OAuth Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 19, 2014

M. Jones
Microsoft
J. Bradley
Ping Identity
N. Sakimura
NRI

March 18, 2014

JSON Web Token (JWT)
draft-ietf-oauth-json-web-token-19

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

Copyright Notice

Table of Contents

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) [RFC7159] 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", "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 [RFC4648], with all trailing '=' characters omitted (as permitted by Section 3.2) and without the inclusion of any line breaks, white space, or other additional characters. (See Appendix C of [JWS] 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 [RFC7159], 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 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:

```
{"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:

```
{"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:

```
[123, 34, 105, 115, 115, 34, 58, 34, 106, 111, 101, 34, 44, 13, 10,
32, 34, 101, 120, 112, 34, 58, 49, 51, 48, 48, 56, 49, 57, 51, 56,
48, 44, 13, 10, 32, 34, 104, 116, 116, 112, 58, 47, 47, 101, 120, 97,
109, 112, 108, 101, 46, 99, 111, 109, 47, 105, 115, 95, 114, 111,
111, 116, 34, 58, 116, 114, 117, 101, 125]
```

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

```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly
9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```

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 [JWS], yields this encoded JWS Signature:

```
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

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
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
.
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

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 MUST 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 or implement in all cases, but rather, provide a starting point for a set of useful, interoperable claims. Applications using JWTs should define which specific claims they use and when they are required or optional. 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 StringOrURI value. Use of this claim is OPTIONAL.

4.1.3. "aud" (Audience) Claim

The "aud" (audience) claim identifies the recipients that the JWT is intended for. Each principal intended to process the JWT MUST identify itself with a value in the audience claim. If the principal processing the claim does not identify itself with a value in the "aud" claim when this claim is present, 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 JWTs 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 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 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 JWTs, 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 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, unless the application defines other specific processing rules for these Claims. 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.

Section 10.4.1 of 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:

```
{"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:

```
{"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):

```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```

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
     .
     eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
     cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
     .
```

7. Rules for Creating and Validating a JWT

To create 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.

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. 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.

5. 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.

6. 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 [RFC7159].

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 [RFC7159].

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 Parameter Name.

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. Implementation Requirements

This section defines which algorithms and features of this specification are mandatory to implement. Applications using this specification can impose additional requirements upon implementations that they use. For instance, an application might require support for encrypted JWTs and Nested JWTs; another might require support for signing JWTs with ECDSA using the P-256 curve and the SHA-256 hash algorithm ("ES256").

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.

Support for encrypted JWTs 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.

Support for Nested JWTs 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 iesg@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

- o Specification Document(s): Section 4.1.4 of [[this document]]

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

- o Claim Name: "iat"
- o Claim Description: Issued At
- o Change Controller: IESG
- o Specification Document(s): Section 4.1.6 of [[this document]]

- o Claim Name: "jti"
- o Claim Description: JWT ID
- o Change Controller: IESG
- o Specification Document(s): Section 4.1.7 of [[this document]]

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

10.2.1. Registry Contents

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

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

10.3. Media Type Registration

10.3.1. Registry Contents

This specification registers the "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.

- o Type Name: application
- o Subtype Name: jwt
- o Required Parameters: n/a
- o Optional Parameters: n/a
- o 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 ('.') characters.

- o Security Considerations: See the Security Considerations section of [[this document]]
- o Interoperability Considerations: n/a
- o Published Specification: [[this document]]
- o Applications that use this media type: OpenID Connect, Mozilla Persona, Salesforce, Google, numerous others
- o Additional Information: Magic number(s): n/a, File extension(s): n/a, Macintosh file type code(s): n/a
- o Person & email address to contact for further information: Michael B. Jones, mbj@microsoft.com
- o Intended Usage: COMMON
- o Restrictions on Usage: none
- o Author: Michael B. Jones, mbj@microsoft.com
- o Change Controller: IESG

10.4. Registration of JWE Header Parameter Names

This specification registers specific Claim Names defined in 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

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

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

- o Header Parameter Name: "aud"
- o Header Parameter Description: Audience
- o Header Parameter Usage Location(s): JWE
- o Change Controller: IESG
- o 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.

The contents of a JWT cannot be relied upon in a trust decision unless its contents have been cryptographically secured and bound to the context necessary for the trust decision. In particular, the key(s) used to sign and/or encrypt the JWT will typically need to verifiably be under the control of the party identified as the issuer of the JWT.

12. References

12.1. Normative References

[ECMAScript]
Ecma International, "ECMAScript Language Specification, 5.1 Edition", ECMA 262, June 2011.

[IANA.MediaTypes]
Internet Assigned Numbers Authority (IANA), "MIME Media Types", 2005.

[JWA] Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms (work in progress),

March 2014.

[JWE] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption (work in progress), March 2014.

[JWK] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key (work in progress), March 2014.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature (work in progress), March 2014.

[RFC2046] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types", RFC 2046, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

12.2. Informative References

[CanvasApp] Facebook, "Canvas Applications", 2010.

[JSS] Bradley, J. and N. Sakimura (editor), "JSON Simple Sign", September 2010.

[MagicSignatures] Panzer (editor), J., Laurie, B., and D. Balfanz, "Magic Signatures", January 2011.

[OASIS.saml-core-2.0-os] Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.

[RFC3275] Eastlake, D., Reagle, J., and D. Solo, "(Extensible Markup Language) XML-Signature Syntax and Processing", RFC 3275, March 2002.

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC6755] Campbell, B. and H. Tschofenig, "An IETF URN Sub-Namespace for OAuth", RFC 6755, October 2012.

[SWT] Hardt, D. and Y. Goland, "Simple Web Token (SWT)", Version 0.9.5.1, November 2009.

[W3C.CR-xml11-20021015]
Cowan, J., "Extensible Markup Language (XML) 1.1", W3C CR CR-xml11-20021015, October 2002.

[W3C.REC-xml-c14n-20010315]
Boyer, J., "Canonical XML Version 1.0", World Wide Web Consortium Recommendation REC-xml-c14n-20010315, March 2001, <http://www.w3.org/TR/2001/REC-xml-c14n-20010315>.

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:

o the Content Encryption Key is encrypted to the recipient using the RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key and

o 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:

```
  eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2In0.
  QR1Owv2ug2WyPBnbQrRARTeEk9kDO2w8qDcjiHnSJflSdv1iNqhWXaKH4MqAkQtM
  oNfABIPJaZm0HaA415sv3aeuBWnD8J-Ui7Ah6cWafs3ZwwFKDFUUsWHSK-IPKxLG
  TkND09XyjORj_CHAgOPJ-Sd8ONQRnJvWn_hXV1BNMHzUjPyYwEsRhDhzjAD26ima
  sOTsgruobpYGoQcXUwFDn7moXPRfDE8-NoQX7N7ZYMmpUDkR-Cx9obNGwJQ3nM52
  YCitxoQVPzjbl7WBuB7AohdBoZOdZ24WlN1lVIeh8v1K4krB8xgKvRU8kgFrEn_a
  1rZgN5TiysnmzTROF869lQ.
  AxY8DCtDaGlsbGljb3RoZQ.
  MKOle7UQrG6nSxTLX6Mqwt0orbHvAKeWnDYvpIAeZ72deHxz3roJDXQyhxx0wKaM
  HDjUEOKIwrtkHthpqEanSBNYHZgmNOV7sln1Eu9g3J8.
  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:

o the Content Encryption Key is encrypted to the recipient using the RSAES-PKCS1-V1_5 algorithm to produce the JWE Encrypted Key,

o the Plaintext is encrypted using the AES_128_CBC_HMAC_SHA_256 algorithm to produce the Ciphertext, and

o 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:

eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2IiwiY3R5IjoiSldUIn0

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:

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDLUhTMjU2IiwiY3R5IjoiSldU
In0.
g_hEwksO1Ax8Qn7HoN-BVeBoa8FXe0kpyk_XdcSmxvcM5_P296JXXtoHISr_DD_M
qewaQSH4dZOQHoUgKLeFly-9RI11TG-_Ge1bZFazBPwKC5lJ6OLANLMd0QSL4fYE
b9ERe-epKYE3xb2jfY1AltHqBO-PM6j23Guj2yDKnFv6WO72tteVzm_2n17SBFvh
DuR9a2nHTE67pe0XGBUS_TK7ecA-iVq5COeVdJR4U4VZGGlxRGPLRHvolVLEHx6D
YyLpw30Ay9R6d68YCLi9FYTq3hIXPK_-dmPlOUlKvPr1GgJzRoeC9G5qCvdcHWsq
JGTO_z3Wfo5zsqwkxruxwA.
UmVkbW9uZCBXQSA5ODA1Mg.
VwHERHPvCNcHHpTjkoigx3_ExK0Qc71RMEParpatm0X_qpg-w8kozSjfNIPPXiTB
BLXR65CIPkFqz4l1Ae9w_uowKiwyi9acgVztAi-pSL8GQSXnaamh9kX1mdh3M_TT
-FZGQFQsFhu0Z72gJKGdfGE-OE7hS1zuBD5oEUfk0Dmb0VzWEzpxxiSSBbBAzP10
l56pPfAtrjEYw-7ygeMkwBl6Z_mLS6w6xUgKlvW6ULmkV-uLC4FUiyKECK4e3WZY
Kw1bpgIqGYsw2v_grHjszJZ-_I5uM-9RA8ycX9KqPRp9gc6pXmoU_-27ATs9XCvr
ZXUtK2902AUzqpeEUJYjWWxSNsS-r1TJ1I-FMJ4XyAiGrfmo9hQPcNBYxPz3GQb2
8Y5CLSQfNgKSGt0A4isp1hBUXBHAndgtcslt7ZoQJaKe_nNJgNliWtWpJ_ebuOpE
l8jdhehdccnRMIwAmU1n7SPkmhIl1HlSOpvcvDfhUN5wuqU955vOBvfkBOh5A11U
zBuo2WlgZ6hYi9-e3w29bR0C2-pp3jbqxEDw3iWaf2dc5b-LnR0FEYXvI_tYk5rd
_J9N0mg0tQ6RbpxNEMNoA9QWk5lgdPvbh9BaO195abQ.
AVO9iT5AV4CzvDJCdhSFlQ
```

Appendix B. **Relationship of JWTs to SAML Assertions**

SAML 2.0 [OASIS.saml-core-2.0-os] 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 [W3C.CR-xml11-20021015] and XML DSIG [RFC3275] contributes to the size of SAML assertions; its use of XML and especially XML Canonicalization [W3C.REC-xml-c14n-20010315] 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 [RFC7159] 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 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 signature, MAC, and encryption algorithms.

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:

Dirk Balfanz, Richard Barnes, Brian Campbell, Breno de Medeiros, Dick Hardt, Joe Hildebrand, Jeff Hodges, Edmund Jay, Yaron Y. Goland, Ben Laurie, James Manger, Prateek Mishra, Tony Nadalin, Axel Nennker, John Panzer, Emmanuel Raviart, David Recordon, Eric Rescorla, Jim Schaad, Paul Tarjan, Hannes Tschofenig, and Sean Turner.

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]]

-19

- Specified that support for Nested JWTs is optional and that applications using this specification can impose additional requirements upon implementations that they use.

- Updated the JSON reference to RFC 7159.

-18

- Clarified that the base64url encoding includes no line breaks, white space, or other additional characters.

- Removed circularity in the audience claim definition.

- Clarified that it is entirely up to applications which claims to use.

- Changed "SHOULD" to "MUST" in "in the absence of such requirements, all claims that are not understood by implementations MUST be ignored".

- Clarified that applications can define their own processing rules for claims replicated in header parameters, rather than always requiring that they be identical in the JWT Header and JWT Claims Set.

- Removed a JWT creation step that duplicated a step in the underlying JWS or JWE creation.

- Added security considerations about using JWTs in trust decisions.

-17

- Corrected RFC 2119 terminology usage.

- Replaced references to draft-ietf-json-rfc4627bis with RFC 7158.

-16

- Changed some references from being normative to informative, per JOSE issue #90.

-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

o Clarified that the "typ" header parameter is used in an application-specific manner and has no effect upon the JWT processing.

o 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.

-08

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

-07

o Defined that the default action for claims that are not understood is to ignore them unless otherwise specified by applications.

o Changed from using the term "byte" to "octet" when referring to 8 bit values.

o Tracked encryption computation changes in the JWE specification.

-06

o 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.

o Allow JWTs to have multiple audiences.

o Applied editorial improvements suggested by Jeff Hodges, Prateek Mishra, and Hannes Tschofenig. Many of these simplified the terminology used.

o Explained why Nested JWTs should be signed and then encrypted.

o Clarified statements of the form "This claim is OPTIONAL" to "Use of this claim is OPTIONAL".

o Referenced String Comparison Rules in JWS.

o Added seriesInfo information to Internet Draft references.

-05

- Updated values for example AES CBC calculations.

-04

- 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.

-03

- 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.

-02

- 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.

-01

- 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.

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

Nat Sakimura
Nomura Research Institute

Email: n-sakimura@nri.co.jp
URI: http://nat.sakimura.org/