OAuth Working Group
Internet-Draft
Updates: RFC 7519 (if approved)
Intended status: Best Current Practice
Expires: April 15, 2020

Y. Sheffer
Intuit
D. Hardt

M. Jones
Microsoft

October 13, 2019

JSON Web Token Best Current Practices
draft-ietf-oauth-jwt-bcp-07

Abstract

JSON Web Tokens, also known as JWTs, 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 https://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 April 15, 2020.

Copyright Notice

Copyright (c) 2019 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 (https://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction 3
  1.1. Target Audience 4
  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 5
  2.3. Incorrect Composition of Encryption and Signature 5
  2.4. Plaintext Leakage through Analysis of Ciphertext Length . 5
  2.5. Insecure Use of Elliptic Curve Encryption 5
  2.6. Multiplicity of JSON Encodings 6
  2.7. Substitution Attacks 6
  2.8. Cross-JWT Confusion 6
  2.9. Indirect Attacks on the Server 6
3. Best Practices 7
  3.1. Perform Algorithm Verification 7
  3.2. Use Appropriate Algorithms 7
  3.3. Validate All Cryptographic Operations 8
  3.4. Validate Cryptographic Inputs 8
  3.5. Ensure Cryptographic Keys have Sufficient Entropy 8
  3.6. Avoid Length-Dependent Encryption Inputs 9
  3.7. Use UTF-8 9
  3.8. Validate Issuer and Subject 9
  3.9. Use and Validate Audience 9
  3.10. Do Not Trust Received Claims 10
  3.11. Use Explicit Typing 10
  3.12. Use Mutually Exclusive Validation Rules for Different Kinds of JWTs 11
4. Security Considerations 11
5. IANA Considerations 12
6. Acknowledgements 12
7. References 12
  7.1. Normative References 12
  7.2. Informative References 13
Appendix A. Document History 15
  A.1. draft-ietf-oauth-jwt-bcp-07 15
  A.2. draft-ietf-oauth-jwt-bcp-06 15
  A.3. draft-ietf-oauth-jwt-bcp-05 15
  A.4. draft-ietf-oauth-jwt-bcp-04 15
  A.5. draft-ietf-oauth-jwt-bcp-03 15
  A.6. draft-ietf-oauth-jwt-bcp-02 15

A.7. draft-ietf-oauth-jwt-bcp-01 15
A.8. draft-ietf-oauth-jwt-bcp-00 15
A.9. draft-sheffer-oauth-jwt-bcp-01 15
A.10. draft-sheffer-oauth-jwt-bcp-00 15
Authors' Addresses 16

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 are 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 are 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 intended audience of this document is:

- Implementers of JWT libraries (and the JWS and JWE libraries used by those libraries),

- 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 BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

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 (see [McLean] and CVE-2015-9235).

For mitigations, see Section 3.1 and Section 3.2.

2.2. Weak Symmetric Keys

In addition, some applications use a keyed MAC algorithm such as "HS256" to sign tokens, but supply a weak symmetric key with insufficient entropy (such as a human memorable password). Such keys are vulnerable to offline brute-force or dictionary attacks once an attacker gets hold of such a token [Langkemper].

For mitigations, see Section 3.5.

2.3. 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.

2.4. Plaintext Leakage through Analysis of Ciphertext Length

Many encryption algorithms leak information about the length of the plaintext, with a varying amount of leakage depending on the algorithm and mode of operation. This problem is exacerbated when the plaintext is initially compressed, because the length of the compressed plaintext and, thus, the ciphertext depend not only on the length of the original plaintext but also on its content. Compression attacks are particularly powerful when there is attacker-controlled data in the same compression space as secret data, as is the case for some attacks on HTTPS.

See [Kelsey] for general background on compression and encryption, and [Alawatugoda] for a specific example of attacks on HTTP cookies.

For mitigations, see Section 3.6.

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. Multiplicity of JSON Encodings

Previous versions of the JSON format such as the obsoleted [RFC7159] allowed several different character encodings: UTF-8, UTF-16 and UTF-32. This is not the case anymore, with the latest standard [RFC8259] only allowing UTF-8 except for internal use within a "closed ecosystem". This ambiguity where older implementations and those used within closed environments may generate non-standard encodings, may result in the JWT being misinterpreted by its recipient. This in turn could be used by a malicious sender to bypass the recipient's validation checks.

For mitigations, see Section 3.7.

2.7. Substitution Attacks

There are attacks in which one recipient will be given a JWT that was intended for it, and will attempt to use it at a different recipient for which that JWT was not intended. For instance, if an OAuth 2.0 [RFC6749] access token is legitimately presented to an OAuth 2.0 protected resource for which it is intended, that protected resource might then present that same access token to a different protected resource for which the access token is not intended, in an attempt to gain access. If such situations are not caught, this can result in the attacker gaining access to resources that it is not entitled to access.

For mitigations, see Section 3.8 and Section 3.9.

2.8. 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 3.8, Section 3.9, Section 3.11, and Section 3.12.

2.9. Indirect Attacks on the Server

Various JWT claims are used by the recipient to perform lookup operations, such as database and LDAP searches. Others include URLs that are similarly looked up by the server. Any of these claims can

be used by an attacker as vectors for injection attacks or server-side request forgery (SSRF) attacks.

For mitigations, see 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 end-to-end 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. The "none" algorithm should only be used when the JWT is cryptographically protected by other means. 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. JWT libraries SHOULD NOT generate JWTs using "none" unless explicitly requested to do by the caller. Similarly, JWT libraries SHOULD NOT consume JWTs using "none" unless explicitly requested by the caller.

Applications SHOULD follow these algorithm-specific recommendations:

- Avoid all RSA-PKCS1 v1.5 encryption algorithms ([RFC8017], Sec. 7.2}, preferring RSA-OAEP ([RFC8017], Sec. 7.1).

- ECDSA signatures [ANSI-X962-2005] require a unique random value for every message that is signed. If even just a few bits of the random value are predictable across multiple messages then the security of the signature scheme may be compromised. In the worst case, the private key may be recoverable by an attacker. To counter these attacks, JWT libraries SHOULD implement ECDSA using the deterministic approach defined in [RFC6979]. This approach is completely compatible with existing ECDSA verifiers and so can be implemented without new algorithm identifiers being required.

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 may contain invalid values, such as points not on the specified elliptic curve or other invalid points (see, e.g. [Valenta], Sec. 7.1). The JWS/JWE library itself must validate these inputs before using them or it must use underlying cryptographic libraries that do so (or both!).

ECDH-ES ephemeral public key (epk) inputs should be validated according to the recipient's chosen elliptic curve. For the NIST prime-order curves P-256, P-384 and P-521, validation MUST be performed according to Section 5.6.2.3.4 "ECC Partial Public-Key Validation Routine" of NIST Special Publication 800-56A revision 3 [nist-sp-800-56a-r3]. Likewise, if the "X25519" or "X448" [RFC8037] algorithms are used, then the security considerations in [RFC8037] apply.

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".

In particular, passwords should only be used to perform key encryption, rather than content encryption, as described in Section 4.8 of [RFC7518]. Note that even when used for key encryption, password-based encryption is still subject to brute-force attacks.

3.6. Avoid Length-Dependent Encryption Inputs

Compression of data SHOULD NOT be done before encryption, because such compressed data often reveals information about the plaintext.

3.7. 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. This is also in line with the latest JSON specification [RFC8259]. Implementations and applications MUST do this, and not use or admit the use of other Unicode encodings for these purposes.

3.8. 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 [RFC8414]. 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.

3.9. 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.

In such cases, the relying party or application MUST validate the audience value and if the audience value is not present or not associated with the recipient, it MUST reject the JWT.

3.10. Do Not Trust Received Claims

The "kid" (key ID) header is used by the relying application to perform key lookup. Applications should ensure that this does not create SQL or LDAP injection vulnerabilities, by validating and/or sanitizing the received value.

Similarly, blindly following a "jku" (JWK set URL) or "x5u" (X.509 URL) header, which may contain an arbitrary URL, could result in server-side request forgery (SSRF) attacks. Applications SHOULD protect against such attacks, e.g., by matching the URL to a whitelist of allowed locations, and ensuring no cookies are sent in the GET request.

3.11. Use Explicit Typing

Sometimes, one kind of JWT can be confused for another. If a particular kind of JWT is subject to such confusion, that JWT can include an explicit JWT type value, and the validation rules can specify checking the type. This mechanism can prevent such confusion. Explicit JWT typing is accomplished by using the "typ" header parameter. For instance, the [RFC8417] 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.

When applying explicit typing to a Nested JWT, the "typ" header parameter containing the explicit type value MUST be present in the inner JWT of the Nested JWT (the JWT whose payload is the JWT Claims Set). In some cases the same "typ" header parameter value will be present in the outer JWT as well, to explicitly type the entire Nested JWT.

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

3.12. 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, application developers may employ a number of strategies:

- 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. As discussed in Section 3.11, for new JWT applications, the use of explicit typing is RECOMMENDED.

4. Security Considerations

This entire document is about security considerations when implementing and deploying JSON Web Tokens.

5. IANA Considerations

This document requires no IANA actions.

6. Acknowledgements

Thanks to Antonio Sanso for bringing the "ECDH-ES" invalid point attack to the attention of JWE and JWT implementers. Tim McLean [McLean] published the RSA/HMAC confusion attack. Thanks to Nat Sakimura for advocating the use of explicit typing. Thanks to Neil Madden for his numerous comments, and to Carsten Bormann, Brian Campbell, Brian Carpenter, Alissa Cooper, Roman Danyliw, Ben Kaduk, Mirja Kuehlewind, Barry Leiba, Eric Rescorla, Adam Roach, Martin Vigoureux, and Eric Vyncke for their reviews.

7. References

7.1. Normative References

[nist-sp-800-56a-r3]
Barker, E., Chen, L., Keller, S., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography, Draft NIST Special Publication 800-56A Revision 3", April 2018, <https://doi.org/10.6028/NIST.SP.800-56Ar3>.

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

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

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

[RFC7516] Jones, M. and J. Hildebrand, "JSON Web Encryption (JWE)", RFC 7516, DOI 10.17487/RFC7516, May 2015, <https://www.rfc-editor.org/info/rfc7516>.

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://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, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[RFC8037] Liusvaara, I., "CFRG Elliptic Curve Diffie-Hellman (ECDH) and Signatures in JSON Object Signing and Encryption (JOSE)", RFC 8037, DOI 10.17487/RFC8037, January 2017, <https://www.rfc-editor.org/info/rfc8037>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

7.2. Informative References

[Alawatugoda] Alawatugoda, J., Stebila, D., and C. Boyd, "Protecting Encrypted Cookies from Compression Side-Channel Attacks", Financial Cryptography and Data Security pp. 86-106, DOI 10.1007/978-3-662-47854-7_6, 2015.

[ANSI-X962-2005] "American National Standard X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA)", November 2005.

[Kelsey] Kelsey, J., "Compression and Information Leakage of Plaintext", Fast Software Encryption pp. 263-276, DOI 10.1007/3-540-45661-9_21, 2002.

[Langkemper] Langkemper, S., "Attacking JWT Authentication", September 2016, <https://www.sjoerdlangkemper.nl/2016/09/28/attacking-jwt-authentication/>.

[McLean] McLean, T., "Critical vulnerabilities in JSON Web Token libraries", March 2015, <https://auth0.com/blog/critical-vulnerabilities-in-json-web-token-libraries//>.

[OpenID.Core]
Sakimura, N., Bradley, J., Jones, M., Medeiros, B., and C. 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, <https://www.rfc-editor.org/info/rfc6749>.

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

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC8414] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata", RFC 8414, DOI 10.17487/RFC8414, June 2018, <https://www.rfc-editor.org/info/rfc8414>.

[RFC8417] Hunt, P., Ed., Jones, M., Denniss, W., and M. Ansari, "Security Event Token (SET)", RFC 8417, DOI 10.17487/RFC8417, July 2018, <https://www.rfc-editor.org/info/rfc8417>.

[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>.

[Valenta] Valenta, L., Sullivan, N., Sanso, A., and N. Heninger, "In search of CurveSwap: Measuring elliptic curve implementations in the wild", March 2018, <https://ia.cr/2018/298>.

Appendix A. Document History

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

A.1. draft-ietf-oauth-jwt-bcp-07

- IESG review comments.

A.2. draft-ietf-oauth-jwt-bcp-06

- Second AD review.
- Removed unworkable recommendation to pad encrypted passwords.

A.3. draft-ietf-oauth-jwt-bcp-05

- Genart review comments.

A.4. draft-ietf-oauth-jwt-bcp-04

- AD review comments.

A.5. draft-ietf-oauth-jwt-bcp-03

- Acknowledgements.

A.6. draft-ietf-oauth-jwt-bcp-02

- Implemented WGLC feedback.

A.7. draft-ietf-oauth-jwt-bcp-01

- Feedback from Brian Campbell.

A.8. draft-ietf-oauth-jwt-bcp-00

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

A.9. draft-sheffer-oauth-jwt-bcp-01

- Added explicit typing.

A.10. draft-sheffer-oauth-jwt-bcp-00

- Initial version.

Authors' Addresses

Yaron Sheffer
Intuit

EMail: yaronf.ietf@gmail.com

Dick Hardt

EMail: dick.hardt@gmail.com

Michael B. Jones
Microsoft

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