signature Algorithms.

classmethod register(*signature_cls*)

Register class for JSON deserialization.

to_partial_json()

Partially serialize.

Following the example, **partial serialization** means the following:

```
assert isinstance(Bar().to_partial_json()[0], Foo)
assert isinstance(Bar().to_partial_json()[1], Foo)

# in particular...
assert Bar().to_partial_json() != ['foo', 'foo']
```

Raises *josepy.errors.SerializationError* – in case of any serialization error.

Returns Partially serializable object.

classmethod from_json(*jobj*)

Deserialize a decoded JSON document.

Parameters **jobj** – Python object, composed of only other basic data types, as decoded from JSON document. Not necessarily `dict` (as decoded from "JSON object" document).

Raises *josepy.errors.DeserializationError* – if decoding was unsuccessful, e.g. in case of unparseable X509 certificate, or wrong padding in JOSE base64 encoded string, etc.

abstract sign(*key*, *msg*)

Sign the `msg` using `key`.

abstract verify(*key*, *msg*, *sig*)

Verify the `msg` and `sig` using `key`.

josepy.jwa.**HS256 = HS256**
HMAC using SHA-256

josepy.jwa.**HS384 = HS384**
HMAC using SHA-384

josepy.jwa.**HS512 = HS512**
HMAC using SHA-512

josepy.jwa.**RS256 = RS256**
RSASSA-PKCS-v1_5 using SHA-256

josepy.jwa.**RS384 = RS384**
RSASSA-PKCS-v1_5 using SHA-384

josepy.jwa.**RS512 = RS512**
RSASSA-PKCS-v1_5 using SHA-512

josepy.jwa.**PS256 = PS256**
RSASSA-PSS using SHA-256 and MGF1 with SHA-256

josepy.jwa.**PS384 = PS384**
RSASSA-PSS using SHA-384 and MGF1 with SHA-384

josepy.jwa.**PS512 = PS512**
RSASSA-PSS using SHA-512 and MGF1 with SHA-512

josepy.jwa.**ES256 = ES256**
ECDSA using P-256 and SHA-256

josepy.jwa.**ES384 = ES384**
ECDSA using P-384 and SHA-384

josepy.jwa.**ES512 = ES512**
ECDSA using P-521 and SHA-512

CHAPTER

SIX

JSON WEB KEY

JSON Web Key.

class josepy.jwk.**JWK**(***kwargs*)

JSON Web Key.

cryptography_key_types = ()

Subclasses should override.

required = NotImplemented

Required members of public key's representation as defined by JWK/JWA.

thumbprint(*hash_function=<class 'cryptography.hazmat.primitives.hashes.SHA256'>*)

Compute JWK Thumbprint.

https://tools.ietf.org/html/rfc7638

Returns bytes

abstract public_key()

Generate JWK with public key.

For symmetric cryptosystems, this would return `self`.

classmethod load(*data*, *password=None*, *backend=None*)

Load serialized key as JWK.

Parameters

- **data** (*str*) – Public or private key serialized as PEM or DER.
- **password** (*str*) – Optional password.
- **backend** – A `PEMSerializationBackend` and `DERSerializationBackend` provider.

Raises *errors.Error* – if unable to deserialize, or unsupported JWK algorithm

Returns JWK of an appropriate type.

Return type *JWK*

class josepy.jwk.**JWKES**(***kwargs*)

ES JWK.

> **Warning:** This is not yet implemented!

fields_to_partial_json()

Serialize fields to JSON.

classmethod fields_from_json(*jobj*)

Deserialize fields from JSON.

public_key()

Generate JWK with public key.

For symmetric cryptosystems, this would return `self`.

class josepy.jwk.**JWKOct**(***kwargs*)

Symmetric JWK.

fields_to_partial_json()

Serialize fields to JSON.

classmethod fields_from_json(*jobj*)

Deserialize fields from JSON.

public_key()

Generate JWK with public key.

For symmetric cryptosystems, this would return `self`.

class josepy.jwk.**JWKRSA**(**args*, ***kwargs*)

RSA JWK.

Variables **key** – `RSAPrivateKey` or `RSAPublicKey` wrapped in *ComparableRSAKey*

public_key()

Generate JWK with public key.

For symmetric cryptosystems, this would return `self`.

classmethod fields_from_json(*jobj*)

Deserialize fields from JSON.

fields_to_partial_json()

Serialize fields to JSON.

CHAPTER

SEVEN

JSON WEB SIGNATURE

JSON Web Signature.

class josepy.jws.**MediaType**

MediaType field encoder/decoder.

PREFIX = 'application/'

MIME Media Type and Content Type prefix.

classmethod decode(*value*)

Decoder.

classmethod encode(*value*)

Encoder.

class josepy.jws.**Header**(***kwargs*)

JOSE Header.

Warning: This class supports **only** Registered Header Parameter Names (as defined in section 4.1 of the protocol). If you need Public Header Parameter Names (4.2) or Private Header Parameter Names (4.3), you must subclass and override `from_json()` and `to_partial_json()` appropriately.

Warning: This class does not support any extensions through the "crit" (Critical) Header Parameter (4.1.11) and as a conforming implementation, `from_json()` treats its occurrence as an error. Please subclass if you seek for a different behaviour.

Variables

- **x5tS256** – "x5t#S256"
- **typ** (*str*) – MIME Media Type, inc. *MediaType.PREFIX*.
- **cty** (*str*) – Content-Type, inc. *MediaType.PREFIX*.

not_omitted()

Fields that would not be omitted in the JSON object.

find_key()

Find key based on header.

Todo: Supports only "jwk" header parameter lookup.

Returns (Public) key found in the header.

Return type .JWK

Raises *josepy.errors.Error* – if key could not be found

class josepy.jws.**Signature**(***kwargs*)

JWS Signature.

Variables

- **combined** – Combined Header (protected and unprotected, *Header*).
- **protected** (*unicode*) – JWS protected header (Jose Base-64 decoded).
- **header** – JWS Unprotected Header (*Header*).
- **signature** (*str*) – The signature.

header_cls

alias of *josepy.jws.Header*

verify(*payload*, *key=None*)

Verify.

Parameters **key** (*JWK*) – Key used for verification.

classmethod sign(*payload*, *key*, *alg*, *include_jwk=True*, *protect=frozenset({})*, ***kwargs*)

Sign.

Parameters **key** (*JWK*) – Key for signature.

fields_to_partial_json()

Serialize fields to JSON.

classmethod fields_from_json(*jobj*)

Deserialize fields from JSON.

class josepy.jws.**JWS**(***kwargs*)

JSON Web Signature.

Variables

- **payload** (*str*) – JWS Payload.
- **signature** (*str*) – JWS Signatures.

signature_cls

alias of *josepy.jws.Signature*

verify(*key=None*)

Verify.

classmethod sign(*payload*, ***kwargs*)

Sign.

property signature

Get a singleton signature.

Return type *JWS.signature_cls*

to_compact()

Compact serialization.

Return type bytes

classmethod from_compact(*compact*)

Compact deserialization.

Parameters **compact** (*bytes*) –

to_partial_json(*flat=True*)

Partially serialize.

Following the example, **partial serialization** means the following:

```
assert isinstance(Bar().to_partial_json()[0], Foo)
assert isinstance(Bar().to_partial_json()[1], Foo)

# in particular...
assert Bar().to_partial_json() != ['foo', 'foo']
```

Raises *josepy.errors.SerializationError* – in case of any serialization error.

Returns Partially serializable object.

classmethod from_json(*jobj*)

Deserialize a decoded JSON document.

Parameters **jobj** – Python object, composed of only other basic data types, as decoded from JSON document. Not necessarily `dict` (as decoded from "JSON object" document).

Raises *josepy.errors.DeserializationError* – if decoding was unsuccessful, e.g. in case of unparseable X509 certificate, or wrong padding in JOSE base64 encoded string, etc.

class josepy.jws.CLI

JWS CLI.

classmethod sign(*args*)

Sign.

classmethod verify(*args*)

Verify.

classmethod run(*args=None*)

Parse arguments and sign/verify.

CHAPTER

EIGHT

UTILITIES

JOSE utilities.

class josepy.util.**abstractclassmethod**(*target*)

Descriptor for an abstract classmethod.

It augments the abc framework with an abstract classmethod. This is implemented as abc.abstractclassmethod in the standard Python library starting with version 3.2.

This particular implementation, allegedly based on Python 3.3 source code, is stolen from http://stackoverflow.com/questions/11217878/python-2-7-combine-abc-abstractmethod-and-classmethod.

class josepy.util.**ComparableX509**(*wrapped*)

Wrapper for OpenSSL.crypto.X509** objects that supports __eq__.

Variables **wrapped** – Wrapped certificate or certificate request.

class josepy.util.**ComparableKey**(*wrapped*)

Comparable wrapper for cryptography keys.

See https://github.com/pyca/cryptography/issues/2122.

public_key()

Get wrapped public key.

class josepy.util.**ComparableRSAKey**(*wrapped*)

Wrapper for cryptography RSA keys.

Wraps around:

- RSAPrivateKey
- RSAPublicKey

class josepy.util.**ImmutableMap**(***kwargs*)

Immutable key to value mapping with attribute access.

update(***kwargs*)

Return updated map.

class josepy.util.**frozendict**(**args*, ***kwargs*)

Frozen dictionary.

CHAPTER

NINE

CHANGELOG

9.1 1.1.0 (2018-04-13)

- Deprecated support for Python 2.6 and 3.3.
- Use the `sign` and `verify` methods when they are available in `cryptography` instead of the deprecated methods `signer` and `verifier`.

9.2 1.0.1 (2017-10-25)

Stop installing mock as part of the default but only as part of the testing dependencies.

9.3 1.0.0 (2017-10-13)

First release after moving the josepy package into a standalone library.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

j

INDEX

H

I

J

L

M

N

O

P

R

S

T

U

V