

---

# **JWCArypto Documentation**

***Release 1.5.4***

**JWCArypto Contributors**

**Feb 22, 2024**



# CONTENTS

<b>1</b>	<b>JSON Web Key (JWK)</b>	<b>3</b>
1.1	Classes	3
1.2	Exceptions	3
1.3	Registries	3
1.4	Examples	3
<b>2</b>	<b>JSON Web Signature (JWS)</b>	<b>5</b>
2.1	Classes	5
2.2	Variables	5
2.3	Exceptions	5
2.4	Registries	5
2.5	Examples	5
<b>3</b>	<b>JSON Web Encryption (JWE)</b>	<b>7</b>
3.1	Classes	7
3.2	Variables	7
3.3	Exceptions	7
3.4	Registries	7
3.5	Examples	7
<b>4</b>	<b>JSON Web Token (JWT)</b>	<b>9</b>
4.1	Classes	9
4.2	Variables	9
4.3	Examples	9
<b>5</b>	<b>Common</b>	<b>11</b>
5.1	Functions	11
5.2	Classes	11
5.3	Exceptions	11
<b>6</b>	<b>Indices and tables</b>	<b>13</b>
	<b>Index</b>	<b>15</b>



JWCArypto 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.

JWCArypto uses the [Cryptography](#) package for all the crypto functions.

Contents:



## 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 available in this module too.

### 1.1 Classes

### 1.2 Exceptions

### 1.3 Registries

### 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":"...", "kty":"oct"}'
```

Create a 2048bit RSA key pair::

```
>>> jwk.JWK.generate(kty='RSA', size=2048)
{"kid":"Missing Key ID", "thumbprint":"..."}
```

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

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

Import a P-256 Public Key::

```
>>> expkey = {"y": "VYlYwBfOTIICojCPfdUjnmkpN-g-lzZKxzjAoFmDRm8",
...          "x": "3mdE0rODWRju6qqU0lKw5oPYdNxBOmisFvJFH1vEu9Q",
```

(continues on next page)

(continued from previous page)

```
...         "crv":"P-256", "kty":"EC"}
>>> key = jwk.JWK(**expkey)
```

**Import a Key from a PEM file::**

```
>>> with open("public.pem", "rb") as pemfile:
...     key = jwk.JWK.from_pem(pemfile.read())
```



## 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

### 2.2 Variables

### 2.3 Exceptions

### 2.4 Registries

### 2.5 Examples

Sign a JWS token::

```
>>> from jwcrypto import jwk, jws
>>> from jwcrypto.common import json_encode
>>> key = jwk.JWK.generate(kty='oct', size=256)
>>> payload = "My Integrity protected message"
>>> jwstoken = jws.JWS(payload.encode('utf-8'))
>>> jwstoken.add_signature(key, None,
...                        json_encode({"alg": "HS256"}),
...                        json_encode({"kid": key.thumbprint()}))
>>> sig = jwstoken.serialize()
```

Verify a JWS token::

```
>>> jwstoken = jws.JWS()
>>> jwstoken.deserialize(sig)
>>> jwstoken.verify(key)
>>> payload = jwstoken.payload
```



## 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

### 3.2 Variables

### 3.3 Exceptions

### 3.4 Registries

### 3.5 Examples

#### 3.5.1 Symmetric keys

Encrypt a JWE token::

```
>>> from jwcrypto import jwk, jwe
>>> from jwcrypto.common import json_encode
>>> key = jwk.JWK.generate(kty='oct', size=256)
>>> payload = "My Encrypted message"
>>> jwetoken = jwe.JWE(payload.encode('utf-8'),
...                   json_encode({"alg": "A256KW",
...                               "enc": "A256CBC-HS512"}))
>>> jwetoken.add_recipient(key)
>>> enc = jwetoken.serialize()
```

Decrypt a JWE token::

```
>>> jwetoken = jwe.JWE()
>>> jwetoken.deserialize(enc)
>>> jwetoken.decrypt(key)
>>> payload = jwetoken.payload
```

### 3.5.2 Asymmetric keys

Encrypt a JWE token::

```
>>> from jwcrypto import jwk, jwe
>>> from jwcrypto.common import json_encode, json_decode
>>> public_key = jwk.JWK()
>>> private_key = jwk.JWK.generate(kty='RSA', size=2048)
>>> public_key.import_key(**json_decode(private_key.export_public()))
>>> payload = "My Encrypted message"
>>> protected_header = {
...     "alg": "RSA-OAEP-256",
...     "enc": "A256CBC-HS512",
...     "typ": "JWE",
...     "kid": public_key.thumbprint(),
... }
>>> jwetoken = jwe.JWE(payload.encode('utf-8'),
...                     recipient=public_key,
...                     protected=protected_header)
>>> enc = jwetoken.serialize()
```

Decrypt a JWE token::

```
>>> jwetoken = jwe.JWE()
>>> jwetoken.deserialize(enc, key=private_key)
>>> payload = jwetoken.payload
```

## 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

### 4.2 Variables

### 4.3 Examples

Create a symmetric key::

```
>>> from jwcrypto import jwt, jwk
>>> key = jwk.JWK(generate='oct', size=256)
>>> key.export()
'{"k":"...", "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()
'eyJhbGciOiJIUzI1NiJ9.eyJpbnVzIjoSSdtIGEgc2lnbmVkaHRva2VuIn0...''
```

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()
'eyJhbGciOiJBMjU2S1ciLCJlbmMiOiJBMjU2Q0JDLUhTNTYIn0...''
```

Now decrypt and verify::

```
>>> from jwcrypto import jwt, jwk
>>> k = {"k": "Wal4ZHCBSml0Al_Y8faoNTKsXCkw8eefKXYFuwTB0pA", "kty": "oct"}
>>> key = jwk.JWK(**k)
>>> e = 'eyJhbGciOiJBMjU2S1ciLCJlbmMiOiJBMjU2Q0JDLUhTNTYIn0.'
```

(continues on next page)

(continued from previous page)

```
↪ ST5RmjQDLj696xo7YFTFuKUhd3naCrm6yMjBM3cqWiFD6U8j2JIsbclsF7ryNg8Ktmt1kQJRKavV6DaTl1T840tP3sIs1qz_
↪ wSxVhZH5GyzbJnPBAUMdzQ.6uiVYwrRBzAm7Uge9rEUjExPWGbgerF177A7tMuQurJAqBhgk3_
↪ 5vee5DRH84kHSapFOxcEuDdMBEQLI7V2E0F57-d01TFStHzwtgtSmeZRQ6JSIL5XlgJouwHfSxn9Z_
↪ TGl5xxq4TksORHED1vnRA.5jPyPWanJVql0ohApEbHmxi3JHp1MXbmVqe2_dVd8FI '
>>> ET = jwt.JWT(key=key, jwt=e, expected_type="JWE")
>>> ST = jwt.JWT(key=key, jwt=ET.claims)
>>> ST.claims
{'"info": "I\'m a signed token"}'
```

## 5.1 Functions

```
jwcrypto.common.base64url_encode(payload)
```

```
jwcrypto.common.base64url_decode(payload)
```

```
jwcrypto.common.json_encode(string)
```

```
jwcrypto.common.json_decode(string)
```

## 5.2 Classes

```
class jwcrypto.common.JWSEHeaderRegistry(init_registry=None)
```

## 5.3 Exceptions

```
class jwcrypto.common.JWException
```

```
class jwcrypto.common.InvalidJWAAlgorithm(message=None)
```

Bases: *JWException*

```
class jwcrypto.common.InvalidCEKeyLength(expected, obtained)
```

Bases: *JWException*

Invalid CEK Key Length.

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

```
class jwcrypto.common.InvalidJWEOperation(message=None, exception=None)
```

Bases: *JWException*

Invalid JWS Object.

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

```
class jwcrypto.common.InvalidJWEKeyType(expected, obtained)
```

Bases: *JWException*

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.common.InvalidJWEKeyLength(*expected, obtained*)

Bases: *JWException*

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.common.InvalidJWSERegOperation(*message=None, exception=None*)

Bases: *JWException*

Invalid JWSE Header Registry Operation.

This exception is raised when there is an error in trying to add a JW Signature or Encryption header to the Registry.

**class** jwcrypto.common.JWKeyNotFound(*message=None*)

Bases: *JWException*

The key needed to complete the operation was not found.

This exception is raised when a JWKeySet is used to perform some operation and the key required to successfully complete the operation is not found.

Note: In the examples, random or generated output values are replaced with ‘...’ to allow for doctesting. Where possible the immutable part of a token has been preserved, and only the variable part replaced with ‘...’



## INDICES AND TABLES

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



## INDEX

### B

`base64url_decode()` (*in module `jwtcrypto.common`*), 11

`base64url_encode()` (*in module `jwtcrypto.common`*), 11

### I

`InvalidCEKeyLength` (*class in `jwtcrypto.common`*), 11

`InvalidJWAAlgorithm` (*class in `jwtcrypto.common`*), 11

`InvalidJWEKeyLength` (*class in `jwtcrypto.common`*), 12

`InvalidJWEKeyType` (*class in `jwtcrypto.common`*), 11

`InvalidJWEOperation` (*class in `jwtcrypto.common`*), 11

`InvalidJWSERegOperation` (*class in `jwtcrypto.common`*), 12

### J

`json_decode()` (*in module `jwtcrypto.common`*), 11

`json_encode()` (*in module `jwtcrypto.common`*), 11

`JWException` (*class in `jwtcrypto.common`*), 11

`JWKeyNotFound` (*class in `jwtcrypto.common`*), 12

`JWSEHeaderRegistry` (*class in `jwtcrypto.common`*), 11