JWCrypto Documentation

Release 0.3.2

JWCrypto Contributors

August 31, 2016

Contents

JWCrypto is an implementation of the Javascript Object Signing and Encryption (JOSE) Web Standards as they are being developed in the JOSE IETF Working Group and related technology.

JWCrypto is Python2 and Python3 compatible and uses the Cryptography package for all the crypto functions.

Contents:

CHAPTER 1

JSON Web Key (JWK)

The jwk Module implements the JSON Web Key standard. A JSON Web Key is represented by a JWK object, related utility classes and functions are availbale in this module too.

1.1 Classes

class jwcrypto.jwk.**JWK**(***kwargs*)
Bases: object

JSON Web Key object

This object represent a Key. It must be instantiated by using the standard defined key/value pairs as arguents of the initialization function.

Creates a new JWK object.

The function arguments must be valid parameters as defined in the 'IANA JSON Web Key Set Parameters registry' and specified in the JWKParamsRegistry variable. The 'kty' parameter must always be provided and its value must be a valid one as defined by the 'IANA JSON Web Key Types registry' and specified in the JWKTypesRegistry variable. The valid key parameters per key type are defined in the JWKValuesregistry variable.

To generate a new random key call the class method generate() with the appropriate 'kty' parameter, and other parameters as needed (key size, public exponents, curve types, etc..)

Valid options per type, when generating new keys:

- oct: size(int)
- RSA: public_exponent(int), size(int)
- EC: curve(str) (one of P-256, P-384, P-521)

Deprecated: Alternatively if the 'generate' parameter is provided, with a valid key type as value then a new key will be generated according to the defaults or provided key strenght options (type specific).

Raises

- **InvalidJWKType** – if the key type is invalid
- **InvalidJWKValue** – if incorrect or inconsistent parameters are provided.

export(*private_key=True*)
Exports the key in the standard JSON format.

Parameters **private_key(bool)** – Whether to export the private key. Defaults to True.

export_public()
Exports the public key in the standard JSON format. This function is deprecated and maintained only for backwards compatibility, use export(private_key=False) instead.

get_curve(*arg*)
Gets the Elliptic Curve associated with the key.

Parameters **arg** – an optional curve name

Raises

- **InvalidJWKType** – the key is not an EC key.
- **InvalidJWKValue** – if the curve names is invalid.

get_op_key(*operation=None*, *arg=None*)
Get the key object associated to the requested opration. For example the public RSA key for the 'verify' operation or the private EC key for the 'decrypt' operation.

Parameters

- **operation** – The requested operation. The valid set of operations is availble in the `JWKOperationsRegistry` registry.
- **arg** – an optional, context specific, argument For example a curve name.

Raises

- **InvalidJWKOperation** – if the operation is unknown or not permitted with this key.
- **InvalidJWKUsage** – if the use constraints do not permit the operation.

thumbprint(*hashalg=<cryptography.hazmat.primitives.hashes.SHA256 object>*)
Returns the key thumbprint as specified by RFC 7638.

Parameters **hashalg** – A hash function (defaults to SHA256)

key_curve
The Curve Name.

key_id
The Key ID. Provided by the kid parameter if present, otherwise returns None.

key_type
The Key type

class jwcrypto.jwk.**JWKSet**(**args*, ***kwargs*)
Bases: `dict`

A set of JWK objects.

Inherits from the standard 'dict' bultin type. Creates a special key 'keys' that is of a type derived from 'set' The 'keys' attribute accepts only *jwcrypto.jwk.JWK* elements.

export(*private_keys=True*)
Exports a RFC 7517 keyset using the standard JSON format

Parameters **private_key(bool)** – Whether to export private keys. Defaults to True.

classmethod **from_json**(*keyset*)
Creates a RFC 7517 keyset from the standard JSON format.

Parameters **keyset** – The RFC 7517 representation of a JOSE Keyset.

get_key(*kid*)
Gets a key from the set. :param kid: the 'kid' key identifier.

import_keyset(*keyset*)
Imports a RFC 7517 keyset using the standard JSON format.

Parameters **keyset** – The RFC 7517 representation of a JOSE Keyset.

1.2 Exceptions

class jwcrypto.jwk.**InvalidJWKType**(*value=None*)
Bases: exceptions.Exception

Invalid JWK Type Exception.

This exception is raised when an invalid parameter type is used.

class jwcrypto.jwk.**InvalidJWKValue**
Bases: exceptions.Exception

Invalid JWK Value Exception.

This exception is raised when an invalid/unknown value is used in the context of an operation that requires specific values to be used based on the key type or other constraints.

class jwcrypto.jwk.**InvalidJWKOperation**(*operation*, *values*)
Bases: exceptions.Exception

Invalid JWK Operation Exception.

This exception is raised when an invalid key operation is requested, based on the key type and declared usage constraints.

class jwcrypto.jwk.**InvalidJWKUsage**(*use*, *value*)
Bases: exceptions.Exception

Invalid JWK usage Exception.

This exception is raised when an invalid key usage is requested, based on the key type and declared usage constraints.

1.3 Registries

jwcrypto.jwk.**JWKTypesRegistry**
Registry of valid Key Types

jwcrypto.jwk.**JWKValuesRegistry**
Registry of valid key values

jwcrypto.jwk.**JWKParamsRegistry**
Regstry of valid key parameters

jwcrypto.jwk.**JWKEllipticCurveRegistry**
Registry of allowed Elliptic Curves

jwcrypto.jwk.**JWKUseRegistry**
Registry of allowed uses

jwcrypto.jwk.**JWKOperationsRegistry**
Registry of allowed operations

1.4 Examples

Create a 256bit symmetric key::

```
>>> from jwcrypto import jwk
>>> key = jwk.JWK.generate(kty='oct', size=256)
```

Export the key with::

```
>>> key.export()
'{"k":"X6TBlwY2so8EwKZ2TFXM7XHSgWBKQJhcspzYydp5Y-o","kty":"oct"}'
```

Create a 2048bit RSA keypair::

```
>>> jwk.JWK.generate(kty='RSA', size=2048)
```

Create a P-256 EC keypair and export the public key::

```
>>> key = jwk.JWK.generate(kty='EC', crv='P-256')
>>> key.export(private_key=False)
'{"y":"VYlYwBfOTIICojCPfdUjnmkpN-g-lzZKxzjAoFmDRm8",
  "x":"3mdE0rODWRju6qqU01Kw5oPYdNxBOMisFvJFH1vEu9Q",
  "crv":"P-256","kty":"EC"}'
```

Import a P-256 Public Key::

```
>>> expkey = {"y":"VYlYwBfOTIICojCPfdUjnmkpN-g-lzZKxzjAoFmDRm8",
              "x":"3mdE0rODWRju6qqU01Kw5oPYdNxBOMisFvJFH1vEu9Q",
              "crv":"P-256","kty":"EC"}
>>> key = jwk.JWK(**expkey)
```

CHAPTER 2

JSON Web Signature (JWS)

The jws Module implements the JSON Web Signature standard. A JSON Web Signature is represented by a JWS object, related utility classes and functions are available in this module too.

2.1 Classes

class jwcrypto.jws.**JWS**(*payload=None*)
Bases: object

JSON Web Signature object

This object represent a JWS token.

Creates a JWS object.

Parameters **payload(bytes)** – An arbitrary value (optional).

add_signature(*key*, *alg=None*, *protected=None*, *header=None*)
Adds a new signature to the object.

Parameters

- **key** – A (*jwcrypto.jwk.JWK*) key of appropriate for the "alg" provided.
- **alg** – An optional algorithm name. If already provided as an element of the protected or unprotected header it can be safely omitted.
- **potected** – The Protected Header (optional)
- **header** – The Unprotected Header (optional)

Raises

- **InvalidJWSObject** – if no payload has been set on the object.
- **ValueError** – if the key is not a JWK object.
- **ValueError** – if the algorithm is missing or is not provided by one of the headers.
- **InvalidJWAAlgorithm** – if the algorithm is not valid, is unknown or otherwise not yet implemented.

deserialize(*raw_jws*, *key=None*, *alg=None*)
Deserialize a JWS token.

NOTE: Destroys any current status and tries to import the raw JWS provided.

Parameters

- **raw_jws** – a 'raw' JWS token (JSON Encoded or Compact notation) string.
- **key** – A (*jwcrypto.jwk.JWK*) verification key (optional). If a key is provided a verification step will be attempted after the object is successfully deserialized.
- **alg** – The signing algorithm (optional). usually the algorithm is known as it is provided with the JOSE Headers of the token.

Raises

- **InvalidJWSObject** – if the raw object is an invaid JWS token.
- **InvalidJWSSignature** – if the verification fails.

serialize(*compact=False*)
Serializes the object into a JWS token.

Parameters **compact(boolean)** – if True generates the compact representation, otherwise generates a standard JSON format.

Raises

- **InvalidJWSOperation** – if the object cannot serialized with the compact representation and *compat* is True.
- **InvalidJWSSignature** – if no signature has been added to the object, or no valid signature can be found.

verify(*key*, *alg=None*)
Verifies a JWS token.

Parameters

- **key** – The (*jwcrypto.jwk.JWK*) verification key.
- **alg** – The signing algorithm (optional). usually the algorithm is known as it is provided with the JOSE Headers of the token.

Raises **InvalidJWSSignature** – if the verification fails.

allowed_algs
Allowed algorithms.

The list of allowed algorithms. Can be changed by setting a list of algorithm names.

class jwcrypto.jws.**JWSCore**(*alg*, *key*, *header*, *payload*, *algs=None*)
Bases: object

The inner JWS Core object.

This object SHOULD NOT be used directly, the JWS object should be used instead as JWS perform necessary checks on the validity of the object and requested operations.

Core JWS token handling.

Parameters

- **alg** – The algorithm used to produce the signature. See RFC 7518
- **key** – A (*jwcrypto.jwk.JWK*) key of appropriate type for the "alg" provided in the 'protected' json string.
- **header** – A JSON string representing the protected header.
- **payload(bytes)** – An arbitrary value
- **algs** – An optional list of allowed algorithms

Raises

- **ValueError** – if the key is not a JWK object
- **InvalidJWAAlgorithm** – if the algorithm is not valid, is unknown or otherwise not yet implemented.

sign()
Generates a signature

verify(*signature*)
Verifies a signature

Raises **InvalidJWSSignature** – if the verification fails.

2.2 Variables

jwcrypto.jws.**default_allowed_algs = ['HS256', 'HS384', 'HS512', 'RS256', 'RS384', 'RS512', 'ES256', 'ES384', 'ES**
Default allowed algorithms

2.3 Exceptions

class jwcrypto.jws.**InvalidJWSSignature**(*message=None*, *exception=None*)
Bases: exceptions.Exception

Invalid JWS Signature.

This exception is raised when a signature cannot be validated.

class jwcrypto.jws.**InvalidJWSObject**(*message=None*, *exception=None*)
Bases: exceptions.Exception

Invalid JWS Object.

This exception is raised when the JWS Object is invalid and/or improperly formatted.

class jwcrypto.jws.**InvalidJWSOperation**(*message=None*, *exception=None*)
Bases: exceptions.Exception

Invalid JWS Object.

This exception is raised when a requested operation cannot be execute due to unsatisfied conditions.

2.4 Registries

jwcrypto.jws.**JWSHeaderRegistry**
Registry of valid header parameters

CHAPTER 3

JSON Web Encryption (JWE)

The jwe Module implements the JSON Web Encryption standard. A JSON Web Encryption is represented by a JWE object, related utility classes and functions are availbale in this module too.

3.1 Classes

class jwcrypto.jwe.**JWE**(*plaintext=None*, *protected=None*, *unprotected=None*, *aad=None*, *algs=None*)
Bases: object

JSON Web Encryption object

This object represent a JWE token.

Creates a JWE token.

Parameters

- **plaintext(bytes)** – An arbitrary plaintext to be encrypted.
- **protected** – A JSON string with the protected header.
- **unprotected** – A JSON string with the shared unprotected header.
- **aad(bytes)** – Arbitrary additional authenticated data
- **algs** – An optional list of allowed algorithms

add_recipient(*key*, *header=None*)
Encrypt the plaintext with the given key.

Parameters

- **key** – A JWK key or password of appropriate type for the 'alg' provided in the JOSE Headers.
- **header** – A JSON string representing the per-recipient header.

Raises

- **ValueError** – if the plaintext is missing or not of type bytes.
- **ValueError** – if the compression type is unknown.
- **InvalidJWAAlgorithm** – if the 'alg' provided in the JOSE headers is missing or unknown, or otherwise not implemented.

decrypt(*key*)
Decrypt a JWE token.

Parameters

- **key** – The (*jwcrypto.jwk.JWK*) decryption key.
- **key** – A (*jwcrypto.jwk.JWK*) decryption key or a password string (optional).

Raises

- **InvalidJWEOperation** – if the key is not a JWK object.
- **InvalidJWEData** – if the ciphertext can't be decrypted or the object is otherwise malformed.

deserialize(*raw_jwe*, *key=None*)
Deserialize a JWE token.

NOTE: Destroys any current status and tries to import the raw JWE provided.

Parameters

- **raw_jwe** – a 'raw' JWE token (JSON Encoded or Compact notation) string.
- **key** – A (*jwcrypto.jwk.JWK*) decryption key or a password string (optional). If a key is provided a decryption step will be attempted after the object is successfully deserialized.

Raises

- **InvalidJWEData** – if the raw object is an invaid JWE token.
- **InvalidJWEOperation** – if the decryption fails.

serialize(*compact=False*)
Serializes the object into a JWE token.

Parameters **compact (boolean)** – if True generates the compact representation, otherwise generates a standard JSON format.

Raises

- **InvalidJWEOperation** – if the object cannot serialized with the compact representation and *compact* is True.
- **InvalidJWEOperation** – if no recipients have been added to the object.

allowed_algs
Allowed algorithms.

The list of allowed algorithms. Can be changed by setting a list of algorithm names.

3.2 Variables

jwcrypto.jwe.**default_allowed_algs = ['RSA1_5', 'RSA-OAEP', 'RSA-OAEP-256', 'A128KW', 'A192KW', 'A256KW**
Default allowed algorithms

3.3 Exceptions

class jwcrypto.jwe.**InvalidJWEOperation**(*message=None*, *exception=None*)
Bases: exceptions.Exception

Invalid JWS Object.

This exception is raised when a requested operation cannot be execute due to unsatisfied conditions.

class jwcrypto.jwe.**InvalidJWEData**(*message=None*, *exception=None*)
Bases: exceptions.Exception

Invalid JWE Object.

This exception is raised when the JWE Object is invalid and/or improperly formatted.

class jwcrypto.jwe.**InvalidJWEKeyType**(*expected*, *obtained*)
Bases: exceptions.Exception

Invalid JWE Key Type.

This exception is raised when the provided JWK Key does not match the type required by the sepcified algorithm.

class jwcrypto.jwe.**InvalidJWEKeyLength**(*expected*, *obtained*)
Bases: exceptions.Exception

Invalid JWE Key Length.

This exception is raised when the provided JWK Key does not match the lenght required by the sepcified algorithm.

class jwcrypto.jwe.**InvalidCEKeyLength**(*expected*, *obtained*)
Bases: exceptions.Exception

Invalid CEK Key Length.

This exception is raised when a Content Encryption Key does not match the required lenght.

3.4 Registries

jwcrypto.jwe.**JWEHeaderRegistry**
Registry of valid header parameters

CHAPTER 4

JSON Web Token (JWT)

The jwt Module implements the JSON Web Token standard. A JSON Web Token is represented by a JWT object, related utility classes and functions are availbale in this module too.

4.1 Classes

class jwcrypto.jwt.**JWT**(*header=None*, *claims=None*, *jwt=None*, *key=None*, *algs=None*, *default_claims=None*, *check_claims=None*)

Bases: object

JSON Web token object

This object represent a generic token.

Creates a JWT object.

Parameters

- **header** – A dict or a JSON string with the JWT Header data.
- **claims** – A dict or a string withthe JWT Claims data.
- **jwt** – a 'raw' JWT token
- **key** – A (*jwcrypto.jwk.JWK*) key to deserialize the token. A (jwcrypt.jwk.JWKSet) can also be used.
- **algs** – An optional list of allowed algorithms
- **default_claims** – An optional dict with default values for registred claims. A None value for NumericDate type claims will cause generation according to system time. Only the values fro RFC 7519 - 4.1 are evaluated.
- **check_claims** – An optional dict of claims that must be present in the token, if the value is not None the claim must match exactly.

Note: either the header,claims or jwt,key parameters should be provided as a deserialization operation (which occurs if the jwt is provided will wipe any header os claim provided by setting those obtained from the deserialization of the jwt token.

Note: if check_claims is not provided the 'exp' and 'nbf' claims are checked if they are set on the token but not enforced if not set. Any other RFC 7519 registered claims are checked only for format conformance.

deserialize(*jwt*, *key=None*)

Deserialize a JWT token.

NOTE: Destroys any current status and tries to import the raw token provided.

Parameters

- **jwt** – a 'raw' JWT token.
- **key** – A (*jwcrypto.jwk.JWK*) verification or decryption key, or a (jwcrypt.jwk.JWKSet) that contains a key indexed by the 'kid' header.

make_encrypted_token(*key*)
Encrypts the payload.

Creates a JWE token with the header as the JWE protected header and the claims as the plaintext. See (*jwcrypto.jwe.JWE*) for details on the exceptions that may be reaised.

Parameters **key** – A (*jwcrypto.jwk.JWK*) key.

make_signed_token(*key*)
Signs the payload.

Creates a JWS token with the header as the JWS protected header and the claims as the payload. See (*jwcrypto.jws.JWS*) for details on the exceptions that may be reaised.

Parameters **key** – A (*jwcrypto.jwk.JWK*) key.

serialize(*compact=True*)
Serializes the object into a JWS token.

Parameters **compact (boolean)** – must be True.

Note: the compact parameter is provided for general compatibility with the serialize() functions of *jwcrypto.jws.JWS* and *jwcrypto.jwe.JWE* so that these objects can all be used interchangeably. However the only valid JWT representtion is the compact representation.

4.2 Examples

Create a symmetric key::

```
>>> from jwcrypto import jwt, jwk
>>> key = jwk.JWK(generate='oct', size=256)
>>> key.export()
'{"k":"Wal4ZHCBsml0Al_Y8faoNTKsXCkw8eefKXYFuwTBOpA","kty":"oct"}'
```

Create a signed token with the generated key::

```
>>> Token = jwt.JWT(header={"alg": "HS256"},
                    claims={"info": "I'm a signed token"})
>>> Token.make_signed_token(key)
>>> Token.serialize()
u'eyJhbGciOiJIUzI1NiJ9.eyJpbmZvIjoiSSdtIGEgc2lnbmVkIHRva2VuIn0.rjnRMAKcaRamEHnENhg0_Fqv7Obo-30U4
```

Further encrypt the token with the same key::

```
>>> Etoken = jwt.JWT(header={"alg": "A256KW", "enc": "A256CBC-HS512"},
                     claims=Token.serialize())
>>> Etoken.make_encrypted_token(key)
>>> Etoken.serialize()
u'eyJhbGciOiJBMjU2S1ciLCJlbmMiOiJBMjU2Q0JDLUhTNTEyIn0.ST5RmjqDLj696xo7YFTFuKUhcd3naCrm6yMjBM3cqW
```

Now decrypt and verify::

```
>>> from jwcrypto import jwt, jwk
>>> k = {"k": "Wal4ZHCBsml0Al_Y8faoNTKsXCkw8eefKXYFuwTBOpA", "kty": "oct"}
>>> key = jwk.JWK(**k)
>>> e = u'eyJhbGciOiJBMjU2S1ciLCJlbmMiOiJBMjU2Q0JDLUhTNTEyIn0.ST5RmjqDLj696xo7YFTFuKUhcd3naCrm6y
>>> ET = jwt.JWT(key=key, jwt=e)
>>> ST = jwt.JWT(key=key, jwt=ET.claims)
>>> ST.claims
u'{"info":"I\'m a signed token"}'
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search