OAuth Working Group	M. Jones
Internet-Draft	Microsoft
Intended status: Standards Track	J. Bradley
Expires: January 31, 2013	Ping Identity
	N. Sakimura
	NRI
	July 30, 2012

TOC

JSON Web Token (JWT)
draft-ietf-oauth-json-web-token-03

Abstract

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

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

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 January 31, 2013.

Table of Contents

TOC

1. Introduction

JSON Web Token (JWT) is a compact token format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. JWTs encode claims to be transmitted as a JavaScript Object Notation (JSON) **[RFC4627]** object that is base64url encoded and digitally signed or MACed and/or encrypted. Signing and MACing is performed using JSON Web Signature (JWS) **[JWS]**. Encryption is performed using JSON Web Encryption (JWE) **[JWE]**.

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

TOC

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]**.

TOC

2. Terminology

JSON Web Token (JWT)
: A string representing a set of claims as a JSON object that is digitally signed or

MACed and/or encrypted. The string consists of multiple parts, the first being the Encoded JWT Header, plus additional parts depending upon the contents of the header, with the parts being separated by period ('.') characters, and each part containing base64url encoded content.

Base64url Encoding
: The URL- and filename-safe Base64 encoding described in **RFC 4648** [RFC4648], Section 5, with the (non URL-safe) '=' padding characters omitted, as permitted by Section 3.2. (See Appendix C of **[JWS]** for notes on implementing base64url encoding without padding.)

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

Header Parameter Name
: The name of a member of the JSON object representing a JWT Header.

Header Parameter Value
: The value of a member of the JSON object representing a JWT Header.

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

Claim Name
: The name of a member of the JSON object representing a JWT Claims Set.

Claim Value
: The value of a member of the JSON object representing a JWT Claims Set.

Encoded JWT Header
: Base64url encoding of the bytes of the UTF-8 **[RFC3629]** representation of the JWT Header.

Collision Resistant Namespace
: A namespace that allows names to be allocated in a manner such that they are highly unlikely to collide with other names. For instance, collision resistance can be achieved through administrative delegation of portions of the namespace or through use of collision-resistant name allocation functions. 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.

TOC

3. JSON Web Token (JWT) Overview

JWTs represent a set of claims as a JSON object that is base64url encoded and digitally signed or MACed and/or encrypted. The JWT Claims Set represents this JSON object. As per **RFC 4627** [RFC4627] Section 2.2, the JSON object consists of zero or more name/value pairs (or members), where the names are strings and the values are arbitrary JSON values. These members are the claims represented by the JWT.

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

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

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

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

TOC

3.1. Example JWT

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

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

Base64url encoding the bytes of the UTF-8 representation of the JWT Header yields this Encoded JWS Header value, which is used as the Encoded JWT Header:

```
eyJ0eXAiOiJKV1QiLA0KICJhbGciOiJIUzI1NiJ9
```

The following is an example of a JWT Claims Set:

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

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

```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly
9leGFtcGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```

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

```
dBjftJeZ4CVP-mB92K27uhbUJU1p1r_wW1gFWFOEjXk
```

Concatenating these 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 **[JWS]**, Appendix A.1. See **Appendix A** for an example of an encrypted JWT.

---

4. JWT Claims

TOC

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

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

4.1. Reserved Claim Names

TOC

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

4.1.1. "exp" (Expiration Time) Claim

TOC

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

4.1.2. "nbf" (Not Before) Claim

TOC

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

4.1.3. "iat" (Issued At) Claim

TOC

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

4.1.4. "iss" (Issuer) Claim

TOC

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. This claim is OPTIONAL.

4.1.5. "aud" (Audience) Claim

TOC

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

TOC

4.1.6. "prn" (Principal) Claim

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

TOC

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. This claim is OPTIONAL.

TOC

4.1.8. "typ" (Type) Claim

The typ (type) claim is used to declare a type for the contents of this JWT Claims Set. The typ value is a case sensitive string. This claim is OPTIONAL.

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

TOC

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 **Section 9.1** or be a URI that contains a Collision Resistant Namespace.

TOC

4.3. Private Claim Names

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

TOC

5. JWT Header

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

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

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

TOC

The `typ` (type) header parameter is used to declare the type of this object. If present, it is RECOMMENDED that its value be either "JWT" or "urn:ietf:params:oauth:token-type:jwt" to indicate that this object is a JWT. The `typ` value is a case sensitive string. This header parameter is OPTIONAL.

5.2. "cty" (Content Type) Header Parameter

TOC

The `cty` (content type) header parameter is used to declare structural information about the JWT. Its value MUST be a string.

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, the use of this header parameter is REQUIRED; in this case, the value MUST be "JWT", to indicate that a nested JWT is carried in this JWT.

The values used for the `cty` header parameter come from the same value space as the `typ` header parameter, with the same rules applying.

---

6. Plaintext JWTs

TOC

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

6.1. Example Plaintext JWT

TOC

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

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

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

```
eyJhbGciOiJub25lIn0
```

The following is an example of a JWT Claims Set:

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

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

```
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
```

The Encoded JWS Signature is the empty string.

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

```
eyJhbGciOiJub25lIn0
.
eyJpc3MiOiJqb2UiLA0KICJleHAiOjEzMDA4MTkzODAsDQogImh0dHA6Ly9leGFt
cGxlLmNvbS9pc19yb290Ijp0cnVlfQ
.
```

---

7. Rules for Creating and Validating a JWT

TOC

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

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

1. The JWT MUST contain at least one period character.
2. Let the Encoded JWT Header be the portion of the JWT before the first period character.
3. The Encoded JWT Header MUST be successfully base64url decoded following the restriction given in this specification that no padding characters have been used.
4. The resulting JWT Header MUST be completely valid JSON syntax conforming to **RFC 4627** [RFC4627].
5. The resulting JWT Header MUST be validated to only include parameters and values whose syntax and semantics are both understood and supported.
6. Determine whether the JWT is a JWS or a JWE by examining the `alg` (algorithm) header value and optionally, the `enc` (encryption method) header value, if present.
7. Depending upon whether the JWT is a JWS or JWE, there are two cases:
    - If the JWT is a JWS, all steps specified in **[JWS]** for validating a JWS MUST be followed. Let the Message be the result of

base64url decoding the JWS Payload.
    - Else, if the JWT is a JWE, all steps specified in **[JWE]** for validating a JWE MUST be followed. Let the Message be the JWE Plaintext.
8. If the JWT Header contains a `cty` (content type) value of "JWT", then the Message contains 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 **RFC 4627** [RFC4627].
11. When used in a security-related context, the JWT Claims Set MUST be validated to only include claims whose syntax and semantics are both understood and supported.

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

Comparisons between JSON strings and other Unicode strings MUST be performed as specified below:

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

---

8. Cryptographic Algorithms

TOC

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

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

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

---

9. IANA Considerations

TOC

9.1. JSON Web Token Claims Registry

TOC

This specification establishes the IANA JSON Web Token Claims registry for reserved JWT Claim Names. The registry records the reserved 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 RFC-EDITOR: The name of the mailing list should be determined in consultation with the IESG and IANA. Suggested name: claims-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.

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

TOC

9.1.1. Registration Template

Claim Name:
: The name requested (e.g., "example"). This name is case sensitive. Names that match other registered names in a case insensitive manner SHOULD NOT be accepted.

Change Controller:
: For standards-track RFCs, state "IETF". For others, give the name of the responsible party. Other details (e.g., postal address, e-mail address, home page URI) may also be included.

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

TOC

9.1.2. Initial Registry Contents

- Claim Name: `exp`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.1** of [[this document]]

- Claim Name: `nbf`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.2** of [[this document]]

- Claim Name: `iat`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.3** of [[this document]]

- Claim Name: `iss`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.4** of [[this document]]

- Claim Name: `aud`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.5** of [[this document]]

- Claim Name: `prn`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.6** of [[this document]]

- Claim Name: `jti`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.7** of [[this document]]

- Claim Name: `typ`
- Change Controller: IETF
- Specification Document(s): **Section 4.1.8** of [[this document]]

TOC

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

TOC

9.2.1. Registry Contents

TOC

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** [I-D.ietf-oauth-urn-sub-ns].

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

9.3. JSON Web Signature and Encryption Type Values Registration

TOC

9.3.1. Registry Contents

TOC

This specification registers the `JWT` type value in the IANA JSON Web Signature and Encryption Type Values registry **[JWS]**:

- "typ" Header Parameter Value: `JWT`
- Abbreviation for MIME Type: application/jwt
- Change Controller: IETF
- Specification Document(s): **Section 5.1** of [[this document]]

9.4. Media Type Registration

TOC

9.4.1. Registry Contents

TOC

This specification registers the `application/jwt` Media Type **[RFC2046]** in the MIME Media Type registry **[RFC4288]** to indicate that the content is a JWT.

- Type Name: application
- Subtype Name: jwt
- Required Parameters: n/a
- Optional Parameters: n/a
- Encoding considerations: JWT values are encoded as a series of base64url encoded values (some of which may be the empty string) separated by period ('.') 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 Browser ID, 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: IETF

10. Security Considerations

TOC

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 key, 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, but some significant concerns are listed here.

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

11. Open Issues

TOC

[[to be removed by the RFC editor before publication as an RFC]]

The following items remain to be considered or done in this draft:

- Track changes to the underlying JOSE specifications.

12. References

TOC

12.1. Normative References

TOC

[I-D.ietf-oauth-urn-sub-ns]	Campbell, B. and H. Tschofenig, "**An IETF URN Sub-Namespace for OAuth**," draft-ietf-oauth-urn-sub-ns-06 (work in progress), July 2012 (**TXT**).
[JWA]	**Jones, M.**, "**JSON Web Algorithms (JWA)**," July 2012.
[JWE]	**Jones, M.**, **Rescorla, E.**, and **J. Hildebrand**, "**JSON Web Encryption (JWE)**," July 2012.
[JWS]	**Jones, M.**, **Bradley, J.**, and **N. Sakimura**, "**JSON Web Signature (JWS)**," July 2012.
[RFC2046]	**Freed, N.** and **N. Borenstein**, "**Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types**," RFC 2046, November 1996 (**TXT**).
[RFC2119]	**Bradner, S.**, "**Key words for use in RFCs to Indicate Requirement Levels**," BCP 14, RFC 2119, March 1997 (**TXT**, **HTML**, **XML**).
[RFC3339]	**Klyne, G., Ed.** and **C. Newman**, "**Date and Time on the Internet: Timestamps**," RFC 3339, July 2002 (**TXT**, **HTML**, **XML**).
[RFC3629]	Yergeau, F., "**UTF-8, a transformation format of ISO 10646**," STD 63, RFC 3629, November 2003 (**TXT**).
[RFC3986]	**Berners-Lee, T.**, **Fielding, R.**, and **L. Masinter**, "**Uniform Resource Identifier (URI): Generic Syntax**," STD 66, RFC 3986, January 2005 (**TXT**, **HTML**, **XML**).
[RFC4288]	Freed, N. and J. Klensin, "**Media Type Specifications and Registration Procedures**," BCP 13, RFC 4288, December 2005 (**TXT**).
[RFC4627]	Crockford, D., "**The application/json Media Type for JavaScript Object Notation (JSON)**," RFC 4627, July 2006 (**TXT**).
[RFC4648]	Josefsson, S., "**The Base16, Base32, and Base64 Data Encodings**," RFC 4648, October 2006 (**TXT**).
[RFC5226]	Narten, T. and H. Alvestrand, "**Guidelines for Writing an IANA Considerations Section in RFCs**," BCP 26, RFC 5226, May 2008 (**TXT**).
[USA15]	**Davis, M.**, **Whistler, K.**, and M. Dürst, "Unicode Normalization Forms," Unicode Standard Annex 15, 09 2009.

12.2. Informative References

TOC

[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-	**Cantor, S.**, **Kemp, J.**, **Philpott, R.**, and **E. Maler**, "**Assertions and Protocol for the OASIS Security**

2.0-os]	**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 (**TXT**).
[RFC4122]	**Leach, P.**, **Mealling, M.**, and **R. Salz**, "**A Universally Unique IDentifier (UUID) URN Namespace**," RFC 4122, July 2005 (**TXT**, **HTML**, **XML**).
[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.

TOC

Appendix A. 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 CBC. AES CBC does not have an integrated integrity check, so a separate integrity check calculation is performed using HMAC SHA-256, with separate encryption and integrity keys being derived from a master key using the Concat KDF with the SHA-256 digest function.

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

- the Content Master 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 CBC algorithm with a 128 bit key to produce the Ciphertext,
- the JWE Integrity Value safeguarding the integrity of the Ciphertext and the parameters used to create it was computed with the HMAC SHA-256 algorithm, and
- the 128 bit Initialization Vector (IV) with the base64url encoding `AxY8DCtDaGlsbGljb3RoZQ` was used.

```
{"alg":"RSA1_5","enc":"A128CBC","int":"HS256","iv":"AxY8DCtDaGls
bGljb3RoZQ"}
```

Other than using the bytes of the UTF-8 representation of the JSON 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:

```
eyJhbGciOiJSU0ExXzUiLCJlbmMiOiJBMTI4Q0JDIiwiaW50IjoiSFMyNTYiLCJp
diI6IkF4WThEQ3REYUdsc2JHbGpiM1JvWlEifQ.
VjBkk22MjrFUMUl8ItbS8CjKjku4HQz4RiHD0eVG4dir-7XbDkPr1q6YtnN1X-av
1EKmEnsrbhSxTvqtY4oEbWKLoEQ7zVm_0BW-rnwxdwrj4QJrhXGnqIL6bC4waZVJ
qYhVQIahVWSQsCRcS1oYXA-2GhT6rk91y118DUkhGDsvdK2_hQsNGE6BQVN1i-Xw
Uoz5sM6_0PRQ1FsYnJATMjVZfa4otHiooZ_KcOlSWIDxhMDqfPOu60--1ej0eZBy
O7Ar_IZvzPAWqJ9agGFQIVGRZviXhN0WeErq9fVTcgeSUPsmurRSTYhTrNFLojqP
qqk8pI61kn8GmZxA80-RUQ.
7kLQQst655TUxmDzysjRLXnD-nfLK5EQK7ODAUkwxc0aRb9NOgu0EMJgOR6Vz8eN
baf8six_OP6BRyUTYrCkH73-inD6Rc-7vc9eC5fcfSM.
COyXNSm-CdfAL22WIKcoyCgQwb85aLW3ttDkzNj_1Wg
```

TOC

Appendix B. Relationship of JWTs to SAML Tokens

SAML 2.0 [OASIS.saml-core-2.0-os] provides a standard for creating tokens with much greater expressivity and more security options than supported by JWTs. However, the cost of this flexibility and expressiveness is both size and complexity. In addition, SAML's use of **XML** [W3C.CR-xml11-20021015] and **XML DSIG** [RFC3275] only contributes to the size of SAML tokens.

JWTs are intended to provide a simple token format that is small enough to fit into HTTP headers and query arguments in URIs. It does this by supporting a much simpler token model than SAML and using the **JSON** [RFC4627] object encoding syntax. It also supports

securing tokens using Message Authentication Codes (MACs) and digital signatures using a smaller (and less flexible) format than XML DSIG.

Therefore, while JWTs can do some of the things SAML tokens do, JWTs are not intended as a full replacement for SAML tokens, but rather as a compromise token format to be used when space is at a premium.

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

TOC

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 HMAC SHA-256, RSA SHA-256, and ECDSA P-256 SHA-256.

Appendix D. Acknowledgements

TOC

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

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

Appendix E. Document History

TOC

[[to be removed by the RFC editor before publication as an RFC]]

-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

TOC

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

John Bradley
Ping Identity
Email: ve7jtb@ve7jtb.com

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