
JWCrypto Documentation

Release 0.1.0

JWCrypto Contributors

December 24, 2015

1	JSON Web Key (JWK)	3
1.1	Classes	3
1.2	Exceptions	4
1.3	Registries	5
2	JSON Web Signature (JWS)	7
2.1	Classes	7
2.2	Variables	9
2.3	Exceptions	9
2.4	Registries	9
3	JSON Web Encryption (JWE)	11
3.1	Classes	11
3.2	Variables	12
3.3	Exceptions	12
3.4	Registries	13
4	JSON Web Token (JWT)	15
4.1	Classes	15
5	Indices and tables	17

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:

JSON Web Key (JWK)

The `jwt` Module implements the [JSON Web Key](#) standard. A JSON Web Key is represented by a JWK object, related utility classes and functions are available in this module too.

1.1 Classes

class `jwtcrypto.jwt.JWK(**kwargs)`
Bases: `object`

JSON Web Key object

This object represents a Key. It must be instantiated by using the standard defined key/value pairs as arguments 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.

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 strength options (type specific).

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)

Raises

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

export()

Exports the key in the standard JSON format

export_public()

Exports the public key in the standard JSON format

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 operation. 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 available 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.

key_id

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

key_type

The Key type

class `jwcrypto.jwk.JWKSet`

Bases: `set`

A set of JWK objects.

Inherits for the standard ‘set’ builtin type.

add (*elem*)

Adds a JWK object to the set

Parameters *elem* – the JWK object to add.

Raises **TypeError** if the object is not a JWK.

export ()

Exports the set using the standard JSON format

get_key (*kid*)

Gets a key from the set. :param kid: the ‘kid’ key identifier.

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`

Registry 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

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.
- **protected** – 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 invalid 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 be serialized with the compact representation and *compact* 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 performs 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', 'ES512']
```

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

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 available 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 represents 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 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 key is not a JWK object.
- **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.

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 (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 invalid 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 be 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 executed 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 specified 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 length required by the specified 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 length.

3.4 Registries

`jwcrypto.jwe.JWEHeaderRegistry`

Registry of valid header parameters

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 available in this module too.

4.1 Classes

class `jwtcrypto.jwt.JWT` (*header=None, claims=None, jwt=None, key=None, algs=None*)

Bases: `object`

JSON Web token object

This object represents 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 with the JWT Claims data.
- **jwt** – a 'raw' JWT token
- **key** – A (`jwtcrypto.jwk.JWK`) key to deserialize the token.
- **algs** – An optional list of allowed algorithms

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 or claim provided by setting those obtained from the deserialization of the jwt token.

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 (`jwtcrypto.jwk.JWK`) verification or decryption key.

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 (`jwtcrypto.jwe.JWE`) for details on the exceptions that may be raised.

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 raised.

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 representation is the compact representation.

Indices and tables

- `genindex`
- `modindex`
- `search`

A

`add()` (jwcrypto.jwk.JWKSet method), 4
`add_recipient()` (jwcrypto.jwe.JWE method), 11
`add_signature()` (jwcrypto.jws.JWS method), 7
`allowed_algs` (jwcrypto.jwe.JWE attribute), 12
`allowed_algs` (jwcrypto.jws.JWS attribute), 8

D

`decrypt()` (jwcrypto.jwe.JWE method), 11
`default_allowed_algs` (in module jwcrypto.jwe), 12
`default_allowed_algs` (in module jwcrypto.jws), 9
`deserialize()` (jwcrypto.jwe.JWE method), 12
`deserialize()` (jwcrypto.jws.JWS method), 7
`deserialize()` (jwcrypto.jwt.JWT method), 15

E

`export()` (jwcrypto.jwk.JWK method), 3
`export()` (jwcrypto.jwk.JWKSet method), 4
`export_public()` (jwcrypto.jwk.JWK method), 3

G

`get_curve()` (jwcrypto.jwk.JWK method), 3
`get_key()` (jwcrypto.jwk.JWKSet method), 4
`get_op_key()` (jwcrypto.jwk.JWK method), 4

I

`InvalidCEKeyLength` (class in jwcrypto.jwe), 13
`InvalidJWEData` (class in jwcrypto.jwe), 12
`InvalidJWEKeyLength` (class in jwcrypto.jwe), 13
`InvalidJWEKeyType` (class in jwcrypto.jwe), 13
`InvalidJWEOperation` (class in jwcrypto.jwe), 12
`InvalidJWKOperation` (class in jwcrypto.jwk), 5
`InvalidJWKType` (class in jwcrypto.jwk), 4
`InvalidJWKUsage` (class in jwcrypto.jwk), 5
`InvalidJWKValue` (class in jwcrypto.jwk), 4
`InvalidJWSObject` (class in jwcrypto.jws), 9
`InvalidJWSOperation` (class in jwcrypto.jws), 9
`InvalidJWSSignature` (class in jwcrypto.jws), 9

J

`JWE` (class in jwcrypto.jwe), 11
`JWEHeaderRegistry` (in module jwcrypto.jwe), 13
`JWK` (class in jwcrypto.jwk), 3
`JWKEllipticCurveRegistry` (in module jwcrypto.jwk), 5
`JWKOperationsRegistry` (in module jwcrypto.jwk), 5
`JWKParamsRegistry` (in module jwcrypto.jwk), 5
`JWKSet` (class in jwcrypto.jwk), 4
`JWKTypesRegistry` (in module jwcrypto.jwk), 5
`JWKUseRegistry` (in module jwcrypto.jwk), 5
`JWKValuesRegistry` (in module jwcrypto.jwk), 5
`JWS` (class in jwcrypto.jws), 7
`JWSCore` (class in jwcrypto.jws), 8
`JWSHeaderRegistry` (in module jwcrypto.jws), 9
`JWT` (class in jwcrypto.jwt), 15

K

`key_id` (jwcrypto.jwk.JWK attribute), 4
`key_type` (jwcrypto.jwk.JWK attribute), 4

M

`make_encrypted_token()` (jwcrypto.jwt.JWT method), 15
`make_signed_token()` (jwcrypto.jwt.JWT method), 16

S

`serialize()` (jwcrypto.jwe.JWE method), 12
`serialize()` (jwcrypto.jws.JWS method), 8
`serialize()` (jwcrypto.jwt.JWT method), 16
`sign()` (jwcrypto.jws.JWSCore method), 9

V

`verify()` (jwcrypto.jws.JWS method), 8
`verify()` (jwcrypto.jws.JWSCore method), 9