Network Working Group M. Jones
Internet-Draft Microsoft
Intended status: Standards Track D. Balfanz
Expires: November 13, 2012 Google
J. Bradley
Ping Identity
Y. Goland
Microsoft
J. Panzer
Google
N. Sakimura
NRI
P. Tarjan
Facebook
May 12, 2012

JSON Web Token (JWT)
draft-jones-json-web-token-10

Abstract

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

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

Requirements Language

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

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

JSON Web Token (JWT) is a compact token format intended for space constrained environments such as HTTP Authorization headers and URI query parameters. JWTs encode claims to be transmitted as a JSON object (as defined in RFC 4627 [RFC4627]) that is base64url encoded and digitally signed or MACed and/or encrypted. Signing and MACing is performed using JSON Web Signature (JWS) [JWS]. Encryption is performed using JSON Web Encryption (JWE) [JWE].

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

2. Terminology

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

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

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

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

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

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

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

Encoded JWT Header Base64url encoding of the bytes of the UTF-8 RFC 3629 [RFC3629] representation of the JWT Header.

Base64url Encoding For the purposes of this specification, this term always refers to 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.)

StringOrURI A JSON string value, with the additional requirement that while arbitrary string values MAY be used, any value containing a ":" character MUST be a URI as defined in RFC 3986 [RFC3986].

IntDate A JSON numeric value representing the number of seconds from 1970-01-01T0:0:0Z UTC until the specified UTC date/time. See 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 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.

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.

4. JWT Claims

The JWT Claims Set represents a JSON object whose members are the claims conveyed by the JWT. The Claim Names within this object MUST be unique; JWTs with duplicate Claim Names MUST be rejected. Note

however, that the set of claims that a JWT must contain to be considered valid is context-dependent and is outside the scope of this specification. When used in a security-related context, implementations MUST understand and support all of the claims present; otherwise, the JWT MUST be rejected for processing.

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

4.1. Reserved Claim Names

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

4.1.1. "exp" (Expiration Time) Claim

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

4.1.2. "nbf" (Not Before) Claim

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

4.1.3. "iat" (Issued At) Claim

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

4.1.4. "iss" (Issuer) Claim

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

4.1.5. "aud" (Audience) Claim

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

4.1.6. "prn" (Principal) Claim

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

4.1.7. "jti" (JWT ID) Claim

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

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 case sensitive. Its value MUST be a string. This claim is OPTIONAL.

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

4.2. Public Claim Names

Claim names can be defined at will by those using JWTs. However, in order to prevent collisions, any new claim name SHOULD either be

defined in the IANA JSON Web Token Claims registry Section 9.1 or be a URI that contains a collision resistant namespace. Examples of collision resistant namespaces include:

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

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

4.3. Private Claim Names

A producer and consumer of a JWT may agree to any claim name that is not a Reserved Name 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.

5. JWT Header

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

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

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

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

6. Plaintext JWTs

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

6.1. Example Plaintext JWT

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

```
{"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

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

1. Create a JWT Claims Set containing the desired claims. Note that white space is explicitly allowed in the representation and no canonicalization is performed before encoding.

2. Let the Message be the bytes of the UTF-8 representation of the JWT Claims Set.

3. Create a JWT Header containing the desired set of header parameters. The JWT MUST conform to either the [JWS] or [JWE] specifications. Note that white space is explicitly allowed in the representation and no canonicalization is performed before encoding.

4. Base64url encode the bytes of the UTF-8 representation of the JWT Header. Let this be the Encoded JWT Header.

5. Depending upon whether the JWT is a JWS or JWE, there are two cases:

   * If the JWT is a JWS, create a JWS using the JWT Header as the JWS Header and the Message as the JWS Payload; all steps specified in [JWS] for creating a JWS MUST be followed.

   * Else, if the JWT is a JWE, create a JWE using the JWT Header as the JWE Header and the Message as the JWE Plaintext; all steps specified in [JWE] for creating a JWE MUST be followed.

6. If a nested signing or encryption operation will be performed, let the Message be the JWS or JWE, and return to Step 3, using a "typ" value of either "JWS" or "JWE" respectively in the new JWT Header created in that step.

7. Otherwise, let the resulting JWT be the JWS or JWE.

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

1. The JWT MUST contain at least one period character.

2. Let the Encoded JWT Header be the portion of the JWT before the first period character.

3. The Encoded JWT Header MUST be successfully base64url decoded following the restriction given in this specification that no padding characters have been used.

4. The resulting JWT Header MUST be completely valid JSON syntax conforming to RFC 4627 [RFC4627].

5. The resulting JWT Header MUST be validated to only include parameters and values whose syntax and semantics are both understood and supported.

6. Determine whether the JWT is a JWS or a JWE by examining the "alg" (algorithm) header value and optionally, the "enc" (encryption method) header value, if present.

7. Depending upon whether the JWT is a JWS or JWE, there are two cases:

   - If the JWT is a JWS, all steps specified in [JWS] for validating a JWS MUST be followed. Let the Message be the result of base64url decoding the JWS Payload.

   - Else, if the JWT is a JWE, all steps specified in [JWE] for validating a JWE MUST be followed. Let the Message be the JWE Plaintext.

8. If the JWT Header contains a "typ" value of either "JWS" or "JWE", then the Message contains a JWT that was the subject of nested signing or encryption operations, respectively. In this case, return to Step 1, using the Message as the JWT.

9. Otherwise, let the JWT Claims Set be the Message.

10. The JWT Claims Set MUST be completely valid JSON syntax conforming to RFC 4627 [RFC4627].

11. When used in a security-related context, the JWT Claims Set MUST be validated to only include claims whose syntax and semantics are both understood and supported.

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

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

1. Remove any JSON applied escaping to produce an array of Unicode code points.

2. Unicode Normalization [USA15] MUST NOT be applied at any point to either the JSON string or to the string it is to be compared against.

3. Comparisons between the two strings MUST be performed as a Unicode code point to code point equality comparison.

8. Cryptographic Algorithms

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

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

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

9. IANA Considerations

9.1. JSON Web Token Claims Registry

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

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

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

- URN: urn:ietf:params:oauth:token-type:jwt

- Common Name: JSON Web Token (JWT) Token Type

- Change controller: IETF

- Description: [[this document]]

9.3. Registration of application/jwt MIME Media Type

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

Type name:
application

Subtype name:
jwt

Required parameters:
n/a

Optional parameters:
n/a

Encoding considerations:
n/a

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:
Michael B. Jones
mbj@microsoft.com

9.4. Registration of "JWT" Type Value

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

"typ" header parameter value:
"JWT"

Abbreviation for MIME type:
application/jwt

Change controller:
Michael B. Jones
mbj@microsoft.com

Description:
[[this document]]

10. Security Considerations

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

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

The following items remain to be done in this draft:

- Provide an example of an encrypted JWT.

12. References

12.1. Normative References

[I-D.ietf-oauth-urn-sub-ns]
Tschofenig, H., "An IETF URN Sub-Namespace for OAuth", draft-ietf-oauth-urn-sub-ns-02 (work in progress), January 2012.

[JWA] Jones, M., "JSON Web Algorithms (JWA)", May 2012.

[JWE] Jones, M., Rescorla, E., and J. Hildebrand, "JSON Web Encryption (JWE)", May 2012.

[JWS] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", May 2012.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

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

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

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

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

[USA15] Davis, M., Whistler, K., and M. Duerst, "Unicode Normalization Forms", Unicode Standard Annex 15, 09 2009.

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.

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

[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.

Appendix A. Relationship of JWTs to SAML Tokens

SAML 2.0 [OASIS.saml-core-2.0-os] provides a standard for creating tokens with much greater expressivity and more security options than supported by JWTs. However, the cost of this flexibility and expressiveness is both size and complexity. In addition, SAML's use of XML [W3C.CR-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 B. 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 HMAC SHA-256, RSA SHA-256, and ECDSA P-256 SHA-256.

Appendix C. Acknowledgements

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

Solutions for signing JSON content were previously explored by Magic

Signatures [MagicSignatures], JSON Simple Sign [JSS], and Canvas Applications [CanvasApp], all of which influenced this draft.

Appendix D. Document History

-10

- Clarified the relationship between "typ" header parameter values, "typ" claim values, and MIME types.
- Clarified that JWTs with duplicate Header Parameter Names or Duplicate Claim names MUST be rejected.
- Required implementation of AES-128-KW and AES-256-KW when the implementation provides encryption capabilities.
- Registered "JWT" typ header parameter value.
- Generalized language to refer to Message Authentication Codes (MACs) rather than Hash-based Message Authentication Codes (HMACs) unless in a context specific to HMAC algorithms.
- Reformatted to give each claim definition and header parameter its own section heading.

-09

- Changed "http://openid.net/specs/jwt/1.0" to "urn:ietf:params:oauth:token-type:jwt" in preparation for OAuth WG draft.

-08

- Removed language that required that a JWT must have three parts. Now the number of parts is explicitly dependent upon the representation of the underlying JWS or JWE.
- Moved the "alg":"none" definition to the JWS spec.
- Registered the "application/jwt" MIME Media Type.
- Clarified that the order of the creation and validation steps is not significant in cases where there are no dependencies between the inputs and outputs of the steps.
- Corrected the Magic Signatures and Simple Web Token (SWT) references.

-07

- Defined the "prn" (principal) claim to identify the subject of the JWT.

- Defined the "jti" (JWT ID) claim to enable replay protection.

- Use the term "JWT Claims Set" rather than "JWT Claims Object" since this is actually a string representing a JSON object and not the JSON object itself.

- Moved "MUST" requirements from the Overview to later in the spec.

- Respect line length restrictions in examples.

- Applied other editorial improvements.

-06

- Reference and use content from [JWS] and [JWE], rather than repeating it here.

- Simplified terminology to better match JWE, where the terms "JWT Header" and "Encoded JWT Header" are now used, for instance, rather than the previous terms "Decoded JWT Header Segment" and "JWT Header Segment". Also changed to "Plaintext JWT" from "Unsigned JWT".

- Describe how to perform nested encryption and signing operations.

- Changed "integer" to "number", since that is the correct JSON type.

- Changed StringAndURI to StringOrURI.

-05

- Added the "nbf" (not before) claim and clarified the meaning of the "iat" (issued at) claim.

-04

- Correct typo found by John Bradley: "the JWT Claim Segment is the empty string" -> "the JWT Crypto Segment is the empty string".

-03

- Added "http://openid.net/specs/jwt/1.0" as a token type identifier URI for JWTs.

- Added "iat" (issued at) claim.

- Changed RSA SHA-256 from MUST be supported to RECOMMENDED that it be supported. Rationale: Several people have objected to the requirement for implementing RSA SHA-256, some because they will only be using HMACs and symmetric keys, and others because they only want to use ECDSA when using asymmetric keys, either for security or key length reasons, or both.

- Defined "alg" value "none" to represent unsigned JWTs.

-02

- Split signature specification out into separate draft-jones-json-web-signature-00. This split introduced no semantic changes.

- The JWT Compact Serialization is now the only token serialization format specified in this draft. The JWT JSON Serialization can continue to be defined in a companion specification.

-01

- Draft incorporating consensus decisions reached at IIW.

-00

- Public draft published before November 2010 IIW based upon the JSON token convergence proposal incorporating input from several implementers of related specifications.

Authors' Addresses

Michael B. Jones
Microsoft

Email: mbj@microsoft.com
URI: http://self-issued.info/

Dirk Balfanz
Google

Email: balfanz@google.com

John Bradley
Ping Identity

Email: ve7jtb@ve7jtb.com

Yaron Y. Goland
Microsoft

Email: yarong@microsoft.com

John Panzer
Google

Email: jpanzer@google.com

Nat Sakimura
Nomura Research Institute

Email: n-sakimura@nri.co.jp

Paul Tarjan
Facebook

Email: pt@fb.com