JSON Web Token (JWT)
draft-ietf-oauth-json-web-token-00

Abstract

JSON Web Token (JWT) is a means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JSON object that is digitally signed or MACed using JSON Web Signature (JWS) and/or encrypted using JSON Web Encryption (JWE).

The suggested pronunciation of JWT is the same as the English word "jot".

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 23, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
2. Terminology
3. JSON Web Token (JWT) Overview
   3.1. Example JWT
1. Introduction

JSON Web Token (JWT) is a compact token format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. JWTs encode claims to be transmitted as a JSON object (as defined in RFC 4627) that is base64url encoded and digitally signed or MACed and/or encrypted. Signing and MACing is performed using JSON Web Signature (JWS) [JWS]. Encryption is performed using JSON Web Encryption (JWE) [JWE].

The suggested pronunciation of JWT is the same as the English word "jot".

2. Terminology

JSON Web Token (JWT)  
A string consisting of multiple parts, the first being the Encoded JWT Header, plus additional parts depending upon the contents of the header, with the parts being separated by period (".") characters, and each part containing base64url encoded content.

JWT Header  
A string representing a JSON object that describes the cryptographic operations applied to the JWT. When the JWT is digitally signed or MACed, the JWT Header is a JWS Header. When the JWT is encrypted, the JWT Header is a JWE Header.

Header Parameter Names  
The names of the members within the JWT Header.

Header Parameter Values  
The values of the members within the JWT Header.

JWT Claims Set
A string representing a JSON object that contains the claims conveyed by the JWT. When the JWT is digitally signed or MACed, the bytes of the UTF-8 representation of the JWT Claims Set are base64url encoded to create the Encoded JWS Payload. When the JWT is encrypted, the bytes of the UTF-8 representation of the JWT Claims Set are used as the JWE Plaintext.

Claim Names
The names of the members of the JSON object represented by the JWT Claims Set.

Claim Values
The values of the members of the JSON object represented by the JWT Claims Set.

Encoded JWT Header
Base64url encoding of the bytes of the UTF-8 representation of the JWT Claims Set.

Base64url Encoding
For the purposes of this specification, this term always refers to the URL- and filename-safe Base64 encoding described in [RFC4648], with the (non URL-safe) '=' padding characters omitted, as permitted by Section 3.2. (See Appendix C of [JWS] for notes on implementing base64url encoding without padding.)

StringOrURI
A JSON string value, with the additional requirement that while arbitrary string values MAY be used, any value containing a "." character MUST be a URI as defined in [RFC3986].

IntDate
A JSON numeric value representing the number of seconds from 1970-01-01T0:0:0Z UTC until the specified UTC date/time. See [RFC3339] for details regarding date/times in general and UTC in particular.

3.  JSON Web Token (JWT) Overview

JWTs represent a set of claims as a JSON object that is base64url encoded and digitally signed or MACed and/or encrypted. The JWT Claims Set represents this JSON object. As per [RFC4627] Section 2.2, the JSON object consists of zero or more name/value pairs (or members), where the names are strings and the values are arbitrary JSON values. These members are the claims represented by the JWT.

The member names within the JWT Claims Set are referred to as Claim Names. The corresponding values are referred to as Claim Values.

The bytes of the UTF-8 representation of the JWT Claims Set are digitally signed or MACed in the manner described in JSON Web Signature (JWS) [JWS] and/or encrypted in the manner described in JSON Web Encryption (JWE) [JWE].

The contents of the JWT Header describe the cryptographic operations applied to the JWT Claims Set. If the JWT Header is a JWS Header, the claims are digitally signed or MACed. If the JWT Header is a JWE Header, the claims are encrypted.

A JWT is represented as a JWS or JWE. The number of parts is dependent upon the representation of the resulting JWS or JWE.

3.1.  Example JWT

The following example JWT Header declares that the encoded object is a JSON Web Token (JWT) and the JWT is MACed using the HMAC SHA-256 algorithm:

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

Base64url encoding the bytes of the UTF-8 representation of the JWT Header yields this Encoded JWS Header value, which is used as the Encoded JWT Header:
The following is an example of a JWT Claims Set:

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

Base64url encoding the bytes of the UTF-8 representation of the JSON Claims Set yields this Encoded JWS Payload (with line breaks for display purposes only):

```text
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImlh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```

Signing the Encoded JWS Header and Encoded JWS Payload with the HMAC SHA-256 algorithm and base64url encoding the signature in the manner specified in [JWS], yields this Encoded JWS Signature:

```text
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wWlgFWFOEjXk
```

Concatenating these parts in this order with period characters between the parts yields this complete JWT (with line breaks for display purposes only):

```text
eyJ0eXAi0iJKV1QiLA0KICJhbGci0iJIUzI1NiJ9.
eyJpc3Mi0iJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImlh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wWlgFWFOEjXk
```

This computation is illustrated in more detail in [JWS], Appendix A.1.

### 4. JWT Claims

The JWT Claims Set represents a JSON object whose members are the claims conveyed by the JWT. The Claim Names within this object MUST be unique; JWTs with duplicate Claim Names MUST be rejected. Note however, that the set of claims that a JWT must contain to be considered valid is context-dependent and is outside the scope of this specification. When used in a security-related context, implementations MUST understand and support all of the claims present; otherwise, the JWT MUST be rejected for processing.

There are three classes of JWT Claim Names: Reserved Claim Names, Public Claim Names, and Private Claim Names.

#### 4.1. Reserved Claim Names

The following claim names are reserved. None of the claims defined below are intended to be mandatory, but rather, provide a starting point for a set of useful, interoperable claims. All the names are short because a core goal of JWTs is for the tokens to be compact. Additional reserved claim names MAY be defined via the IANA JSON Web Token Claims registry Section 9.1.
4.1.1. "exp" (Expiration Time) Claim

The exp (expiration time) claim identifies the expiration time on or after which the token MUST NOT be accepted for processing. The processing of the exp claim requires that the current date/time MUST be before the expiration date/time listed in the exp claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing an IntDate value. This claim is OPTIONAL.

4.1.2. "nbf" (Not Before) Claim

The nbf (not before) claim identifies the time before which the token MUST NOT be accepted for processing. The processing of the nbf claim requires that the current date/time MUST be after or equal to the not-before date/time listed in the nbf claim. Implementers MAY provide for some small leeway, usually no more than a few minutes, to account for clock skew. Its value MUST be a number containing an IntDate value. This claim is OPTIONAL.

4.1.3. "iat" (Issued At) Claim

The iat (issued at) claim identifies the time at which the JWT was issued. This claim can be used to determine the age of the token. Its value MUST be a number containing an IntDate value. This claim is OPTIONAL.

4.1.4. "iss" (Issuer) Claim

The iss (issuer) claim identifies the principal that issued the JWT. The processing of this claim is generally application specific. The iss value is case sensitive. Its value MUST be a string containing a StringOrURI value. This claim is OPTIONAL.

4.1.5. "aud" (Audience) Claim

The aud (audience) claim identifies the audience that the JWT is intended for. The principal intended to process the JWT MUST be identified with the value of the audience claim. If the principal processing the claim does not identify itself with the identifier in the aud claim value then the JWT MUST be rejected. The interpretation of the audience value is generally application specific. The aud value is case sensitive. Its value MUST be a string containing a StringOrURI value. This claim is OPTIONAL.

4.1.6. "prn" (Principal) Claim

The prn (principal) claim identifies the subject of the JWT. The processing of this claim is generally application specific. The prn value is case sensitive. Its value MUST be a string containing a StringOrURI value. This claim is OPTIONAL.

4.1.7. "jti" (JWT ID) Claim

The jti (JWT ID) claim provides a unique identifier for the JWT. The identifier value MUST be assigned in a manner that ensures that there is a negligible probability that the same value will be accidentally assigned to a different data object. The jti claim can be used to prevent
the JWT from being replayed. The \texttt{jti} value is case sensitive. Its value MUST be a string. This claim is OPTIONAL.

**4.1.8. "typ" (Type) Claim**

The \texttt{typ} (type) claim is used to declare a type for the contents of this JWT Claims Set. The \texttt{typ} value is case sensitive. Its value MUST be a string. This claim is OPTIONAL.

The values used for the \texttt{typ} claim SHOULD come from the same value space as the \texttt{typ} header parameter, with the same rules applying.

**4.2. Public Claim Names**

Claim names can be defined at will by those using JWTs. However, in order to prevent collisions, any new claim name SHOULD either be defined in the IANA JSON Web Token Claims registry \textbf{Section 9.1} or be a URI that contains a collision resistant namespace. Examples of collision resistant namespaces include:

- Domain Names,
- Object Identifiers (OIDs) as defined in the ITU-T X.660 and X.670 Recommendation series, or
- Universally Unique IDentifier (UUID) as defined in \textbf{RFC 4122} [RFC4122].

In each case, the definer of the name or value needs to take reasonable precautions to make sure they are in control of the part of the namespace they use to define the claim name.

**4.3. Private Claim Names**

A producer and consumer of a JWT may agree to any claim name that is not a Reserved Name \textbf{Section 4.1} or a Public Name \textbf{Section 4.2}. Unlike Public Names, these private names are subject to collision and should be used with caution.

**5. JWT Header**

The members of the JSON object represented by the JWT Header describe the cryptographic operations applied to the JWT and optionally, additional properties of the JWT. The member names within the JWT Header are referred to as Header Parameter Names. These names MUST be unique; JWTs with duplicate Header Parameter Names MUST be rejected. The corresponding values are referred to as Header Parameter Values.

Implementations MUST understand the entire contents of the header; otherwise, the JWT MUST be rejected for processing.

There are two ways of distinguishing whether the JWT is a JWS or JWE. The first is by examining the \texttt{alg} (algorithm) header value. If the value represents a signature algorithm, the JWT is a JWS; if it represents an encryption algorithm, the JWT is a JWE. A second method is determining whether an \texttt{enc} (encryption method) member exists. If the \texttt{enc} member exists, the JWT is a JWE; otherwise, the JWT is a JWS. Both methods will yield the same result.

JWS Header Parameters are defined by \textbf{[JWS]}. JWE Header Parameters are defined by \textbf{[JWE]}. This specification further specifies the use of the following header parameters in both the cases where the JWT is a JWS and where it is a JWE.
5.1. "typ" (Type) Header Parameter

The typ (type) header parameter is used to declare structural information about the JWT. In the normal case where nested signing or encryption operations are not employed, the use of this header parameter is OPTIONAL, and if present, it is RECOMMENDED that its value be either "JWT" or "urn:ietf:params:oauth:token-type:jwt". In the case that nested signing or encryption steps are employed, the use of this header parameter is REQUIRED; in this case, the value MUST either be "JWS", to indicate that a nested digitally signed or MACed JWT is carried in this JWT or "JWE", to indicate that a nested encrypted JWT is carried in this JWT.

6. Plaintext Jtos
ts

To support use cases where the JWT content is secured by a means other than a signature and/or encryption contained within the token (such as a signature on a data structure containing the token), JWTs MAY also be created without a signature or encryption. A plaintext JWT is a JWS using the none JWS alg header parameter value defined in JSON Web Algorithms (JWA) [JWA]; it is a JWS with an empty JWS Signature value.

6.1. Example Plaintext JWT

The following example JWT Header declares that the encoded object is a Plaintext JWT:

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

Base64url encoding the bytes of the UTF-8 representation of the JWT Header yields this Encoded JWT Header:

```
eyJhbGciOiJub25lIn0
```

The following is an example of a JWT Claims Set:

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

Base64url encoding the bytes of the UTF-8 representation of the JSON Claims Set yields this Encoded JWS Payload (with line breaks for display purposes only):

```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
```

cGxlLnVbS9pc19yb290Ijp0cnVlfQ

The Encoded JWS Signature is the empty string.

Concatenating these parts in this order with period characters between the parts yields this complete JWT (with line breaks for display purposes only):

```
eyJhbGciOiJub25lIn0.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLnVbS9pc19yb290Ijp0cnVlfQ.
```
7. Rules for Creating and Validating a JWT

To create a JWT, one MUST perform these steps. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps.

1. Create a JWT Claims Set containing the desired claims. Note that white space is explicitly allowed in the representation and no canonicalization is performed before encoding.
2. Let the Message be the bytes of the UTF-8 representation of the JWT Claims Set.
3. Create a JWT Header containing the desired set of header parameters. The JWT MUST conform to either the [JWS] or [JWE] specifications. Note that white space is explicitly allowed in the representation and no canonicalization is performed before encoding.
4. Base64url encode the bytes of the UTF-8 representation of the JWT Header. Let this be the Encoded JWT Header.
5. Depending upon whether the JWT is a JWS or JWE, there are two cases:
   - If the JWT is a JWS, create a JWS using the JWT Header as the JWS Header and the Message as the JWS Payload; all steps specified in [JWS] for creating a JWS MUST be followed.
   - Else, if the JWT is a JWE, create a JWE using the JWT Header as the JWE Header and the Message as the JWE Plaintext; all steps specified in [JWE] for creating a JWE MUST be followed.
6. If a nested signing or encryption operation will be performed, let the Message be the JWS or JWE, and return to Step 3, using a typ value of either "JWS" or "JWE" respectively in the new JWT Header created in that step.
7. Otherwise, let the resulting JWT be the JWS or JWE.

When validating a JWT the following steps MUST be taken. The order of the steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps. If any of the listed steps fails then the token MUST be rejected for processing.

1. The JWT MUST contain at least one period character.
2. Let the Encoded JWT Header be the portion of the JWT before the first period character.
3. The Encoded JWT Header MUST be successfully base64url decoded following the restriction given in this specification that no padding characters have been used.
4. The resulting JWT Header MUST be completely valid JSON syntax conforming to RFC 4627 [RFC4627].
5. The resulting JWT Header MUST be validated to only include parameters and values whose syntax and semantics are both understood and supported.
6. Determine whether the JWT is a JWS or a JWE by examining the alg (algorithm) header value and optionally, the enc (encryption method) header value, if present.
7. Depending upon whether the JWT is a JWS or JWE, there are two cases:
   - If the JWT is a JWS, all steps specified in [JWS] for validating a JWS MUST be followed. Let the Message be the result of base64url decoding the JWS Payload.
   - Else, if the JWT is a JWE, all steps specified in [JWE] for validating a JWE MUST be followed. Let the Message be the JWE Plaintext.
8. If the JWT Header contains a typ value of either "JWS" or "JWE", then the Message contains a JWT that was the subject of nested signing or encryption operations, respectively. In this case, return to Step 1, using the Message as the JWT.
9. Otherwise, let the JWT Claims Set be the Message.
10. The JWT Claims Set MUST be completely valid JSON syntax conforming to RFC 4627 [RFC4627].
11. When used in a security-related context, the JWT Claims Set MUST be validated to only include claims whose syntax and semantics are both understood and supported.

Processing a JWT inevitably requires comparing known strings to values in the token. For example, in checking what the algorithm is, the Unicode string encoding alg will be checked against the member names in the JWT Header to see if there is a matching header parameter name. A similar process occurs when determining if the value of the alg header parameter represents a supported algorithm.
Comparisons between JSON strings and other Unicode strings MUST be performed as specified below:

1. Remove any JSON applied escaping to produce an array of Unicode code points.
2. Unicode Normalization [USA15] MUST NOT be applied at any point to either the JSON string or to the string it is to be compared against.
3. Comparisons between the two strings MUST be performed as a Unicode code point to code point equality comparison.

8.Cryptographic Algorithms

JWTs use JSON Web Signature (JWS) [JWS] and JSON Web Encryption (JWE) [JWE] to sign and/or encrypt the contents of the JWT.

Of the JWS signing algorithms, only HMAC SHA-256 and none MUST be implemented by conforming JWT implementations. It is RECOMMENDED that implementations also support the RSA SHA-256 and ECDSA P-256 SHA-256 algorithms. Support for other algorithms and key sizes is OPTIONAL.

If an implementation provides encryption capabilities, of the JWE encryption algorithms, only RSA-PKCS1-1.5 with 2048 bit keys, AES-128-KW, AES-256-KW, AES-128-CBC, and AES-256-CBC MUST be implemented by conforming implementations. It is RECOMMENDED that implementations also support ECDH-ES with 256 bit keys, AES-128-GCM, and AES-256-GCM. Support for other algorithms and key sizes is OPTIONAL.

9. IANA Considerations

9.1. JSON Web Token Claims Registry

This specification establishes the IANA JSON Web Token Claims registry for reserved JWT claim names. Inclusion in the registry is RFC Required in the RFC 5226 [RFC5226] sense. The registry records the reserved claim name and a reference to the RFC that defines it. This specification registers the claim names defined in Section 4.1.


This specification registers the value token-type:jwt in the registry urn:ietf:params:oauth:token-type:jwt established in An IETF URN Sub-Namespace for OAuth [I-D.ietf-oauth-urn-sub-ns].

- URN: urn:ietf:params:oauth:token-type:jwt
- Common Name: JSON Web Token (JWT) Token Type
- Change controller: IETF
- Description: [[this document]]

9.3. Registration of application/jwt MIME Media Type

This specification registers the application/jwt MIME Media Type RFC 2045 [RFC2045].

- Type name: application
- Subtype name: jwt
- Required parameters: n/a
9.4. Registration of "JWT" Type Value

This specification registers the following typ header parameter value in the JSON Web Signature and Encryption "typ" Values registry established by the JSON Web Algorithms (JWA) [JWA] specification:

"typ" header parameter value: "JWT"
Abbreviation for MIME type: application/jwt
Change controller: IETF
Description: [[ this document ]]

10. Security Considerations

All the security considerations in the JWS specification also apply to JWT, as do the JWE security considerations when encryption is employed. In particular, the JWS JSON Security Considerations and Unicode Comparison Security Considerations apply equally to the JWT Claims Set in the same manner that they do to the JWS Header.

11. Open Issues and Things To Be Done (TBD)

The following items remain to be done in this draft:

- Provide an example of an encrypted JWT.
12. References

12.1. Normative References


12.2. Informative References


Appendix A. Relationship of JWTs to SAML Tokens

SAML 2.0 [OASIS.saml-core-2.0-os] provides a standard for creating tokens with much greater expressivity and more security options than supported by JWTs. However, the cost of this flexibility and expressiveness is both size and complexity. In addition, SAML’s use of XML [W3C.CR-xmll1-20021015] and XML DSIG [RFC3275] only contributes to the size of SAML tokens.

JWTs are intended to provide a simple token format that is small enough to fit into HTTP headers and query arguments in URIs. It does this by supporting a much simpler token model than SAML and using the JSON [RFC4627] object encoding syntax. It also supports securing tokens using Message Authentication Codes (MACs) and digital signatures using a smaller (and less flexible) format than XML DSIG.

Therefore, while JWTs can do some of the things SAML tokens do, JWTs are not intended as a full replacement for SAML tokens, but rather as a compromise token format to be used when space is at a premium.
Appendix B. Relationship of JWTs to Simple Web Tokens ( SWTs)

Both JWTs and Simple Web Tokens ( SWT), at their core, enable sets of claims to be communicated between applications. For SWTs, both the claim names and claim values are strings. For JWTs, while claim names are strings, claim values can be any JSON type. Both token types offer cryptographic protection of their content: SWTs with HMAC SHA-256 and JWTs with a choice of algorithms, including HMAC SHA-256, RSA SHA-256, and ECDSA P-256 SHA-256.

Appendix C. Acknowledgements

The authors acknowledge that the design of JWTs was intentionally influenced by the design and simplicity of Simple Web Tokens ( SWT) and ideas for JSON tokens that Dick Hardt discussed within the OpenID community.

Solutions for signing JSON content were previously explored by Magic Signatures [MagicSignatures], JSON Simple Sign [JSS], and Canvas Applications [CanvasApp], all of which influenced this draft. Dirk Balfanz, Yaron Y. Goland, John Panzer, and Paul Tarjan all made significant contributions to the design of this specification.

Appendix D. Document History

-00

- Created the initial IETF draft based upon draft-jones-json-web-token-10 with no normative changes.

Authors' Addresses

Michael B. Jones
Microsoft
Email: mbj@microsoft.com
URI: http://self-issued.info/

John Bradley
Ping Identity
Email: ve7jtb@ve7jtb.com

Nat Sakimura
Nomura Research Institute
Email: n-sakimura@nri.co.jp