LEARNING jwt

#jwt

Table of Contents

About

Chapter 1: Getting started with jwt

Remarks

A JSON Web Token (JWT) is a compact, URL-safe way of representing claims that can be exchanged between parties.

All JWTs consist of a **header** and **payload**, which are JSON hashes. These objects are stringified and Base64-encoded. The encoded header and payload are combined with a digital signature (JWS), and all three components are concatenated with "." (period).

Further Reading

- Use Cases and Requirements for JSON Object Signing and Encryption (RFC 7165)
- JSON Web Signature specification (RFC 7515)
- JSON Web Encryption specification (RFC 7516)
- JSON Web Key (RFC 7517)
- JSON Web Algorithms (RFC 7518)
- JSON Web Token specification (RFC 7519)
- IANA List of JSON Web Token Claims (RFC 7519 IANA list)
- Examples of Protecting Content Using JSON Object Signing and Encryption (RFC 7520)
- JSON Web Key (JWK) Thumbprint (RFC 7638)
- JSON Web Signature (JWS) Unencoded Payload Option (RFC 7797)

Examples

Unsigned JWT

An unsigned JWT has the header value `alg: none` and an empty JWS (signature) component:

```
eyJhbGciOiJub25lIn0
.eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ

.
```

The trailing dot indicates that the signature is empty.

Header

```
{
  "alg": "none"
}
```

Payload

```
{
  "iss": "joe",
  "exp": 1300819380,
  "http://example.com/is_root": true
}
```

Signed JWT (JWS)

A signed JWT includes a Base64 Url Safe encoded signature as the third component. The algorithm used to generate the signature is indicated in the header.

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9
.eyJzdWIiOiJKb2huIERvZSIsImFkbWluIjp0cnVlLCJpYXQiOjE0NzAzNTM5OTQsImV4cCI6MTQ3MDM1NzYyNywianRpIjoiNmU0MD

.7CfBdVP4uKsb0cogYepCvMLm8rcpjBYW1XZzA-a5e44
```

Header

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

This JWT was signed with the HMAC-SHA256 algorithm, hence `alg: HS256`.

Payload

```
{
  "sub": "John Doe",
  "admin": true,
  "iat": 1470353994,
  "exp": 1470357627,
  "jti": "6e404ba8-f885-4d5f-bfa2-e3f5a08380a4"
}
```

This JWT can be verified with the UTF-8 secret `notsosecret`.

JSON Web Encryption (JWE)

JSON Web Encryption (JWE) represents encrypted content using JavaScript Object Notation (JSON) based data structures. It defines a way to encrypt your claims data so that only intended receiver can read the information present in a token.

In the JWE JSON Serialization, a JWE is represented as a JSON object containing some or all of these eight members:

```
  "protected", with the value BASE64URL(UTF8(JWE Protected Header))
  "unprotected", with the value JWE Shared Unprotected Header
  "header", with the value JWE Per-Recipient Unprotected Header
  "encrypted_key", with the value BASE64URL(JWE Encrypted Key)
  "iv", with the value BASE64URL(JWE Initialization Vector)
  "ciphertext", with the value BASE64URL(JWE Ciphertext)
  "tag", with the value BASE64URL(JWE Authentication Tag)
  "aad", with the value BASE64URL(JWE AAD)
```

The six base64url-encoded result strings and the two unprotected JSON object values are represented as members within a JSON object.

Example JWE

The following example JWE Header declares that:

- the Content Encryption Key is encrypted to the recipient using the RSA-PKCS1_1.5 algorithm to produce the JWE Encrypted Key
- the Plaintext is encrypted using the AES-256-GCM algorithm to produce the JWE Ciphertext
- the specified 64-bit Initialization Vector with the base64url encoding __79_Pv6-fg was used
- the thumbprint of the X.509 certificate that corresponds to the key used to encrypt the JWE has the base64url encoding 7noOPq-hJ1_hCnvWh6IeYI2w9Q0.

```
{
 "alg":"RSA1_5",
 "enc":"A256GCM",
 "iv":"__79_Pv6-fg",
 "x5t":"7noOPq-hJ1_hCnvWh6IeYI2w9Q0"
}
```

Base64url encoding the bytes of the UTF-8 representation of the JWE Header yields this Encoded JWE Header value (with line breaks for display purposes only):

```
eyJhbGciOiJSU0ExXzUiLA0KICJlbmMiOiJBMjU2R0NNIiwNCiAiaXYiOiJfXzc5
X1B2Ni1mZyIsDQogIng1dCI6Ijdub09QcS1oSjFfaENudldoNkllWUkydzlRMCJ
```

Read JSON Web Encryption specification (RFC 7516) for more information

How to tell if you have a JWS or JWE?

From Section 9 of JSON Web Encryption specification (RFC 7516):

> The JOSE Header for a JWS can be distinguished from the JOSE Header for a JWE by examining the "alg" (algorithm) Header Parameter value. If the value represents a digital signature or MAC algorithm, or is the value "none", it is for a JWS; if it represents a Key Encryption, Key Wrapping, Direct Key Agreement, Key Agreement with Key Wrapping, or Direct Encryption algorithm, it is for a JWE. (Extracting the "alg" value to examine is straightforward when using the JWS Compact Serialization or the JWE Compact Serialization and may be more difficult when using the JWS JSON Serialization or the JWE JSON Serialization.)

And

> The JOSE Header for a JWS can also be distinguished from the JOSE Header for a JWE by determining whether an "enc" (encryption algorithm) member exists. If the "enc" member exists, it is a JWE; otherwise, it is a JWS.

JWS (signed)##

```
{
  "alg": "HS256"
}
```

JWE (encrypted)

```
{
  "alg":"RSA1_5",
  "enc":"A256GCM",
  "iv":"__79_Pv6-fg",
  "x5t":"7noOPq-hJ1_hCnvWh6IeYI2w9Q0"
}
```

What to store in a JWT

The JWT RFC stablish three classes of claims:

- **Registered claims** like `sub`, `iss`, `exp` or `nbf`
- **Public claims** with public names or names registered by IANA which contain values that should be unique like `email`, `address` or `phone_number`. See full list
- **Private claims** to use in your own context and values can collision

None of these claims are mandatory

A JWT is self-contained and should avoid use the server session providing the necessary data to perform the authentication (no need of server storage and database access). Therefore, `role` or `permissions` info can be included in private claims of JWT.

Registered Claims

The following Claim Names are registered in the IANA "JSON Web Token Claims" registry established by Section 10.1.

- `iss` (issuer): identifies the principal that issued the JWT.
- `sub` (subject): identifies the principal that is the subject of the JWT. Must be unique
- `aud` (audience): identifies the recipients that the JWT is intended for (array of strings/uri)
- `exp` (expiration time): identifies the expiration time (UTC Unix) after which you must no longer accept this token. It should be after the issued-at time.

- `nbf`(not before): identifies the UTC Unix time before which the JWT must not be accepted
- `iat` (issued at): identifies the UTC Unix time at which the JWT was issued
- `jti` (JWT ID): provides a unique identifier for the JWT.

Example

```
{
    "iss": "stackoverflow",
    "sub": "joe",
    "aud": ["all"],
    "iat": 1300819370,
    "exp": 1300819380,
    "jti": "3F2504E0-4F89-11D3-9A0C-0305E82C3301"
    "context": {
        "user": {
            "key": "joe",
            "displayName": "Joe Smith"
        },
        "roles":["admin","finaluser"]
    }
}
```

Read Getting started with jwt online: https://riptutorial.com/jwt/topic/5213/getting-started-with-jwt

Chapter 2: Invalidating Json Web Tokens

Remarks

There are several reason to invalidate a JWT token before its expiration time: account deleted/blocked/suspended, password or permissions changed, user logged out by admin.

JWT is self-contained, signed and stored outside of the server context, so revoking a token is not a simple action.

Examples

Remove the token from client storage

Remove the token from the client storage to avoid usage

Tokens are issued by the server and you can not force browsers to delete a cookie/localStorage or control how external clients are managing your tokens. Obviously **if attackers have stolen the token before logout** they still could use the token, therefore **are needed additional measures in server side** (see below for token blacklist strategy)

Cookies

You cannot force browsers to delete a cookie. The client can configure the browser in such a way that the cookie persists, even if it's expired. But the server can set the value to empty and include expires field to invalidate the cookie value.

```
Set-Cookie: token=deleted; path=/; expires=Thu, 01 Jan 1970 00:00:00 GMT
```

Delete 'token' with javascript

```
document.cookie = 'token=; Path=/; Expires=Thu, 01 Jan 1970 00:00:01 GMT;';
localStorage.removeItem('token')
sessionStorage.removeItem('token')
```

Token blacklist

Mark invalid tokens, store until their expiration time and check it in every request.

Blacklist breaks JWT statelessness because it requires maintaining the state. One of the benefits of JWT is no need server storage, so if you need to revoke tokens without waiting for the expiration, think also about the downside

Manage the blacklist

The blacklist can be easily managed in your own service/database. The storage size probably would not be large because it is only needed to store tokens that were between logout and expiry time.

Include the full token or just the unique ID `jti`. Set the `iat` (issued at) to remove old tokens.

To revoke all tokens after updating critical data on user (password, permissions, etc) set a new entry with `sub` and `iat` when `currentTime - maxExpiryTime < last iss`. The entry can be discarded when `currentTime - maxExpiryTime > lastModified` (no more non-expired tokens sent).

Rotate tokens

Set **expiration time short and rotate tokens**. Issue a new **access token** every few request. Use **refresh tokens** to allow your application to obtain new access tokens without needing to re-authenticate

Refresh and access tokens

- **access token**: Authorize access to a protected resource. Limited lifetime. Must be kept secret, security considerations are less strict due to their shorter life.
- **Refresh token**: Allows your application to obtain new access tokens without needing to re-authenticate. Long lifetime. Store in secure long-term storage

Usage recomendations:

- **Web applications**: refresh the access token before it expires, each time user open the application and at fixed intervals. Alternatively renew the access token when a user performs an action. If the user uses an expired access token, the session is considered inactive and a new access token is required. This new token can be obtained with a refresh token or requiring credentials
- **Mobile/Native applications**: Application login once and only once. Refresh token does not expire and can be exchanged for a valid JWT. Take in account special events like changing password

Other common techniques

- Allow change user unique ID if account is compromised with a new user&password login
- To invalidate tokens when user changes their password or permissions, sign the token with a hash of those fields. If any of these field change, any previous tokens automatically fail to verify. The downside is that it requires access to the database
- Change signature algorithm to revoke all current tokens in a major security issue

Read Invalidating Json Web Tokens online: https://riptutorial.com/jwt/topic/6224/invalidating-json-web-tokens

Chapter 3: Serializations

Examples

JWS Compact Serialization

The Compact Serialization is the most common serialization format and is designed to be used in a web context.

JWS are represented into a string that contains Base64 Url Safe encoded information seperated by an dot ".".

This mode does not support unprotected headers.

Line breaks added for readability

```
BASE64URL(UTF8(JWS Protected Header)) || '.' ||
BASE64URL(JWS Payload) || '.' ||
BASE64URL(JWS Signature)
```

Example

```
eyJhbGciOiJQUzM4NCIsImtpZCI6ImJpbGJvLmJhZ2dpbnNAaG9iYml0b24uZX
hhbXBsZSJ9
.
SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywgZ29pbmcgb3V0IH
lvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9hZCwgYW5kIGlmIHlvdSBk
b24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXigJlzIG5vIGtub3dpbmcgd2hlcm
UgeW91IG1pZ2h0IGJlIHN3ZXB0IG9mZiB0by4
.
cu22eBqkYDKgIlTpzDXGvaFfz6WGoz7fUDcfT0kkOy42miAh2qyBzk1xEsnk2I
pN6-tPid6VrklHkqsGqDqHCdP6O8TTB5dDDItllVo6_1OLPpcbUrhiUSMxbbXU
vdvWXzg-UD8biiReQFlfz28zGWVsdiNAUf8ZnyPEgVFn442ZdNqiVJRmBqrYRX
e8P_ijQ7p8Vdz0TTrxUeT3lm8d9shnr2lfJT8ImUjvAA2Xez2Mlp8cBE5awDzT
0qI0n6uiP1aCN_2_jLAeQTlqRHtfa64QQSUmFAAjVKPbByi7xho0uTOcbH510a
6GYmJUAfmWjwZ6oD4ifKo8DYM-X72Eaw
```

JWE Compact Serialization

The Compact Serialization is the most common serialization format and is designed to be used in a web context.

JWE are represented into a string that contains Base64 Url Safe encoded information seperated by an dot ".".

This mode does not support unprotected headers or AAD.

Line breaks added for readability

```
BASE64URL(UTF8(JWE Protected Header)) || '.' ||
BASE64URL(JWE Encrypted Key) || '.' ||
BASE64URL(JWE Initialization Vector) || '.' ||
BASE64URL(JWE Ciphertext) || '.' ||
BASE64URL(JWE Authentication Tag)
```

Example

```
eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InNhbXdpc2UuZ2FtZ2VlQGhvYmJpdG
9uLmV4YW1wbGUiLCJlbmMiOiJBMjU2R0NNIn0
.
rT99rwrBTbTI7IJM8fU3Eli7226HEB7IchCxNuh7lCiud48LxeolRdtFF4nzQi
beYOl5S_PJsAXZwSXtDePz9hk-BbtsTBqC2UsPOdwjC9NhNupNNu9uHIVftDyu
cvI6hvALeZ6OGnhNV4v1zx2k7O1D89mAzfw-_kT3tkuorpDU-CpBENfIHX1Q58
-Aad3FzMuo3Fn9buEP2yXakLXYa15BUXQsupM4A1GD4_H4Bd7V3u9h8Gkg8Bpx
KdUV9ScfJQTcYm6eJEBz3aSwIaK4T3-dwWpuBOhROQXBosJzS1asnuHtVMt2pK
IIfux5BC6huIvmY7kzV7W7aIUrpYm_3H4zYvyMeq5pGqFmW2k8zpO878TRlZx7
pZfPYDSXZyS0CfKKkMozT_qiCwZTSz4duYnt8hS4Z9sGthXn9uDqd6wycMagnQ
fOTs_lycTWmY-aqWVDKhjYNRf03NiwRtb5BE-tOdFwCASQj3uuAgPGrO2AWBe3
8UjQb0lvXn1SpyvYZ3WFc7WOJYaTa7A8DRn6MC6T-xDmMuxC0G7S2rscw5lQQU
06MvZTlFOt0UvfuKBa03cxA_nIBIhLMjY2kOTxQMmpDPTr6Cbo8aKaOnx6ASE5
Jx9paBpnNmOOKH35j_QlrQhDWUN6A2Gg8iFayJ69xDEdHAVCGRzN3woEI2ozDR
s
.
-nBoKLH0YkLZPSI9
.
o4k2cnGN8rSSw3IDo1YuySkqeS_t2m1GXklSgqBdpACm6UJuJowOHC5ytjqYgR
L-I-soPlwqMUf4UgRWWeaOGNw6vGW-xyM01lTYxrXfVzIIaRdhYtEMRBvBWbEw
P7ua1DRfvaOjgZv6Ifa3brcAM64d8p5lhhNcizPersuhw5f-pGYzseva-TUaL8
iWnctc-sSwy7SQmRkfhDjwbz0fz6kFovEgj64X1I5s7E6GLp5fnbYGLa1QUiML
7Cc2GxgvI7zqWo0YIEc7aCflLG1-8BboVWFdZKLK9vNoycrYHumwzKluLWEbSV
maPpOslY2n525DxDfWaVFUfKQxMF56vn4B9QMpWAbnypNimbM8zVOw
.
UCGiqJxhBI3IFVdPalHHvA
```

General JWS JSON Serialization Syntax

The JWS JSON Serialization represents digitally signed or MACed content as a JSON object. This representation is neither optimized for compactness nor URL-safe.

This syntax is optimized for more than one digital signature and/or MAC operation.

Line breaks added for readability

```
     {
      "payload":"<payload contents>",
      "signatures":[
       {"protected":"<integrity-protected header 1 contents>",
        "header":<non-integrity-protected header 1 contents>,
        "signature":"<signature 1 contents>"},
       ...
       {"protected":"<integrity-protected header N contents>",
        "header":<non-integrity-protected header N contents>,
        "signature":"<signature N contents>"}]
     }
```

Example

```
{
  "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
      Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
      ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
      gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
      ZiB0by4",
  "signatures": [
    {
      "protected": "eyJhbGciOiJSUzI1NiJ9",
      "header": {
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "MIsjqtVlOpa71KE-Mss8_Nq2YH4FGhiocsqrgi5Nvy
          G53uoimic1tcMdSg-qptrzZc7CG6Svw2Y13TDIqHzTUrL_lR2ZFc
          ryNFiHkSw129EghGpwkpxaTn_THJTCglNbADko1MZBCdwzJxwqZc
          -1RlpO2HibUYyXSwO97BSe0_evZKdjvvKSgsIqjytKSeAMbhMBdM
          ma622_BG5t4sdbuCHtFjp9iJmkio47AIwqkZV1aIZsv33uPUqBBC
          XbYoQJwt7mxPftHmNlGoOSMxR_3thmXTCm4US-xiNOyhbm8afKK6
          4jU6_TPtQHiJeQJxz9G3Tx-083B745_AfYOnlC9w"
    },
    {
      "header": {
        "alg": "ES512",
        "kid": "bilbo.baggins@hobbiton.example"
      },
      "signature": "ARcVLnaJJaUWG8fG-8t5BREVAuTY8n8YHjwDO1muhc
          dCoFZFFjfISu0Cdkn9Ybdlmi54ho0x924DUz8sK7ZXkhc7AFM8Ob
          LfTvNCrqcI3Jkl2U5IX3utNhODH6v7xgy1Qahsn0fyb4zSAkje8b
          AWz4vIfj5pCMYxxm4fgV3q7ZYhm5eD"
    },
    {
      "protected": "eyJhbGciOiJIUzI1NiIsImtpZCI6IjAxOGMwYWU1LT
          RkOWItNDcxYi1iZmQ2LWVlZjMxNGJjNzAzNyJ9",
      "signature": "s0h6KThzkfBBBkLspW1h84VsJZFTsPPqMDA7g1Md7p
          0"
    }
  ]
}
```

Flattened JWS JSON Serialization Syntax

As the General JWS JSON Serialization Syntax, the JWS JSON Serialization represents digitally signed or MACed content as a JSON object. This representation is neither optimized for compactness nor URL-safe.

The flattened syntax is optimized for the single digital signature or MAC case.

Line breaks added for readability

```
  {
   "payload":"<payload contents>",
   "protected":"<integrity-protected header contents>",
   "header":<non-integrity-protected header contents>,
   "signature":"<signature contents>"
```

```
    }
```

Example

```
  {
    "payload": "SXTigJlzIGEgZGFuZ2Vyb3VzIGJ1c2luZXNzLCBGcm9kbywg
        Z29pbmcgb3V0IHlvdXIgZG9vci4gWW91IHN0ZXAgb250byB0aGUgcm9h
        ZCwgYW5kIGlmIHlvdSBkb24ndCBrZWVwIHlvdXIgZmVldCwgdGhlcmXi
        gJlzIG5vIGtub3dpbmcgd2hlcmUgeW91IG1pZ2h0IGJlIHN3ZXB0IG9m
        ZiB0by4",
    "protected": "eyJhbGciOiJIUzI1NiJ9",
    "header": {
      "kid": "018c0ae5-4d9b-471b-bfd6-eef314bc7037"
    },
    "signature": "bWUSVaxorn7bEF1djytBd0kHv70Ly5pvbomzMWSOr20"
  }
```

General JWE JSON Serialization Syntax

The JWE JSON Serialization represents encrypted content as a JSON object. This representation is neither optimized for compactness nor URL safe.

This syntax is optimized for more than one recipient.

Line breaks added for readability

```
    {
     "protected":"<integrity-protected shared header contents>",
     "unprotected":<non-integrity-protected shared header contents>,
     "recipients":[
      {"header":<per-recipient unprotected header 1 contents>,
       "encrypted_key":"<encrypted key 1 contents>"},
      ...
      {"header":<per-recipient unprotected header N contents>,
       "encrypted_key":"<encrypted key N contents>"}],
     "aad":"<additional authenticated data contents>",
     "iv":"<initialization vector contents>",
     "ciphertext":"<ciphertext contents>",
     "tag":"<authentication tag contents>"
    }
```

Example

```
  {
    "recipients": [
      {
        "encrypted_key": "dYOD28kab0Vvf4ODgxVAJXgHcSZICSOp8M51zj
            wj4w6Y5G4XJQsNNIBiqyvUUAOcpL7S7-cFe7Pio7gV_Q06WmCSa-
            vhW6me4bWrBf7cHwEQJdXihidAYWVajJIaKMXMvFRMV6iDlRr076
            DFthg2_AV0_tSiV6xSEIFqt1xnYPpmP91tc5WJDOGb-wqjw0-b-S
            1laS11QVbuP78dQ7Fa0zAVzzjHX-xvyM2wxj_otxr9clN1LnZMbe
            YSrRicJK5xodvWgkpIdkMHo4LvdhRRvzoKzlic89jFWPlnBq_V4n
            5trGuExtp_-dbHcGlihqc_wGgho9fLMK8JOArYLcMDNQ",
        "header": {
          "alg": "RSA1_5",
          "kid": "frodo.baggins@hobbiton.example"
        }
      },
      {
        "encrypted_key": "ExInT0io9BqBMYF6-maw5tZlgoZXThD1zWKsHi
            xJuw_elY4gSSId_w",
        "header": {
          "alg": "ECDH-ES+A256KW",
          "kid": "peregrin.took@tuckborough.example",
          "epk": {
            "kty": "EC",
            "crv": "P-384",
            "x": "Uzdvk3pi5wKCRc1izp5_r0OjeqT-I68i8g2b8mva8diRhs
                E2xAn2DtMRb25Ma2CX",
            "y": "VDrRyFJh-Kwd1EjAgmj5Eo-CTHAZ53MC7PjjpLioy3ylEj
                I1pOMbw91fzZ84pbfm"
          }
        }
      },
      {
        "encrypted_key": "a7CclAejo_7JSuPB8zeagxXRam8dwCfmkt9-Wy
            TpS1E",
        "header": {
          "alg": "A256GCMKW",
          "kid": "18ec08e1-bfa9-4d95-b205-2b4dd1d4321d",
          "tag": "59Nqh1LlYtVIhfD3pgRGvw",
          "iv": "AvpeoPZ9Ncn9mkBn"
        }
      }
    ],
    "unprotected": {
      "cty": "text/plain"
    },
    "protected": "eyJlbmMiOiJBMTI4Q0JDLUhTMjU2In0",
    "iv": "VgEIHY20EnzUtZFl2RpB1g",
    "ciphertext": "ajm2Q-OpPXCr7-MHXicknb1lsxLdXxK_yLds0KuhJzfWK
        04SjdxQeSw2L9mu3a_k1C55kCQ_3xlkcVKC5yr__Is48VOoK0k63_QRM
        9tBURMFqLByJ8vOYQX0oJW4VUHJLmGhF-tVQWB7Kz8mr8zeE7txF0MSa
        P6ga7-siYxStR7_G07Thd1jh-zGT0wxM5g-VRORtq0K6AXpLlwEqRp7p
        kt2zRM0ZAXqSpe1O6FJ7FHLDyEFnD-zDIZukLpCbzhzMDLLw2-8I14FQ
        rgi-iEuzHgIJFIJn2wh9Tj0cg_kOZy9BqMRZbmYXMY9YQjorZ_P_JYG3
        ARAIF3OjDNqpdYe-K_5Q5crGJSDNyij_ygEiItR5jssQVH2ofDQdLCht
        azE",
    "tag": "BESYyFN7T09KY7i8zKs5_g"
  }
```

Flattened JWE JSON Serialization Syntax

The flattened JWE JSON Serialization syntax is based upon the general syntax, but flattens it, optimizing it for the single-recipient case.

Line breaks added for readability

```
    {
     "protected":"<integrity-protected header contents>",
     "unprotected":<non-integrity-protected header contents>,
     "header":<more non-integrity-protected header contents>,
     "encrypted_key":"<encrypted key contents>",
     "aad":"<additional authenticated data contents>",
     "iv":"<initialization vector contents>",
     "ciphertext":"<ciphertext contents>",
     "tag":"<authentication tag contents>"
    }
```

Example

```
 {
   "protected": "eyJhbGciOiJBMTI4S1ciLCJraWQiOiI4MWIyMDk2NS04Mz
       MyLTQzZDktYTQ2OC04MjE2MGFkOTFhYzgiLCJlbmMiOiJBMTI4R0NNIn
       0",
   "encrypted_key": "4YiiQ_ZzH76TaIkJmYfRFgOV9MIpnx4X",
   "aad": "WyJ2Y2FyZCIsW1sidmVyc2lvbiIse30sInRleHQiLCI0LjAiXSxb
       ImZuIix7fSwidGV4dCIsIk1lcmlhZG9jIEJyYW5keWJ1Y2siXSxbIm4i
       LHt9LCJ0ZXh0IixbIkJyYW5keWJ1Y2siLCJNZXJpYWRvYyIsIk1yLiIs
       IiJdXSxbImJkYXkiLHt9LCJ0ZXh0IiwiVEEgMjk4MiJdLFsiZ2VuZGVy
       Iix7fSwidGV4dCIsIk0iXV1d",
   "iv": "veCx9ece2orS7c_N",
   "ciphertext": "Z_3cbr0k3bVM6N3oSNmHz7Lyf3iPppGf3Pj17wNZqteJ0
       Ui8p74SchQP8xygM1oFRWCNzeIa6s6BcEtp8qEFiqTUEyiNkOWDNoF14
       T_4NFqF-p2Mx8zkbKxI7oPK8KNarFbyxIDvICNqBLba-v3uzXBdB89fz
       OI-Lv4PjOFAQGHrgv1rjXAmKbgkft9cB4WeyZw8MldbBhc-V_KWZslrs
       LNygon_JJWd_ek6LQn5NRehvApqf9ZrxB4aq3FXBxOxCys35PhCdaggy
       2kfUfl2OkwKnWUbgXVD1C6HxLIlqHhCwXDG59weHrRDQeHyMRoBljoV3
       X_bUTJDnKBFOod7nLz-cj48JMx3SnCZTpbQAkFV",
   "tag": "vOaH_Rajnpy_3hOtqvZHRA"
 }
```

Read Serializations online: https://riptutorial.com/jwt/topic/5988/serializations

Credits

S. No	Chapters	Contributors
1	Getting started with jwt	Alex, Community, Florent Morselli, Nate Barbettini, pedrofb, RamenChef, Set
2	Invalidating Json Web Tokens	pedrofb
3	Serializations	Florent Morselli