

OAuth Working Group Y.
Sheffer
Internet-Draft
Intuit
Intended status: Best Current Practice D.
Hardt
Expires: January 20, 2018
Amazon M.
Jones
Microsoft
2017 July 19,

**JSON Web Token Best Current Practices
draft-ietf-oauth-jwt-bcp-00**

Abstract

JSON Web Tokens, also known as JWTs [[RFC7519](#)], are URL-safe JSON-based security tokens that contain a set of claims that can be signed and/or encrypted. JWTs are being widely used and deployed as a simple security token format in numerous protocols and applications, both in the area of digital identity, and in other application areas.

The goal of this Best Current Practices document is to provide actionable guidance leading to secure implementation and deployment of JWTs.

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 20, 2018.

Copyright Notice

Copyright (c) 2017 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal

Provisions Relating to IETF Documents
(<http://trustee.ietf.org/license-info>) in effect on the date of

Sheffer, et al.
1]

Expires January 20, 2018

[Page

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

- [1.](#) Introduction
[2](#)
- [1.1.](#) Target Audience
[3](#)
- [1.2.](#) Conventions used in this document
[4](#)
- [2.](#) Threats and Vulnerabilities
[4](#)
- [2.1.](#) Weak Signatures and Insufficient Signature Validation
[4](#)
- [2.2.](#) Weak symmetric keys
[4](#)
- [2.3.](#) Multiplicity of JSON encodings
[4](#)
- [2.4.](#) Incorrect Composition of Encryption and Signature
[5](#)
- [2.5.](#) Insecure Use of Elliptic Curve Encryption
[5](#)
- [2.6.](#) Substitution Attacks
[5](#)
- [2.7.](#) Cross-JWT Confusion
[5](#)
- [3.](#) Best Practices
[5](#)
- [3.1.](#) Perform Algorithm Verification
[6](#)
- [3.2.](#) Use Appropriate Algorithms
[6](#)
- [3.3.](#) Validate All Cryptographic Operations
[6](#)
- [3.4.](#) Validate Cryptographic Inputs
[6](#)
- [3.5.](#) Ensure Cryptographic Keys have Sufficient Entropy
[7](#)
- [3.6.](#) Use UTF-8
[7](#)
- [3.7.](#) Validate Issuer and Subject
[7](#)
- [3.8.](#) Use and Validate Audience
[7](#)
- [3.9.](#) Use Explicit Typing
[8](#)

	3.10. Use Mutually Exclusive Validation Rules for Different Kinds of JWTs	8
<u>4.</u>	IANA Considerations	9
<u>5.</u>	Acknowledgements	9
<u>6.</u>	References	9
	<u>6.1.</u> Normative References	9
	<u>6.2.</u> Informative References	10
<u>Appendix A.</u>	Document History	11
	<u>A.1.</u> draft-ietf-oauth-jwt-bcp-00	11
	<u>A.2.</u> draft-sheffer-oauth-jwt-bcp-01	11
	<u>A.3.</u> draft-sheffer-oauth-jwt-bcp-00	11
	Authors' Addresses	11

1. Introduction

JSON Web Tokens, also known as JWTs [[RFC7519](#)], are URL-safe JSON-based security tokens that contain a set of claims that can be signed and/or encrypted. The JWT specification has seen rapid adoption

because it encapsulates security-relevant information in one, easy to protect location, and because it is easy to implement using widely-available tools. One application area in which JWTs are commonly used is representing digital identity information, such as OpenID Connect ID Tokens [[OpenID.Core](#)] and OAuth 2.0 [[RFC6749](#)] access tokens and refresh tokens, the details of which are deployment-specific.

Since the JWT specification was published, there have been several widely published attacks on implementations and deployments. Such attacks are the result of under-specified security mechanisms, as well as incomplete implementations and incorrect usage by applications.

The goal of this document is to facilitate secure implementation and deployment of JWTs. Many of the recommendations in this document will actually be about implementation and use of the cryptographic mechanisms underlying JWTs that are defined by JSON Web Signature (JWS) [[RFC7515](#)], JSON Web Encryption (JWE) [[RFC7516](#)], and JSON Web Algorithms (JWA) [[RFC7518](#)]. Others will be about use of the JWT claims themselves.

These are intended to be minimum recommendations for the use of JWTs in the vast majority of implementation and deployment scenarios. Other specifications that reference this document can have stricter requirements related to one or more aspects of the format, based on their particular circumstances; when that is the case, implementers are advised to adhere to those stricter requirements. Furthermore, this document provides a floor, not a ceiling, so stronger options are always allowed (e.g., depending on differing evaluations of the importance of cryptographic strength vs. computational load).

Community knowledge about the strength of various algorithms and feasible attacks can change quickly, and experience shows that a Best

Current Practice (BCP) document about security is a point-in-time statement. Readers are advised to seek out any errata or updates that apply to this document.

1.1. Target Audience

The targets of this document are:

- Implementers of JWT libraries (and the JWS and JWE libraries used by them),
- Implementers of code that uses such libraries (to the extent that some mechanisms may not be provided by libraries, or until they are), and

- Developers of specifications that rely on JWTs, both inside and outside the IETF.

1.2. Conventions used in this document

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 [[RFC2119](#)].

2. Threats and Vulnerabilities

This section lists some known and possible problems with JWT implementations and deployments. Each problem description is followed by references to one or more mitigations to those problems.

2.1. Weak Signatures and Insufficient Signature Validation

Signed JSON Web Tokens carry an explicit indication of the signing algorithm, in the form of the "alg" header parameter, to facilitate cryptographic agility. This, in conjunction with design flaws in some libraries and applications, have led to several attacks:

- The algorithm can be changed to "none" by an attacker, and some libraries would trust this value and "validate" the JWT without checking any signature.
- An "RS256" (RSA, 2048 bit) parameter value can be changed into "HS256" (HMAC, SHA-256), and some libraries would try to validate the signature using HMAC-SHA256 and using the RSA public key as the HMAC shared secret.

For mitigations, see [Section 3.1](#) and [Section 3.2](#).

2.2. Weak symmetric keys

In addition, some applications sign tokens using a weak symmetric key and a keyed MAC algorithm such as "HS256". In most cases, these keys are human memorable passwords that are vulnerable to dictionary attacks [[Langkemper](#)].

For mitigations, see [Section 3.5](#).

2.3. Multiplicity of JSON encodings

Many practitioners are not aware that JSON [[RFC7159](#)] allows several different character encodings: UTF-8, UTF-16 and UTF-32. As a result, the JWT might be misinterpreted by its recipient.

For mitigations, see [Section 3.6](#).

2.4. Incorrect Composition of Encryption and Signature

Some libraries that decrypt a JWE-encrypted JWT to obtain a JWS-signed object do not always validate the internal signature.

For mitigations, see [Section 3.3](#).

2.5. Insecure Use of Elliptic Curve Encryption

Per [[Sanso](#)], several JOSE libraries fail to validate their inputs correctly when performing elliptic curve key agreement (the "ECDH-ES"

algorithm). An attacker that is able to send JWEs of its choosing that use invalid curve points and observe the cleartext outputs resulting from decryption with the invalid curve points can use this vulnerability to recover the recipient's private key.

For mitigations, see [Section 3.4](#).

2.6. Substitution Attacks

There are attacks in which one recipient will have a JWT intended for

it and attempt to use it at a different recipient that it was not intended for. If not caught, these attacks can result in the attacker gaining access to resources that it is not entitled to access.

For mitigations, see [Section 3.7](#) and [Section 3.8](#).

2.7. Cross-JWT Confusion

As JWTs are being used by more different protocols in diverse application areas, it becomes increasingly important to prevent cases

of JWT tokens that have been issued for one purpose being subverted and used for another. Note that this is a specific type of substitution attack. If the JWT could be used in an application context in which it could be confused with other kinds of JWTs, then mitigations MUST be employed to prevent these substitution attacks.

For mitigations, see [Section 3.7](#), [Section 3.8](#), [Section 3.9](#), and [Section 3.10](#).

3. Best Practices

The best practices listed below should be applied by practitioners to mitigate the threats listed in the preceding section.

3.1. Perform Algorithm Verification

Libraries MUST enable the caller to specify a supported set of algorithms and MUST NOT use any other algorithms when performing cryptographic operations. The library MUST ensure that the "alg" or "enc" header specifies the same algorithm that is used for the cryptographic operation. Moreover, each key MUST be used with exactly one algorithm, and this MUST be checked when the cryptographic operation is performed.

3.2. Use Appropriate Algorithms

As [Section 5.2 of \[RFC7515\]](#) says, "it is an application decision which algorithms may be used in a given context. Even if a JWS can be successfully validated, unless the algorithm(s) used in the JWS are acceptable to the application, it SHOULD consider the JWS to be invalid."

Therefore, applications MUST only allow the use of cryptographically current algorithms that meet the security requirements of the application. This set will vary over time as new algorithms are introduced and existing algorithms are deprecated due to discovered cryptographic weaknesses. Applications must therefore be designed to enable cryptographic agility.

That said, if a JWT is cryptographically protected by a transport layer, such as TLS using cryptographically current algorithms, there may be no need to apply another layer of cryptographic protections to the JWT. In such cases, the use of the "none" algorithm can be perfectly acceptable. JWTs using "none" are often used in application contexts in which the content is optionally signed; then the URL-safe claims representation and processing can be the same in both the signed and unsigned cases.

3.3. Validate All Cryptographic Operations

All cryptographic operations used in the JWT MUST be validated and the entire JWT MUST be rejected if any of them fail to validate. This is true not only of JWTs with a single set of Header Parameters but also for Nested JWTs, in which both outer and inner operations MUST be validated using the keys and algorithms supplied by the application.

3.4. Validate Cryptographic Inputs

Some cryptographic operations, such as Elliptic Curve Diffie-Hellman key agreement ("ECDH-ES") take inputs that contain invalid values, such as points not on the specified elliptic curve or other

invalid points. Either the JWS/JWE library itself must validate these inputs before using them or it must use underlying cryptographic libraries that do so (or both!).

3.5. Ensure Cryptographic Keys have Sufficient Entropy

The Key Entropy and Random Values advice in [Section 10.1 of \[RFC7515\]](#) and the Password Considerations in [Section 8.8 of \[RFC7518\]](#) MUST be followed. In particular, human-memorizable passwords MUST NOT be directly used as the key to a keyed-MAC algorithm such as "HS256".

3.6. Use UTF-8

[\[RFC7515\]](#), [\[RFC7516\]](#), and [\[RFC7519\]](#) all specify that UTF-8 be used for encoding and decoding JSON used in Header Parameters and JWT Claims Sets. Implementations and applications MUST do this, and not use other Unicode encodings for these purposes.

3.7. Validate Issuer and Subject

When a JWT contains an "iss" (issuer) claim, the application MUST validate that the cryptographic keys used for the cryptographic operations in the JWT belong to the issuer. If they do not, the application MUST reject the JWT.

The means of determining the keys owned by an issuer is application-specific. As one example, OpenID Connect [\[OpenID.Core\]](#) issuer values

are "https" URLs that reference a JSON metadata document that contains a "jwks_uri" value that is an "https" URL from which the issuer's keys are retrieved as a JWK Set [\[RFC7517\]](#). This same mechanism is used by [\[I-D.ietf-oauth-discovery\]](#). Other applications may use different means of binding keys to issuers.

Similarly, when the JWT contains a "sub" (subject) claim, the application MUST validate that the subject value corresponds to a valid subject and/or issuer/subject pair at the application. This may include confirming that the issuer is trusted by the application.

If the issuer, subject, or the pair are invalid, the application MUST reject the JWT.

3.8. Use and Validate Audience

If the same issuer can issue JWTs that are intended for use by more than one relying party or application, the JWT MUST contain an "aud" (audience) claim that can be used to determine whether the JWT is being used by an intended party or was substituted by an attacker at an unintended party. Furthermore, the relying party or application

MUST validate the audience value and if the audience value is not associated with the recipient, it MUST reject the JWT.

3.9. Use Explicit Typing

Confusion of one kind of JWT for another can be prevented by having all the kinds of JWTs that could otherwise potentially be confused include an explicit JWT type value and include checking the type value in their validation rules. Explicit JWT typing is accomplished

by using the "typ" header parameter. For instance, the [\[I-D.ietf-secevent-token\]](#) specification uses the "application/secevent+jwt" media type to perform explicit typing of Security Event Tokens (SETs).

Per the definition of "typ" in [Section 4.1.9 of \[RFC7515\]](#), it is RECOMMENDED that the "application/" prefix be omitted from the "typ" value. Therefore, for example, the "typ" value used to explicitly include a type for a SET SHOULD be "secevent+jwt". When explicit typing is employed for a JWT, it is RECOMMENDED that a media type name of the format "application/example+jwt" be used, where "example"

is replaced by the identifier for the specific kind of JWT.

Note that the use of explicit typing may not achieve disambiguation from existing kinds of JWTs, as the validation rules for existing kinds of JWTs often do not use the "typ" header parameter value. Explicit typing is RECOMMENDED for new uses of JWTs.

3.10. Use Mutually Exclusive Validation Rules for Different Kinds of JWTs

Each application of JWTs defines a profile specifying the required and optional JWT claims and the validation rules associated with them. If more than one kind of JWT can be issued by the same issuer,

the validation rules for those JWTs MUST be written such that they are mutually exclusive, rejecting JWTs of the wrong kind. To prevent

substitution of JWTs from one context into another, a number of strategies may be employed:

- Use explicit typing for different kinds of JWTs. Then the distinct "typ" values can be used to differentiate between the different kinds of JWTs.
- Use different sets of required claims or different required claim values. Then the validation rules for one kind of JWT will

reject those with different claims or values.

- Use different sets of required header parameters or different required header parameter values. Then the validation rules for

one kind of JWT will reject those with different header parameters or values.

- Use different keys for different kinds of JWTs. Then the keys used to validate one kind of JWT will fail to validate other kinds of JWTs.
- Use different "aud" values for different uses of JWTs from the same issuer. Then audience validation will reject JWTs substituted into inappropriate contexts.
- Use different issuers for different kinds of JWTs. Then the distinct "iss" values can be used to segregate the different kinds of JWTs.

Given the broad diversity of JWT usage and applications, the best combination of types, required claims, values, header parameters, key usages, and issuers to differentiate among different kinds of JWTs will, in general, be application specific.

4. IANA Considerations

This document requires no IANA actions.

5. Acknowledgements

Thanks to Antonio Sanso for bringing the "ECDH-ES" invalid point attack to the attention of JWE and JWT implementers. Thanks to Nat Sakimura for advocating the use of explicit typing.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", [BCP 14](#), [RFC 2119](#), DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", [RFC 7159](#), DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.

[RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", [RFC 7515](#), DOI 10.17487/RFC7515, May 2015, <<http://www.rfc-editor.org/info/rfc7515>>.

- [RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", [RFC 7516](#), DOI 10.17487/RFC7516, May 2015, <<http://www.rfc-editor.org/info/rfc7516>>.
- [RFC7518] Jones, M., "JSON Web Algorithms (JWA)", [RFC 7518](#), DOI 10.17487/RFC7518, May 2015, <<http://www.rfc-editor.org/info/rfc7518>>.
- [RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", [RFC 7519](#), DOI 10.17487/RFC7519, May 2015, <<http://www.rfc-editor.org/info/rfc7519>>.

6.2. Informative References

- [I-D.ietf-oauth-discovery]
Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", [draft-ietf-oauth-discovery-06](#) (work in progress), March 2017.
- [I-D.ietf-secevent-token]
"Security
Hunt, P., Denniss, W., Ansari, M., and M. Jones,
Event Token (SET)", [draft-ietf-secevent-token-02](#) (work in progress), June 2017.
- [Langkemper]
Langkemper, S., "Attacking JWT Authentication", September 2016, <<https://www.sjoerdlangkemper.nl/2016/09/28/attacking-jwt-authentication/>>.
- [OpenID.Core]
C.
Sakimura, N., Bradley, J., Jones, M., Medeiros, B., and
Mortimore, "OpenID Connect Core 1.0", November 2014,
<http://openid.net/specs/openid-connect-core-1_0.html>.
- [RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", [RFC 6749](#), DOI 10.17487/RFC6749, October 2012, <<http://www.rfc-editor.org/info/rfc6749>>.
- [RFC7517] Jones, M., "JSON Web Key (JWK)", [RFC 7517](#), DOI 10.17487/RFC7517, May 2015, <<http://www.rfc-editor.org/info/rfc7517>>.
- [Sanso] Sanso, A., "Critical Vulnerability Uncovered in JSON Encryption", March 2017, <<https://blogs.adobe.com/security/2017/03/critical-vulnerability-uncovered-in-json-encryption.html>>.

Appendix A. Document History

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

A.1. [draft-ietf-oauth-jwt-bcp-00](#)

- Initial WG draft. No change from the latest individual version.

A.2. [draft-sheffer-oauth-jwt-bcp-01](#)

- Added explicit typing.

A.3. [draft-sheffer-oauth-jwt-bcp-00](#)

- Initial version.

Authors' Addresses

Yaron Sheffer
Intuit

E-Mail: aronf.ietf@gmail.com

Dick Hardt
Amazon

E-Mail: dick@amazon.com

Michael B. Jones
Microsoft

E-Mail: mbj@microsoft.com

URI: <http://self-issued.info/>

