# python-jose Documentation

***Release 0.2.0***

**Michael Davis**

June 27, 2017

Contents

A JOSE implementation in Python

The JavaScript Object Signing and Encryption (JOSE) technologies - JSON Web Signature (JWS), JSON Web Encryption (JWE), JSON Web Key (JWK), and JSON Web Algorithms (JWA) - collectively can be used to encrypt and/or sign content using a variety of algorithms. While the full set of permutations is extremely large, and might be daunting to some, it is expected that most applications will only use a small set of algorithms to meet their needs.

CHAPTER 1

# Contents

## JSON Web Signature

JSON Web Signatures (JWS) are used to digitally sign a JSON encoded object and represent it as a compact URL-safe string.

### Supported Algorithms

The following algorithms are currently supported.

| Algorithm Value | Digital Signature or MAC Algorithm |
|---|---|
| HS256 | HMAC using SHA-256 hash algorithm |
| HS384 | HMAC using SHA-384 hash algorithm |
| HS512 | HMAC using SHA-512 hash algorithm |
| RS256 | RSASSA using SHA-256 hash algorithm |
| RS384 | RSASSA using SHA-384 hash algorithm |
| RS512 | RSASSA using SHA-512 hash algorithm |
| ES256 | ECDSA using SHA-256 hash algorithm |
| ES384 | ECDSA using SHA-384 hash algorithm |
| ES512 | ECDSA using SHA-512 hash algorithm |

### Examples

#### Signing tokens

```
>>> from jose import jws
>>> signed = jws.sign({'a': 'b'}, 'secret', algorithm='HS256')
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhIjoiYiJ9.jiMyrsmD8AoHWeQgmxZ5yq8z0lXS67_QGs52AzC8Ru8'
```

#### Verifying token signatures

```
>>> jws.verify(signed, 'secret', algorithms=['HS256'])
{'a': 'b'}
```

# JSON Web Token

JSON Web Tokens (JWT) are a JWS with a set of reserved claims to be used in a standardized manner.

## JWT Reserved Claims

| Claim | Name | Format | Usage |
|---|---|---|---|
| 'exp' | Expiration | int | The time after which the token is invalid. |
| 'nbf' | Not Before | int | The time before which the token is invalid. |
| 'iss' | Issuer | str | The principal that issued the JWT. |
| 'aud' | Audience | str or list(str) | The recipient that the JWT is intended for. |
| 'iat' | Issued At | int | The time at which the JWT was issued. |

# JSON Web Key

JSON Web Keys (JWK) are a JSON data structure representing a cryptographic key.

## Examples

### Verifying token signatures

```
>>> from jose import jwk
>>> from jose.utils import base64url_decode
>>>
>>> token = "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LTRkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9.SXTigJ
>>> hmac_key = {
    "kty": "oct",
    "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037",
    "use": "sig",
    "alg": "HS256",
    "k": "hJtXIZ2uSN5kbQfbtTNWbpdmhkV8FJG-Onbc6mxCcYg"
}
>>>
>>> key = jwk.construct(hmac_key)
>>>
>>> message, encoded_sig = token.rsplit('.', 1)
>>> decoded_sig = base64url_decode(encoded_sig)
>>> key.verify(message, decoded_sig)
```

## Note

python-jose requires the use of public keys, as opposed to X.509 certificates. If you have an X.509 certificate that you would like to convert to a public key that python-jose can consume, you can do so with openssl.

```
> openssl x509 -pubkey -noout < cert.pem
```

CHAPTER 2

# APIs

## JWS API

jose.jws.**get_unverified_claims**(*token*)
Returns the decoded claims without verification of any kind.

**Parameters** **token** (*str*) – A signed JWS to decode the headers from.

**Returns** The str representation of the token claims.

**Return type** str

**Raises** JWSError – If there is an exception decoding the token.

jose.jws.**get_unverified_header**(*token*)
Returns the decoded headers without verification of any kind.

**Parameters** **token** (*str*) – A signed JWS to decode the headers from.

**Returns** The dict representation of the token headers.

**Return type** dict

**Raises** JWSError – If there is an exception decoding the token.

jose.jws.**get_unverified_headers**(*token*)
Returns the decoded headers without verification of any kind.

This is simply a wrapper of get_unverified_header() for backwards compatibility.

**Parameters** **token** (*str*) – A signed JWS to decode the headers from.

**Returns** The dict representation of the token headers.

**Return type** dict

**Raises** JWSError – If there is an exception decoding the token.

jose.jws.**sign**(*payload*, *key*, *headers=None*, *algorithm='HS256'*)
Signs a claims set and returns a JWS string.

**Parameters**

- **payload** (*str*) – A string to sign
- **key** (*str*) – The key to use for signing the claim set
- **headers** (*dict, optional*) – A set of headers that will be added to the default headers. Any headers that are added as additional headers will override the default headers.

- **algorithm** (*str, optional*) – The algorithm to use for signing the the claims. Defaults to HS256.

**Returns** The string representation of the header, claims, and signature.

**Return type** str

**Raises** `JWSError` – If there is an error signing the token.

#### Examples

```
>>> jws.sign({'a': 'b'}, 'secret', algorithm='HS256')
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhIjoiYiJ9.jiMyrsmD8AoHWeQgmxZ5yq8z0lXS67_QGs52AzC8Ru8'
```

`jose.jws.`**`verify`**(*token*, *key*, *algorithms*, *verify=True*)

Verifies a JWS string's signature.

**Parameters**

- **token** (*str*) – A signed JWS to be verified.
- **key** (*str*) – A key to attempt to verify the payload with.
- **algorithms** (*str or list*) – Valid algorithms that should be used to verify the JWS.

**Returns** The str representation of the payload, assuming the signature is valid.

**Return type** str

**Raises** `JWSError` – If there is an exception verifying a token.

#### Examples

```
>>> token = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhIjoiYiJ9.jiMyrsmD8AoHWeQgmxZ5yq8z0lXS67_QG
>>> jws.verify(token, 'secret', algorithms='HS256')
```

## JWT API

`jose.jwt.`**`decode`**(*token*, *key*, *algorithms=None*, *options=None*, *audience=None*, *issuer=None*, *subject=None*, *access_token=None*)

Verifies a JWT string's signature and validates reserved claims.

**Parameters**

- **token** (*str*) – A signed JWS to be verified.
- **key** (*str*) – A key to attempt to verify the payload with.
- **algorithms** (*str or list*) – Valid algorithms that should be used to verify the JWS.
- **audience** (*str*) – The intended audience of the token. If the "aud" claim is included in the claim set, then the audience must be included and must equal the provided claim.
- **issuer** (*str or iterable*) – Acceptable value(s) for the issuer of the token. If the "iss" claim is included in the claim set, then the issuer must be given and the claim in the token must be among the acceptable values.
- **subject** (*str*) – The subject of the token. If the "sub" claim is included in the claim set, then the subject must be included and must equal the provided claim.

- **access_token** (*str*) – An access token returned alongside the id_token during the authorization grant flow. If the "at_hash" claim is included in the claim set, then the access_token must be included, and it must match the "at_hash" claim.
- **options** (*dict*) – A dictionary of options for skipping validation steps.

  **defaults = {** 'verify_signature': True, 'verify_aud': True, 'verify_iat': True, 'verify_exp': True, 'verify_nbf': True, 'verify_iss': True, 'verify_sub': True, 'verify_jti': True, 'leeway': 0,

  }

**Returns** The dict representation of the claims set, assuming the signature is valid and all requested data validation passes.

**Return type** dict

**Raises** `JWTError` – If the signature is invalid in any way.

#### Examples

```
>>> payload = 'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhIjoiYiJ9.jiMyrsmD8AoHWeQgmxZ5yq8z0lXS67_
>>> jwt.decode(payload, 'secret', algorithms='HS256')
```

`jose.jwt.`**`encode`**(*claims*, *key*, *algorithm='HS256'*, *headers=None*, *access_token=None*)

Encodes a claims set and returns a JWT string.

JWTs are JWS signed objects with a few reserved claims.

**Parameters**

- **claims** (*dict*) – A claims set to sign
- **key** (*str*) – The key to use for signing the claim set
- **algorithm** (*str, optional*) – The algorithm to use for signing the the claims. Defaults to HS256.
- **headers** (*dict, optional*) – A set of headers that will be added to the default headers. Any headers that are added as additional headers will override the default headers.
- **access_token** (*str, optional*) – If present, the 'at_hash' claim will be calculated and added to the claims present in the 'claims' parameter.

**Returns** The string representation of the header, claims, and signature.

**Return type** str

**Raises** `JWTError` – If there is an error encoding the claims.

#### Examples

```
>>> jwt.encode({'a': 'b'}, 'secret', algorithm='HS256')
'eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJhIjoiYiJ9.jiMyrsmD8AoHWeQgmxZ5yq8z0lXS67_QGs52AzC8Ru8'
```

`jose.jwt.`**`get_unverified_claims`**(*token*)

Returns the decoded claims without verification of any kind.

**Parameters** **token** (*str*) – A signed JWT to decode the headers from.

**Returns** The dict representation of the token claims.

**Return type** dict

**Raises** JWTError – If there is an exception decoding the token.

jose.jwt.**get_unverified_header**(*token*)
Returns the decoded headers without verification of any kind.

**Parameters** **token** (*str*) – A signed JWT to decode the headers from.

**Returns** The dict representation of the token headers.

**Return type** dict

**Raises** JWTError – If there is an exception decoding the token.

jose.jwt.**get_unverified_headers**(*token*)
Returns the decoded headers without verification of any kind.

This is simply a wrapper of get_unverified_header() for backwards compatibility.

**Parameters** **token** (*str*) – A signed JWT to decode the headers from.

**Returns** The dict representation of the token headers.

**Return type** dict

**Raises** JWTError – If there is an exception decoding the token.

## JWK API

**class** jose.jwk.**HMACKey**(*key*, *algorithm*)
Performs signing and verification operations using HMAC and the specified hash function.

**SHA256**()
Returns a sha256 hash object; optionally initialized with a string

**SHA384**()
Returns a sha384 hash object; optionally initialized with a string

**SHA512**()
Returns a sha512 hash object; optionally initialized with a string

jose.jwk.**construct**(*key_data*, *algorithm=None*)
Construct a Key object for the given algorithm with the given key_data.

CHAPTER 3

# Principles

This is a JOSE implementation that is fully compatible with Google App Engine which requires the use of the PyCrypto library.

CHAPTER 4

# Thanks

This library was originally based heavily on the work of the guys over at PyJWT.

## j

## C



## D



## E



## G



## H



## J



## S



## V