JSON Web Token (JWT)
draft-ietf-oauth-json-web-token-15

Abstract

JSON Web Token (JWT) is a compact URL-safe means of representing claims to be transferred between two parties. The claims in a JWT are encoded as a JavaScript Object Notation (JSON) object that is used as the payload of a JSON Web Signature (JWS) structure or as the plaintext of a JSON Web Encryption (JWE) structure, enabling the claims to be digitally signed or MACed and/or encrypted.

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

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 July 24, 2014.

Copyright Notice

Copyright (c) 2014 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
   1.1.  Notational Conventions
2.   Terminology
3.   JSON Web Token (JWT) Overview
   3.1.  Example JWT
4.   JWT Claims
   4.1.  Registered Claim Names
      4.1.1.  "iss" (Issuer) Claim
1. Introduction

JSON Web Token (JWT) is a compact claims representation format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. JWTs encode claims to be transmitted as a JavaScript Object Notation (JSON) object that is used as the payload of a JSON Web Signature (JWS) [JWS] structure or as the plaintext of a JSON Web Encryption (JWE) [JWE] structure, enabling the claims to be digitally signed or MACed and/or encrypted. JWTs are always represented using the JWS Compact Serialization or the JWE Compact Serialization.

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

1.1. Notational Conventions

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 Key words for use in RFCs to Indicate Requirement Levels [RFC2119]. If these words are used without being spelled in uppercase then they are to be interpreted with their normal natural language meanings.
2. Terminology

**JSON Web Token (JWT)**
A string representing a set of claims as a JSON object that is encoded in a JWS or JWE, enabling the claims to be digitally signed or MACed and/or encrypted.

**Base64url Encoding**
Base64 encoding using the URL- and filename-safe character set defined in Section 5 of [RFC 4648](https://tools.ietf.org/html/rfc4648), with all trailing '=' characters omitted (as permitted by Section 3.2). (See Appendix C of [JWS](https://tools.ietf.org/html/rfc7519) for notes on implementing base64url encoding without padding.)

**JWT Header**
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**
A name/value pair that is member of the JWT Header.

**Header Parameter Name**
The name of a member of the JWT Header.

**Header Parameter Value**
The value of a member of the JWT Header.

**JWT Claims Set**
A JSON object that contains the Claims conveyed by the JWT.

**Claim**
A piece of information asserted about a subject. A Claim is represented as a name/value pair consisting of a Claim Name and a Claim Value.

**Claim Name**
The name portion of a Claim representation. A Claim Name is always a string.

**Claim Value**
The value portion of a Claim representation. A Claim Value can be any JSON value.

**Encoded JWT Header**
Base64url encoding of the JWT Header.

**Nested JWT**
A JWT in which nested signing and/or encryption are employed. In nested JWTs, a JWT is used as the payload or plaintext value of an enclosing JWS or JWE structure, respectively.

**Plaintext JWT**
A JWT whose Claims are not integrity protected or encrypted.

**Collision-Resistant Name**
A name in a namespace that enables names to be allocated in a manner such that they are highly unlikely to collide with other names. Examples of collision-resistant namespaces include: Domain Names, Object Identifiers (OIDs) as defined in the ITU-T X.660 and X.670 Recommendation series, and Universally Unique Identifiers (UUIDs) [RFC4122]. When using an administratively delegated namespace, the definer of a name needs to take reasonable precautions to ensure they are in control of the portion of the namespace they use to define the name.

**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 [RFC3986]. StringOrURI values are compared as case-sensitive strings with no transformations or canonicalizations applied.

**IntDate**
A JSON numeric value representing the number of seconds from 1970-01-01T0:0:0Z UTC until the specified UTC date/time. See RFC 3339 [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 encoded in a JWS and/or JWE structure. This JSON object is the JWT Claims Set. As per Section 4 of [I-D.ietf-json-rfc4627bis], 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...
The member names within the JWT Claims Set are referred to as Claim Names. The corresponding values are referred to as Claim Values.

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 JWT is represented as a JWS, and the claims are digitally signed or MACed, with the JWT Claims Set being the JWS Payload. If the JWT Header is a JWE Header, the JWT is represented as a JWE, and the claims are encrypted, with the JWT Claims Set being the input Plaintext. A JWT may be enclosed in another JWE or JWS structure to create a Nested JWT, enabling nested signing and encryption to be performed.

A JWT is represented as a sequence of URL-safe parts separated by period (\[\].) characters. Each part contains a base64url encoded value. The number of parts in the JWT is dependent upon the representation of the resulting JWS or JWE object using the JWS Compact Serialization or the JWE Compact Serialization.

### 3.1. Example JWT

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

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

The following octet sequence is the UTF-8 representation of the JWT Header/JWS Header above:

```
[123, 34, 116, 121, 112, 34, 58, 34, 74, 87, 84, 34, 44, 13, 10, 32, 34, 97, 108, 103, 34, 58, 34, 72, 83, 50, 53, 54, 34, 125]
```

Base64url encoding the octets of the UTF-8 representation of the JWT Header yields this Encoded JWT Header value (which is also the underlying encoded JWS Header value):

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
```

The following is an example of a JWT Claims Set:

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

The following octet sequence, which is the UTF-8 representation of the JWT Claims Set above, is the JWS Payload:

```
```

Base64url encoding the JWS Payload yields this encoded JWS Payload (with line breaks for display purposes only):

```
eypc3Mi0iIjyb2UilA0KICJ1eHAi0jEzMDA4MTkzODAsDQogImh0dHA6Ly9leGtcc6xlMnVbS9pc19yb290Ijp0cnVlFQ
```

Computing the MAC of the encoded JWS Header and encoded JWS Payload with the HMAC SHA-256 algorithm and base64url encoding the HMAC value in the manner specified in...
SHA-256 algorithm and base64url encoding the HMAC value in the manner specified in \[JWS\], yields this encoded JWS Signature:

dBjftJeZ4CVP-mB92K27uhbUU1p1r_wW1gFWF0EjXk

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

eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
dBjftJeZ4CVP-mB92K27uhbUU1p1r_wW1gFWF0EjXk

This computation is illustrated in more detail in Appendix A.1 of \[JWS\]. See Appendix A.1 for an example of an encrypted JWT.

4. JWT Claims

The JWT Claims Set represents a JSON object whose members are the claims conveyed by the JWT. The Claim Names within a JWT Claims Set MUST be unique; recipients MUST either reject JWTs with duplicate Claim Names or use a JSON parser that returns only the lexically last duplicate member name, as specified in Section 15.12 (The JSON Object) of ECMAScript 5.1 [ECMAScript].

The set of claims that a JWT must contain to be considered valid is context-dependent and is outside the scope of this specification. Specific applications of JWTs will require implementations to understand and process some claims in particular ways. However, in the absence of such requirements, all claims that are not understood by implementations SHOULD be ignored.

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

4.1. Registered Claim Names

The following Claim Names are registered in the IANA JSON Web Token Claims registry defined in Section 10.1. None of the claims defined below are intended to be mandatory to use, 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 representation to be compact.

4.1.1. "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 a case-sensitive string containing a StringOrURI value. Use of this claim is OPTIONAL.

4.1.2. "sub" (Subject) Claim

The sub (subject) claim identifies the principal that is the subject of the JWT. The Claims in a JWT are normally statements about the subject. The subject value MAY be scoped to be locally unique in the context of the issuer or MAY be globally unique. The processing of this claim is generally application specific. The sub value is a case-sensitive string containing a
4.1.3. "aud" (Audience) Claim

The aud (audience) claim identifies the audiences that the JWT is intended for. Each principal intended to process the JWT MUST identify itself with a value in audience claim. If the principal processing the claim does not identify itself with a value in the aud claim, then the JWT MUST be rejected. In the general case, the aud value is an array of case-sensitive strings, each containing a StringOrURI value. In the special case when the JWT has one audience, the aud value MAY be a single case-sensitive string containing a StringOrURI value. The interpretation of audience values is generally application specific. Use of this claim is OPTIONAL.

4.1.4. "exp" (Expiration Time) Claim

The exp (expiration time) claim identifies the expiration time on or after which the JWT 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. Use of this claim is OPTIONAL.

4.1.5. "nbf" (Not Before) Claim

The nbf (not before) claim identifies the time before which the JWT 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. Use of this claim is OPTIONAL.

4.1.6. "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 JWT. Its value MUST be a number containing an IntDate value. Use of 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 jti value is a case-sensitive string. Use of this claim is OPTIONAL.

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 registered in the IANA JSON Web Token Claims registry defined in Section 10.1 or be a Public Name: a value that contains a Collision-Resistant Name. 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 use Claim Names that are Private Names: names that are not Registered Claim Names Section 4.1 or Public Claim Names Section 4.2. Unlike Public Claim Names, Private Claim 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; recipients MUST either reject JWEs with duplicate Header Parameter Names or use a JSON parser that returns only the lexically last duplicate member name, as specified in Section 15.12 (The JSON Object) of ECMAScript 5.1 [ECMAScript]. The corresponding values are referred to as Header Parameter Values.

JWS Header Parameters are defined by [JWS]. JWE Header Parameters are defined by [JWE]. This specification further specifies the use of the following Header Parameter 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 defined by [JWS] and [JWE] is used to declare the MIME Media Type [IANA.MediaTypes] of this complete JWT in contexts where this is useful to the application. This parameter has no effect upon the JWT processing. If present, it is RECOMMENDED that its value be JWT to indicate that this object is a JWT. While media type names are not case-sensitive, it is RECOMMENDED that JWT always be spelled using uppercase characters for compatibility with legacy implementations. Use of this Header Parameter is OPTIONAL.

5.2. "cty" (Content Type) Header Parameter

The cty (content type) Header Parameter defined by [JWS] and [JWE] is used by this specification to convey 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 NOT RECOMMENDED. In the case that nested signing or encryption is employed, this Header Parameter MUST be present; in this case, the value MUST be JWT, to indicate that a Nested JWT is carried in this JWT. While media type names are not case-sensitive, it is RECOMMENDED that JWT always be spelled using uppercase characters for compatibility with legacy implementations. See Appendix A.2 for an example of a Nested JWT.

5.3. Replicating Claims as Header Parameters

In some applications using encrypted JWEs, it is useful to have an unencrypted representation of some Claims. This might be used, for instance, in application processing rules to determine whether and how to process the JWT before it is decrypted.

This specification allows Claims present in the JWT Claims Set to be replicated as Header
This specification allows Claims present in the JWT Claims Set to be replicated as Header Parameters in a JWT that is a JWE, as needed by the application. If such replicated Claims are present, the application receiving them SHOULD verify that their values are identical. It is the responsibility of the application to ensure that only claims that are safe to be transmitted in an unencrypted manner are replicated as Header Parameter Values in the JWT.

This specification registers the iss (issuer), sub (subject), and aud (audience) Header Parameter Names for the purpose of providing unencrypted replicas of these Claims in encrypted JWTs for applications that need them. Other specifications MAY similarly register other names that are registered Claim Names as Header Parameter Names, as needed.

6. Plaintext JWTs

To support use cases where the JWT content is secured by a means other than a signature and/or encryption contained within the JWT (such as a signature on a data structure containing the JWT), 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 the empty string for its 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 octets 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 octets of the UTF-8 representation of the JWT Claims Set yields this encoded JWS Payload (with line breaks for display purposes only):

```
eyJpcmx3MjoiIjI1b2UiLA0KICJ1eHAI0jEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
 cGx1LmNvbS9pc19yb290Ijp0cnVlfQ
```

The encoded JWS Signature is the empty string.

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

```
eyJhbGciOiJub25lIn0
.
eyJpcmx3MjoiIjI1b2UiLA0KICJ1eHAI0jEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGx1LmNvbS9pc19yb290Ijp0cnVlfQ
.
```
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 need be performed before encoding.
2. Let the Message be the octets 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 need be performed before encoding.
4. Base64url encode the octets 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 cty (content type) value of JWT 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 JWT 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 [I-D.ietf-json-rfc4627bis].
5. The resulting JWT Header MUST be validated to only include parameters and values whose syntax and semantics are both understood and supported or that are specified as being ignored when not understood.
6. Determine whether the JWT is a JWS or a JWE using any of the methods described in Section 9 of [JWE].
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 cty (content type) value of JWT, then the Message is a JWT that was the subject of nested signing or encryption operations. 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 [I-D.ietf-json-rfc4627bis].

7.1. String Comparison Rules

Processing a JWT inevitably requires comparing known strings to values in JSON objects. 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.
Comparisons between JSON strings and other Unicode strings MUST be performed by comparing Unicode code points without normalization, as specified in the String Comparison Rules in Section 5.3 of [JWS].

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 signature and MAC algorithms specified in JSON Web Algorithms (JWA) [JWA], only HMAC SHA-256 (HS256) and none MUST be implemented by conforming JWT implementations. It is RECOMMENDED that implementations also support RSASSA-PKCS1-V1_5 with the SHA-256 hash algorithm (RS256) and ECDSA using the P-256 curve and the SHA-256 hash algorithm (ES256). Support for other algorithms and key sizes is OPTIONAL.

If an implementation provides encryption capabilities, of the encryption algorithms specified in [JWA], only RSAES-PKCS1-V1_5 with 2048 bit keys (RSA1_5), AES Key Wrap with 128 and 256 bit keys (A128KW and A256KW), and the composite authenticated encryption algorithm using AES CBC and HMAC SHA-2 (A128CBC-HS256 and A256CBC-HS512) MUST be implemented by conforming implementations. It is RECOMMENDED that implementations also support using ECDH-ES to agree upon a key used to wrap the Content Encryption Key (ECDH-ES+A128KW and ECDH-ES+A256KW) and AES in Galois/Counter Mode (GCM) with 128 bit and 256 bit keys (A128GCM and A256GCM). Support for other algorithms and key sizes is OPTIONAL.

9. URI for Declaring that Content is a JWT

This specification registers the URN urn:ietf:params:oauth:token-type:jwt for use by applications that declare content types using URIs (rather than, for instance, MIME Media Types) to indicate that the content referred to is a JWT.

10. IANA Considerations

10.1. JSON Web Token Claims Registry

This specification establishes the IANA JSON Web Token Claims registry for JWT Claim Names. The registry records the Claim Name and a reference to the specification that defines it. This specification registers the Claim Names defined in Section 4.1.

Values are registered with a Specification Required [RFC5226] after a two-week review period on the [TBD]@ietf.org mailing list, on the advice of one or more Designated Experts. However, to allow for the allocation of values prior to publication, the Designated Expert(s) may approve registration once they are satisfied that such a specification will be published.

Registration requests must be sent to the [TBD]@ietf.org mailing list for review and comment, with an appropriate subject (e.g., "Request for access token type: example"). [Note to the RFC Editor: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: jwt-reg-review.]

Within the review period, the Designated Expert(s) will either approve or deny the registration request, communicating this decision to the review list and IANA. Denials should include an explanation and, if applicable, suggestions as to how to make the request successful. Registration requests that are undetermined for a period longer than 21 days can be brought to the IESG's attention (using the ietf@iesg.org mailing list) for resolution.
Criteria that should be applied by the Designated Expert(s) includes determining whether the proposed registration duplicates existing functionality, determining whether it is likely to be of general applicability or whether it is useful only for a single application, and whether the registration makes sense.

IANA must only accept registry updates from the Designated Expert(s) and should direct all requests for registration to the review mailing list.

It is suggested that multiple Designated Experts be appointed who are able to represent the perspectives of different applications using this specification, in order to enable broadly-informed review of registration decisions. In cases where a registration decision could be perceived as creating a conflict of interest for a particular Expert, that Expert should defer to the judgment of the other Expert(s).

10.1.1. Registration Template

Claim Name:
The name requested (e.g., "example"). Because a core goal of this specification is for the resulting representations to be compact, it is RECOMMENDED that the name be short -- not to exceed 8 characters without a compelling reason to do so. This name is case-sensitive. Names may not match other registered names in a case-insensitive manner unless the Designated Expert(s) state that there is a compelling reason to allow an exception in this particular case.

Claim Description:
Brief description of the Claim (e.g., "Example description").

Change Controller:
For Standards Track RFCs, state "IESG". For others, give the name of the responsible party. Other details (e.g., postal address, email address, home page URI) may also be included.

Specification Document(s):
Reference to the document(s) that specify the parameter, preferably including URI(s) that can be used to retrieve copies of the document(s). An indication of the relevant sections may also be included but is not required.

10.1.2. Initial Registry Contents

- Claim Name: iss
  Claim Description: Issuer
  Change Controller: IESG
  Specification Document(s): Section 4.1.1 of [[ this document ]]

- Claim Name: sub
  Claim Description: Subject
  Change Controller: IESG
  Specification Document(s): Section 4.1.2 of [[ this document ]]

- Claim Name: aud
  Claim Description: Audience
  Change Controller: IESG
  Specification Document(s): Section 4.1.3 of [[ this document ]]

- Claim Name: exp
  Claim Description: Expiration Time
  Change Controller: IESG
  Specification Document(s): Section 4.1.4 of [[ this document ]]

- Claim Name: nbf
  Claim Description: Not Before
  Change Controller: IESG
  Specification Document(s): Section 4.1.5 of [[ this document ]]

- Claim Name: iat
10.2. Sub-Namespace Registration of urn:ietf:params:oauth:token-type:jwt

10.2.1. Registry Contents

This specification registers the value \textit{token-type:jwt} in the IANA urn:ietf:params:oauth registry established in \textbf{An IETF URN Sub-Namespace for OAuth} [RFC6755], which can be used to indicate that the content is a JWT.

- URN: urn:ietf:params:oauth:token-type:jwt
- Common Name: JSON Web Token (JWT) Token Type
- Change Controller: IESG
- Specification Document(s): [[this document]]

10.3. Media Type Registration

10.3.1. Registry Contents

This specification registers the \textit{application/jwt} Media Type [RFC2046] in the MIME Media Types registry [IANA.MediaTypes], which can be used to indicate that the content is a JWT.

- Type Name: application
- Subtype Name: jwt
- Required Parameters: n/a
- Optional Parameters: n/a
- Encoding considerations: 8bit; JWT values are encoded as a series of base64url encoded values (some of which may be the empty string) separated by period (\textperiodcentered) characters.
- Security Considerations: See the Security Considerations section of [[ this document ]]
- Interoperability Considerations: n/a
- Published Specification: [[ this document ]]
- Applications that use this media type: OpenID Connect, Mozilla Persona, Salesforce, Google, numerous others
- Additional Information: Magic number(s): n/a, File extension(s): n/a, Macintosh file type code(s): n/a
- Person & email address to contact for further information: Michael B. Jones, mbj@microsoft.com
- Intended Usage: COMMON
- Restrictions on Usage: none
- Author: Michael B. Jones, mbj@microsoft.com
- Change Controller: IESG

10.4. Registration of JWE Header Parameter Names

This specification registers specific Claim Names defined in \textbf{Section 4.1} in the IANA JSON
Web Signature and Encryption Header Parameters registry defined in [JWS] for use by
Claims replicated as Header Parameters, per Section 5.3.

10.4.1. Registry Contents

- Header Parameter Name: iss
  - Header Parameter Description: Issuer
  - Header Parameter Usage Location(s): JWE
  - Change Controller: IESG
  - Specification Document(s): Section 4.1.1 of [[ this document ]]

- Header Parameter Name: sub
  - Header Parameter Description: Subject
  - Header Parameter Usage Location(s): JWE
  - Change Controller: IESG
  - Specification Document(s): Section 4.1.2 of [[ this document ]]

- Header Parameter Name: aud
  - Header Parameter Description: Audience
  - Header Parameter Usage Location(s): JWE
  - Change Controller: IESG
  - Specification Document(s): Section 4.1.3 of [[ this document ]]

11. Security Considerations

All of the security issues faced by any cryptographic application must be faced by a
JWT/JWS/JWE/JWK agent. Among these issues are protecting the user’s private and symmetric
keys, preventing various attacks, and helping the user avoid mistakes such as inadvertently
encrypting a message for the wrong recipient. The entire list of security considerations is
beyond the scope of this document.

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.

While syntactically, the signing and encryption operations for Nested JWTs may be applied in
any order, normally senders should sign the message and then encrypt the result (thus
encrypting the signature). This prevents attacks in which the signature is stripped, leaving
just an encrypted message, as well as providing privacy for the signer. Furthermore,
signatures over encrypted text are not considered valid in many jurisdictions.

Note that potential concerns about security issues related to the order of signing and
encryption operations are already addressed by the underlying JWS and JWE specifications; in
particular, because JWE only supports the use of authenticated encryption algorithms,
cryptographic concerns about the potential need to sign after encryption that apply in many
contexts do not apply to this specification.

12. References

12.1. Normative References

- [I-D.ietf-json-rfc4627bis] Bray, T., “The JSON Data Interchange Format,” draft-ietf-json-rfc4627bis-10 (work in progress),
  December 2013 (TXT).
12.2. Informative References


Appendix A. JWT Examples

This section contains examples of JWTs. For other example JWTs, see Section 6.1 and Appendices A.1, A.2, and A.3 of [JWS].

A.1. Example Encrypted JWT

This example encrypts the same claims as used in Section 3.1 to the recipient using RSAES-PKCS1-V1_5 and AES_128_CBC_HMAC_SHA_256.

The following example JWE Header (with line breaks for display purposes only) declares that:

- the Content Encryption Key is encrypted to the recipient using the RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key and
- the Plaintext is encrypted using the AES_128_CBC_HMAC_SHA_256 algorithm to produce the Ciphertext.

```
{"alg":"RSA1_5","enc":"A128CBC-HS256"}
```

Other than using the octets of the UTF-8 representation of the JWT Claims Set from Section 3.1 as the plaintext value, the computation of this JWT is identical to the computation of the JWE in Appendix A.2 of [JWE], including the keys used.
The final result in this example (with line breaks for display purposes only) is:

```
eyJhbGciOiJSU0ExEzZzU1CjI1BMTI4Q0JDLUhTMjU2In0.
QR10tW2ug2WyPbnbQrRARteEk9kD02w8qDcjihSjI3dvi11nQwhXaKH4MqAkQtM
onfA6rBA3MaHoa415sv3aeuBwnD8J-U7Ah6cWaf3ZwwFKDFUusWHSK-IPkxLG
TkND09xjyj0Rj-CAg0Pj-Sd8fQRnJvWn_hXv1BNMhzUjPyYwEsRHJHzjAD261ma
SOtsr0upbYGoQcXUWFDn7moXPRTFDBE-NoQX7n7YmmpUDkr-Cx9o6ng6wQj3nM52
YCI1txoQPzjb17JwuBoB7d0h3dZ24W1n1Ie5h8v1K4rB8xgKvR7u8kgFrEn_a
1rZgNStysnm2TR0F8691Q.
Axy8DctDaGlslbGljb3RoQZ.
MKo1e7U7QrG6nSxTLx6Mqw0rbHzAKewNyDypIAeZ72deHxz3roJDXQyhxx0wKaM
HDJjUEOK1wrkHthpqEanSBNYHZGMN0V7slnEu9g3J8.
fiK51VwhsxJ-siBMR-YFIa
```

A.2. Example Nested JWT

This example shows how a JWT can be used as the payload of a JWE or JWS to create a Nested JWT. In this case, the JWT Claims Set is first signed, and then encrypted.

The inner signed JWT is identical to the example in Appendix A.2 of [JWS]. Therefore, its computation is not repeated here. This example then encrypts this inner JWT to the recipient using RSAES-PKCS1-V1_5 and AES_128_CBC_HMAC_SHA_256.

The following example JWE Header (with line breaks for display purposes only) declares that:

- the Content Encryption Key is encrypted to the recipient using the RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key,
- the Plaintext is encrypted using the AES_128_CBC_HMAC_SHA_256 algorithm to produce the Ciphertext, and
- the Plaintext is itself a JWT.

```
{"alg":"RSA1_5","enc":"A128CBC-HS256","cty":"JWT"}
```

Base64url encoding the octets of the UTF-8 representation of the JWE Header yields this encoded JWE Header value:

```
eyJhbGciOiJSU0ExEzZzU1CjI1BMTI4Q0JDLuhTMjU2Iiw1Y3R5IjoiSlUI
```

The computation of this JWT is identical to the computation of the JWE in Appendix A.2 of [JWE], other than that different JWE Header, Plaintext, Initialization Vector, and Content Encryption Key values are used. (The RSA key used is the same.)

The Payload used is the octets of the ASCII representation of the JWT at the end of Appendix Section A.2.1 of [JWS] (with all whitespace and line breaks removed), which is a sequence of 458 octets.

The Initialization Vector value used is:

```
[82, 101, 100, 109, 111, 110, 100, 32, 87, 65, 32, 57, 56, 48, 53, 50]
```

This example uses the Content Encryption Key represented in JSON Web Key [JWK] format below:

```
{"kty":"oct",
 "k":"GawgguFyGrWKav7AX4VKUg"}
```

The final result for this Nested JWT (with line breaks for display purposes only) is:

```
eyJhbGciOiJSU0ExEzZzU1CjI1BMTI4Q0JDLuhTMjU2Iiw1Y3R5IjoiSlUI
QR10tW2ug2WyPbnbQrRARteEk9kD02w8qDcjihSjI3dvi11nQwhXaKH4MqAkQtM
onfA6rBA3MaHoa415sv3aeuBwnD8J-U7Ah6cWaf3ZwwFKDFUusWHSK-IPkxLG
TkND09xjyj0Rj-CAg0Pj-Sd8fQRnJvWn_hXv1BNMhzUjPyYwEsRHJHzjAD261ma
SOtsr0upbYGoQcXUWFDn7moXPRTFDBE-NoQX7n7YmmpUDkr-Cx9o6ng6wQj3nM52
YCI1txoQPzjb17JwuBoB7d0h3dZ24W1n1Ie5h8v1K4rB8xgKvR7u8kgFrEn_a
1rZgNStysnm2TR0F8691Q.
Axy8DctDaGlslbGljb3RoQZ.
MKo1e7U7QrG6nSxTLx6Mqw0rbHzAKewNyDypIAeZ72deHxz3roJDXQyhxx0wKaM
HDJjUEOK1wrkHthpqEanSBNYHZGMN0V7slnEu9g3J8.
fiK51VwhsxJ-siBMR-YFIa
```
Appendix B. Relationship of JWTs to SAML Assertions

SAML 2.0 [OASIS.saml-core-2.0-0s] provides a standard for creating security tokens with greater expressivity and more security options than supported by JWTs. However, the cost of this flexibility and expressiveness is both size and complexity. SAML’s use of XML and XML DSIG contributes to the size of SAML assertions; its use of XML and especially XML Canonicalization contributes to their complexity.

JWTs are intended to provide a simple security 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 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 assertions do, JWTs are not intended as a full replacement for SAML assertions, but rather as a token format to be used when ease of implementation or compactness are considerations.

JWTs are intended to provide a simple security 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 [I-D.ietf-json-rfc4627bis] 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 assertions do, JWTs are not intended as a full replacement for SAML assertions, but rather as a token format to be used when ease of implementation or compactness are considerations.

SAML Assertions are always statements made by an entity about a subject. JWTs are often used in the same manner, with the entity making the statements being represented by the iss (issuer) claim, and the subject being represented by the sub (subject) claim. However, with these claims being optional, other uses of the JWT format are also permitted.

Appendix C. Relationship of JWTs to Simple Web Tokens (SWTs)

Both JWTs and Simple Web Tokens SWT [SWT], at their core, enable sets of claims to be communicated between applications. For SWTs, both the entity making the statements being represented by the iss (issuer) claim, and the subject being represented by the sub (subject) claim. However, these claims being optional, other uses of the SWT format are also permitted.

Appendix D. 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.

This specification is the work of the OAuth Working Group, which includes dozens of active and dedicated participants. In particular, the following individuals contributed ideas, feedback, and wording that influenced this specification:


Hannes Tschofenig and Derek Atkins chaired the OAuth working group and Sean Turner and Stephen Farrell served as Security area directors during the creation of this specification.

Appendix E. Document History
[[ to be removed by the RFC Editor before publication as an RFC ]]

-15
- Replaced references to RFC 4627 with draft-ietf-json-rfc4627bis.

-14
- Referenced the JWE section on Distinguishing between JWS and JWE Objects.

-13
- Added Claim Description registry field.
- Used Header Parameter Description registry field.
- Removed the phrases "JWA signing algorithms" and "JWA encryption algorithms".
- Removed the term JSON Text Object.

-12
- Tracked the JOSE change refining the typ and cty definitions to always be MIME Media Types, with the omission of "application/" prefixes recommended for brevity. For compatibility with legacy implementations, it is RECOMMENDED that JWT always be spelled using uppercase characters when used as a typ or cty value. As side effects, this change removed the typ Claim definition and narrowed the uses of the URI urn:ietf:params:oauth:token-type:jwt.
- Updated base64url definition to match JOSE definition.
- Changed terminology from "Reserved Claim Name" to "Registered Claim Name" to match JOSE terminology change.
- Applied other editorial changes to track parallel JOSE changes.
- Clarified that the subject value may be scoped to be locally unique in the context of the issuer or may be globally unique.

-11
- Added a Nested JWT example.
- Added sub to the list of Claims registered for use as Header Parameter values when an unencrypted representation is required in an encrypted JWT.

-10
- Allowed Claims to be replicated as Header Parameters in encrypted JWTs as needed by applications that require an unencrypted representation of specific Claims.

-09
- Clarified that the typ header parameter is used in an application-specific manner and has no effect upon the JWT processing.
- Stated that recipients MUST either reject JWTs with duplicate Header Parameter Names or with duplicate Claim Names or use a JSON parser that returns only the lexically last duplicate member name.

- Tracked a change to how JWEs are computed (which only affected the example encrypted JWT value).

- Defined that the default action for claims that are not understood is to ignore them unless otherwise specified by applications.
- Changed from using the term "byte" to "octet" when referring to 8 bit values.
- Tracked encryption computation changes in the JWE specification.

- Changed the name of the prn claim to sub (subject) both to more closely align with SAML name usage and to use a more intuitive name.
- Allow JWTs to have multiple audiences.
- Applied editorial improvements suggested by Jeff Hodges, Prateek Mishra, and Hannes Tschofenig. Many of these simplified the terminology used.
- Explained why Nested JWTs should be signed and then encrypted.
- Clarified statements of the form "This claim is OPTIONAL" to "Use of this claim is OPTIONAL".
- Referenced String Comparison Rules in JWS.
- Added seriesInfo information to Internet Draft references.

- Updated values for example AES CBC calculations.

- Promoted Initialization Vector from being a header parameter to being a top-level JWE element. This saves approximately 16 bytes in the compact serialization, which is a significant savings for some use cases. Promoting the Initialization Vector out of the header also avoids repeating this shared value in the JSON serialization.
- Applied changes made by the RFC Editor to RFC 6749's registry language to this specification.
- Reference RFC 6755 -- An IETF URN Sub-Namespace for OAuth.

- Added statement that "StringOrURI values are compared as case-sensitive strings with no transformations or canonicalizations applied".
- Indented artwork elements to better distinguish them from the body text.

- Added an example of an encrypted JWT.
- Added this language to Registration Templates: "This name is case sensitive. Names that match other registered names in a case insensitive manner SHOULD NOT be accepted."
- Applied editorial suggestions.

- Added the cty (content type) header parameter for declaring type information about the secured content, as opposed to the typ (type) header parameter, which declares type information about this object. This significantly simplified nested JWTs.
- Moved description of how to determine whether a header is for a JWS or a JWE from the JWT spec to the JWE spec.
- Changed registration requirements from RFC Required to Specification Required with Expert Review.
- Added Registration Template sections for defined registries.
- Added Registry Contents sections to populate registry values.
- Added "Collision Resistant Namespace" to the terminology section.
- Numerous editorial improvements.

- 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  
URI: http://www.thread-safe.com/

Nat Sakimura  
Nomura Research Institute  
Email: n-sakimura@nri.co.jp  
URI: http://nat.sakimura.org/