Web Authorization Protocol
Internet-Draft
Obsoletes: 8725 (if approved)
Updates: 7519 (if approved)
Intended status: Best Current Practice
Expires: 23 April 2026

Y. Sheffer
Intuit
D. Hardt
M. Jones
Self-Issued Consulting

20 October 2025

JSON Web Token Best Current Practices
draft-ietf-oauth-rfc8725bis-01

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. This Best Current Practices (BCP) specification updates RFC 7519 to provide actionable guidance leading to secure implementation and deployment of JWTs.

This BCP specification furthermore replaces the existing JWT BCP specification RFC 8725 to provide additional actionable guidance covering threats and attacks that have been discovered since RFC 8725 was published.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-oauth-rfc8725bis/.

Discussion of this document takes place on the Web Authorization Protocol Working Group mailing list (mailto:oauth@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/oauth/. Subscribe at https://www.ietf.org/mailman/listinfo/oauth/.

Source for this draft and an issue tracker can be found at https://github.com/https://github.com/oauth-wg/draft-ietf-oauth-rfc8725bis.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Copyright Notice

Table of Contents

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 audiences of this document are:

* 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, has 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 Message Authentication Code (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][JWT-Cracker].

For mitigations, see Section 3.5.

2.3. Incorrect Use and Composition of Encryption and Signature

Most authentication use cases only require a simple signed JWT as their token. However verifiers don't always check that the received JWT is a JWS (a signed JWT) as opposed to a JWE (a JWT with encrypted structure). This can result in vulnerabilities when the verifier's library does not distinguish between successful decryption and successful signature validation [CVE-2023-51774].

In the more complicated use cases where confidentiality is required, 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. JWEs are vulnerable to this leakage. This problem is exacerbated when the plaintext is initially compressed, because the length of the compressed plaintext and, thus, the ciphertext depends 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, which 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 Javascript Object Signing and Encryption (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 Sections 3.8 and 3.9.

2.8. Cross-JWT Confusion

As JWTs are used by more protocols in diverse ways, it becomes increasingly important to prevent JWT tokens that have been issued for one purpose being 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 Sections 3.8, 3.9, 3.11, and 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 Lightweight Directory Access Protocol (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.

2.10. Computation Cost of Unreasonable Number of Hash Iterations

The p2c (PBES2 Count) header parameter for the PBES2 encryption algorithms specifies the number of iterative hash computations to be performed. Attackers can use a very large count, thereby imposing an unreasonable computational burden on recipients.

For mitigations, see Section 3.13.

2.11. Algorithm Verification Code Not Defensively Written

Some JWT implementations included a list of disallowed algorithm names, e.g., do not use "none". These same applications misinterpreted the JOSE specifications when parsing the token, reading algorithm values as if they were case-insensitive. The end result was that an attacker could change the "alg" value to "noNE" and bypass the security check.

For mitigations, see Section 3.1.

2.12. JWE Decompression Bomb Attack

JWE supports the optional compression of the plaintext prior to encryption via the "zip" header parameter as defined in [RFC7516] Section 4.1.3. Upon decryption, recipients are expected to decompress the payload before further processing. However, if the recipient does not enforce limits on the size of the decompressed output, an attacker can craft a malicious JWE with a highly compressed, arbitrarily large payload. This can cause excessive resource consumption (CPU, memory), resulting in Denial of Service (DoS).

For mitigation, see Section 3.15.

2.13. JWT Format Confusion

Some JWS implementations support both the Compact and JSON Serializations. While JWTs MUST use the Compact Serialization, if an application by mistake verifies a JWT using the JSON Serialization but extracts claims by parsing it as a JWT using the Compact Serialization (e.g., via string splitting), an attacker can craft a valid JSON JWS with a forged payload. This mismatch in format handling can lead to authentication bypass or impersonation.

For mitigations, see Section 3.14.

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.

Libraries SHOULD opt for defensive security policies to cope with potential issues in the underlying infrastructure, such as the JSON parser. In particular, libraries SHOULD use allowlists for critical parameters such as "alg" instead of blocklists.

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. The URL-safe claims representation and processing in this context can be the same in both the signed and unsigned cases. JWT libraries SHOULD NOT generate JWTs using "none" unless explicitly requested to do so 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], Section 7.2), preferring RSAES-OAEP ([RFC8017], Section 7.1).

* Elliptic Curve Digital Signature Algorithm (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.

Libraries MUST allow the verifier to distinguish between signed JWTs (JWSes) and encrypted JWTs (JWEs). This allows verifiers to easily establish a policy of only accepting signed JWTs.

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. This includes points not on the specified elliptic curve or other invalid points (e.g., [Valenta], Section 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!).

Elliptic Curve Diffie-Hellman Ephemeral Static (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 "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography" [nist-sp-800-56a-r3]. 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". Moreover, 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 Compression of Encryption Inputs

Compression of data SHOULD NOT be used when creating a JWE, 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 allow 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, or may do so in the future, 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.

In such cases, the relying party or application MUST validate the audience value, and if no audience value is present or none of the values are 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" Header Parameter value, compared to the associated media type. 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. Therefore, for example, the media type name for a SET SHOULD be "application/secevent+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 of 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.

3.13. Limit Hash Iteration Count

Implementations are RECOMMENDED to set a reasonable upper limit on the number of hash iterations that can be performed when validating encrypted content using PBES2 encryption algorithms, so as to prevent attackers from imposing an unreasonable computational burden on recipients. [OWASP-Password-Storage] states a specific iteration count (600,000 at time of publishing) is required when using HMAC-SHA-256 to achieve FIPS-140 compliance. Rejecting inputs with a p2c (PBES2 Count) value larger than double the recommended OWASP value is RECOMMENDED.

3.14. Check JWT Format Type

Implementations MUST confirm the JWT is in a legal format while parsing it. Legal JWTs, being dot-concatenated base64url strings, contain only the ASCII characters for letters, numbers, dash, underscore, and period. Content with any other characters - especially braces and quotation marks - is not a JWT and MUST be rejected.

3.15. Limit JWE Decompression Size

Implementations are RECOMMENDED to set a reasonable upper limit on the decompressed size of a JWE such as 250 KB.

4. Security Considerations

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

5. IANA Considerations

This document has 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 published the RSA/HMAC confusion attack [McLean]. 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 Kテシhlewind, Barry Leiba, Dan Moore, Eric Rescorla, Adam Roach, Martin Vigoureux, and テ詠ic Vyncke for their reviews.

7. References

7.1. Normative References

[nist-sp-800-56a-r3]
Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for pair-wise key-establishment schemes using discrete logarithm cryptography", National Institute of Standards and Technology, DOI 10.6028/nist.sp.800-56ar3, 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/rfc/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/rfc/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/rfc/rfc7515>.

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

[RFC7518] Jones, M., "JSON Web Algorithms (JWA)", RFC 7518, DOI 10.17487/RFC7518, May 2015, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/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/rfc/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/rfc/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/rfc/rfc8259>.

7.2. Informative References

[Alawatugoda]
Alawatugoda, J., Stebila, D., and C. Boyd, "Protecting Encrypted Cookies from Compression Side-Channel Attacks", Springer Berlin Heidelberg, Lecture Notes in Computer Science pp. 86-106, DOI 10.1007/978-3-662-47854-7_6, ISBN ["9783662478530", "9783662478547"], 2015, <https://doi.org/10.1007/978-3-662-47854-7_6>.

[ANSI-X962-2005]
American National Standards Institute, "Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)", November 2005.

[CVE-2015-9235]
NIST, "CVE-2015-9235 Detail", National Vulnerability Database, May 2018, <https://nvd.nist.gov/vuln/detail/CVE-2015-9235>.

[CVE-2023-51774]
NIST, "CVE-2023-51774 Detail", National Vulnerability Database, February 2024, <https://nvd.nist.gov/vuln/detail/CVE-2023-51774>.

[JWT-Cracker]
Rius, B., "JWT Cracker", n.d., <https://github.com/brendan-rius/c-jwt-cracker>.

[Kelsey]
Kelsey, J., "Compression and Information Leakage of Plaintext", Springer Berlin Heidelberg, Lecture Notes in Computer Science pp. 263-276, DOI 10.1007/3-540-45661-9_21, ISBN ["9783540440093", "9783540456612"], 2002, <https://doi.org/10.1007/3-540-45661-9_21>.

[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., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-core-1_0.html>.

[OWASP-Password-Storage]
OWASP, "Password Storage Cheat Sheet", OWASP Cheat Sheet Series , 2025, <https://cheatsheetseries.owasp.org/cheatsheets/Password_Storage_Cheat_Sheet.html>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/rfc/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/rfc/rfc7159>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/rfc/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/rfc/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/rfc/rfc8417>.

[Sanso] Sanso, A., "Critical Vulnerability in JSON Web Encryption", March 2017, <https://auth0.com/blog/critical-vulnerability-in-json-web-encryption/>.

[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. Changes from RFC 8725

This document obsoletes RFC 8725 and provides several significant improvements and additions:

1. Algorithm Verification: Added defensive checking to address incorrect reading of alg values as being case-insensitive (Section 3.1).

2. Encryption-Signature Confusion: Added mitigation for attacks where verifiers don't distinguish between successful decryption and successful signature validation (Section 3.12).

3. PBES2 Count Limits: Added requirements to reject unreasonably large p2c (PBES2 Count) values to prevent DoS attacks (Section 3.13).

4. JWT Format Confusion: Added mitigation for JWT serialization format confusion attacks (Section 3.14).

5. Compression DoS: Added mitigation for DoS attacks resulting from abuse of compression in JWE (Section 3.15).

Appendix B. Document History

[[Note to RFC Editor: please remove before publication.]]

B.1. draft-ietf-oauth-rfc8725bis-01

* Applied editorial suggestions by Dan Moore.

* Described changes relative to RFC 8725.

B.2. draft-ietf-oauth-rfc8725bis-00

* Draft adopted, no textual changes

B.3. draft-sheffer-oauth-rfc8725bis-02

* Obsoletes RFC 8725 and updates RFC 7519.

B.4. draft-sheffer-oauth-rfc8725bis-01

* Mitigate encryption-signature confusion.

* Reject unreasonably large p2c (PBES2 Count) values.

* Defensive checking to address incorrect reading of alg values as being case-insensitive.

* Mitigate DoS attacks resulting from abuse of compression.

* Mitigate JWT serialization format confusion.

B.5. draft-sheffer-oauth-rfc8725bis-00

* Initial version, text is identical to RFC 8725.

Authors' Addresses

Yaron Sheffer
Intuit
Email: yaronf.ietf@gmail.com

Dick Hardt
Email: dick.hardt@gmail.com

Michael B. Jones
Self-Issued Consulting
Email: michael_b_jones@hotmail.com
URI: https://self-issued.info/