## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>3</td>
</tr>
<tr>
<td>2 JWK</td>
<td>5</td>
</tr>
<tr>
<td>2.1 JWK format</td>
<td>5</td>
</tr>
<tr>
<td>3 JWS</td>
<td>7</td>
</tr>
<tr>
<td>3.1 Definition</td>
<td>7</td>
</tr>
<tr>
<td>3.2 API</td>
<td>7</td>
</tr>
<tr>
<td>3.3 Example</td>
<td>7</td>
</tr>
<tr>
<td>3.4 Algorithm support</td>
<td>7</td>
</tr>
<tr>
<td>4 JWE</td>
<td>9</td>
</tr>
<tr>
<td>4.1 Algorithm support</td>
<td>9</td>
</tr>
<tr>
<td>5 Serialization</td>
<td>11</td>
</tr>
<tr>
<td>6 JWT</td>
<td>13</td>
</tr>
<tr>
<td>7 Errors</td>
<td>15</td>
</tr>
<tr>
<td>8 References</td>
<td>17</td>
</tr>
</tbody>
</table>
Contents

- Javascript Object Signing and Encryption (JOSE)
  - Overview
  - JWK
    * JWK format
  - JWS
    * Definition
    * API
    * Example
    * Algorithm support
  - JWE
    * Algorithm support
      - CEK Encryption (alg)
      - Claims Encryption (enc)
  - Serialization
  - JWT
  - Errors
- References
JOSE\(^1\) is a framework intended to provide a method to securely transfer claims (such as authorization information) between parties. The JOSE framework provides a collection of specifications to serve this purpose. A JSON Web Token (JWT)\(^2\) contains claims that can be used to allow a system to apply access control to resources it owns. One potential use case of the JWT is as the means of authentication and authorization for a system that exposes resources through an OAuth 2.0 model\(^5\).

Claims are a set of key/value pairs that provide a target system with sufficient information about the given client to apply the appropriate level of access control to resources under its ownership. Claim names are split into three classes: Registered (IANA), Public and Private. Further details about claims can be found in section 4 of the JWT specification.

JWTs can be represented as either JSON Web Signature (JWS)\(^3\) or a JSON Web Encryption (JWE)\(^4\) objects. Claims within a JWS can be read as they are simply base64-encoded (but carry with them a signature for authentication). Claims in a JWE on the other hand, are encrypted and as such, are entirely opaque to clients using them as their means of authentication and authorization.

---

\(^1\) JOSE: JSON Object Signing and Encryption  
https://datatracker.ietf.org/wg/jose/charter/

\(^2\) JWT: JSON Web Tokens  

\(^3\) JWT Authorization Grants  

\(^4\) JWS: JSON Web Signing  

\(^5\) JWE: JSON Web Encryption  
A JSON Web Key (JWK) is a JSON data structure that represents a cryptographic key. Using a JWK rather than one or more parameters allows for a generalized key as input that can be applied to a number of different algorithms that may expect a different number of inputs. All JWE and JWS operations expect a JWK rather than inflexible function parameters.

### 2.1 JWK format

```json
jwk = {'k': <password>}
```

Currently, the only key/value pair that’s required throughout the JWE and JWS flows is ‘k’, indicating the key, or password.

**Note:** The password must match algorithm requirements (i.e. a key used with an RSA algorithm must be at least 2048 bytes and be a valid private or public key, depending on the cryptographic operation). Other fields may be required in future releases.

---

3.1 Definition

A deserialized JWS is represented as a `namedtuple` with the following definition:

3.2 API

3.3 Example

```python
import jose

claims = {
    'iss': 'http://www.example.com',
    'exp': int(time()) + 3600,
    'sub': 42,
}

jwk = {'k': 'password'}

jws = jose.sign(claims, jwk, alg='HS256')
# JWS(header='eyJhbGciOiAiSFMyNTYifQ',
# payload='eyJpc3MiOiAiaHR0cDovL3d3dy5leGFtcGxlLmNvbSIsICJzdWIiOiA0MiwgImV4cCI6IDEzOTU2NzQ0Mjd9',
# signature='WYApAiwiKd-eDC1A1fg7XFrnfHzUTgrmdRQY4M19Vr8')
# issue the compact serialized version to the clients. this is what will be
# transported along with requests to target systems.

jwt = jose.serialize_compact(jws)
# 'eyJhbGciOiAiSFMyNTYifQ.eyJpc3MiOiAiaHR0cDovL3d3dy5leGFtcGxlLmNvbSIsICJzdWIiOiA0MiwgImV4cCI6IDEzOTU2NzQ0Mjd9.WYApAiwiKd-eDC1A1fg7XFrnfHzUTgrmdRQY4M19Vr8'

jose.verify(jose.deserialize_compact(jwt), jwk, 'HS256')
# JWT(header={u'alg': u'HS256'}, claims={u'iss': u'http://www.example.com', u'sub': 42, u'exp': 1395674427})
```

3.4 Algorithm support

<table>
<thead>
<tr>
<th>Symmetric</th>
<th>HS256, HS384, HS512</th>
</tr>
</thead>
<tbody>
<tr>
<td>Asymmetric</td>
<td>RS256, RS384, RS512</td>
</tr>
</tbody>
</table>
import jose
from time import time
from Crypto.PublicKey import RSA

# key for demonstration purposes
key = RSA.generate(2048)

claims = {
    'iss': 'http://www.example.com',
    'exp': int(time()) + 3600,
    'sub': 42,
}

# encrypt claims using the public key
pub_jwk = {'k': key.publickey().exportKey('PEM')}

jwe = jose.encrypt(claims, pub_jwk)

# issue the compact serialized version to the clients. this is what will be
# transported along with requests to target systems.
jwt = jose.serialize_compact(jwe)

# decrypt on the other end using the private key
priv_jwk = {'k': key.exportKey('PEM')}

jwt = jose.decrypt(jose.deserialize_compact(jwt), priv_jwk)

4.1 Algorithm support
Note: There are two different encryption algorithms employed to fully encrypt a JWE: Encryption of the Content Encryption Key (CEK) and encryption of the JWT claims. The encryption algorithm used to encrypt the CEK is set through the \texttt{alg} parameter of \texttt{encrypt()} and the claims encryption is defined by the \texttt{enc} parameter.

### 4.1.1 CEK Encryption (\texttt{alg})

<table>
<thead>
<tr>
<th>Symmetric</th>
<th>Asymmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>[None]</td>
<td>RSA-OAEP</td>
</tr>
</tbody>
</table>

### 4.1.2 Claims Encryption (\texttt{enc})

<table>
<thead>
<tr>
<th>Symmetric</th>
<th>Asymmetric</th>
</tr>
</thead>
<tbody>
<tr>
<td>A128CBC-HS256, A192CBC-HS256, A256CBC-HS512</td>
<td>[N/A]</td>
</tr>
</tbody>
</table>
CHAPTER 5

Serialization
A JWT is a *namedtuple* result produced by either decrypting or verifying a JWE or a JWS.
CHAPTER 7

Errors
References