Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: 15 November 2026

S. Knauer

14 May 2026

Secure Webhook Token (SWT)
draft-knauer-secure-webhook-token-02

Abstract

The Secure Webhook Token (SWT) is a specialized JSON Web Token (JWT) format designed for securely authorizing and verifying webhook requests. It defines a set of claims to ensure that the token is explicitly tied to webhook events and that its integrity can be reliably validated by the recipient. A key feature of SWT is the introduction of a unique claim, webhook, which encapsulates webhook-specific information, such as the event type.

By providing a structured and secure approach to webhook authentication, SWT aims to be a lightweight and flexible solution while maintaining compatibility with typical JWT structures. It is designed with the best practices in mind and may serve as a foundation for future standardization efforts.

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 15 November 2026.

Copyright Notice

Table of Contents

1. Introduction

The increasing use of webhooks requires a secure, standardized approach to authorizing webhook requests and verifying the sender’s authenticity. The SWT specifies a JWT-based [RFC7519] structure, which includes unique claims tailored for webhook events. This specification mandates that all tokens must be transmitted using HTTP POST requests (see Section 9.3.3 of [RFC9110]), with the token in the Authorization header and event data in the request body, ensuring secure transmission and optimal compatibility with typical HTTP implementations.

1.1. Notational Conventions

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 "Key words for use in RFCs to Indicate Requirement Levels" [RFC2119]. The interpretation should only be applied when the terms appear in all capital letters.

2. Secure Webhook Token (SWT) Overview

This specification defines required claims within the JWT [RFC7519] payload and prescribes the transport protocol as HTTP POST. The token is transmitted in the Authorization header while the webhook event data is sent in the request body. These constraints ensure interoperability across diverse systems and provide a consistent approach to webhook authorization.

3. Transport Requirements

SWTs MUST be transmitted using HTTP POST requests only.

The token MUST be included in the Authorization header using the Bearer scheme, while the webhook event data MUST be sent in the request body.

4. Token Size Limitation

Recommended Maximum Token Size: While there is no strict maximum size for an SWT, the token SHOULD be kept concise, containing only essential claims to support efficient processing and minimize resource usage. The primary use case for SWTs is to securely authorize and trigger events rather than transmit extensive data.

Header Size Considerations: Since the token is transmitted in the Authorization header, implementers should be aware of HTTP header size limits imposed by web servers and proxies. Most web servers have a default maximum HTTP header size of 8KB, though this can often be configured. To ensure broad compatibility, the JWT token itself SHOULD remain reasonably small (typically under 4KB).

5. JWT Structure

An SWT comprises the following:

5.1. Header

* *alg*: Specifies the signing algorithm [RFC7518], typically "HS256" (HMAC SHA-256).
* *typ*: Specifies the token type, which MUST be "SWT". This distinguishes Secure Webhook Tokens from generic JWTs and enables recipients to recognize and process them according to this specification.

5.1.1. Example Header Structure

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

5.2. Payload

The payload contains the following required claims, carefully chosen to meet security and identification needs:

* *webhook*: Custom claim specific to this specification.
  - *event*: Describes the webhook event type (e.g., "payment.received"). Values should be concise to meet the size limitations.
  - *hash* (OPTIONAL): The cryptographic hash of the request body (see Request Body Integrity (#request-body-integrity)). This field MUST be present when the request body is non-empty and MUST be omitted when the request body is empty.
  - *retry_count* (OPTIONAL): A non-negative integer indicating the delivery attempt number (0 = initial attempt, 1+ = retry). See JWT webhook claim (#webhook-claim-specification) for detailed specification.

* *iss* (Issuer): Identifies the token issuer (e.g., a unique name or domain representing the entity). Short, meaningful identifiers are RECOMMENDED.

* *sub* (Subject) (OPTIONAL): While traditionally used to identify the principal subject of a JWT, the sub claim is not required in SWT. The purpose and intent of the token are unambiguously defined by the webhook claim. However, implementers may include sub to identify the system, service, or entity responsible for issuing the token, if such context is useful.

* *exp* (Expiration): Specifies the token’s expiration time as a Unix timestamp. This prevents tokens from being valid indefinitely.

* *nbf* (Not Before): Specifies the earliest time the token is considered valid, using a Unix timestamp.

* *iat* (Issued At): The timestamp when the token was issued, aiding in token age validation.

* *jti* (JWT ID): A unique identifier for each token to prevent replay attacks. A UUID [RFC9562] or similarly unique identifier is RECOMMENDED.

5.2.1. Example Payload Structure

5.2.1.1. SWT payload with empty body (notification only):

```
{
  "webhook": {
    "event": "ping"
  },
  "exp": 1733987961,
  "nbf": 1733987661,
  "iat": 1733987661,
  "iss": "swt.example.com",
  "jti": "2020B14D-C365-4BCF-84CD-5D423E0C6687"
}
```

5.2.1.2. SWT payload with body content:

```
{
  "webhook": {
    "event": "delivery.scheduled",
    "hash": "sha-256:22b793b7c031e6a3...05ba920b",
    "retry_count": 0
  },
  "exp": 1733987961,
  "nbf": 1733987661,
  "iat": 1733987661,
  "iss": "swt.example.com",
  "jti": "2020B14D-C365-4BCF-84CD-5D423E0C6687"
}
```

5.2.1.3. SWT payload for retry attempt:

```
{
  "webhook": {
    "event": "delivery.scheduled",
    "hash": "sha-256:22b793b7c031e6a3...05ba920b",
    "retry_count": 3
  },
  "exp": 1733988561,
  "nbf": 1733988261,
  "iat": 1733988261,
  "iss": "swt.example.com",
  "jti": "A1C3F89E-2F4D-4A8C-9B2E-7D8F1C3E5A9B"
}
```

6. Request Body Integrity

6.1. Overview

To ensure payload integrity when sending body content, webhook requests with non-empty request bodies MUST include a cryptographic hash of the request body. Instead of using separate fields for algorithm and value, a self-describing hash field is used:

```
"webhook": {
  "event": "delivery.scheduled",
  "hash": "sha-512:f8f24b01df358f10...df03db5cf"
}
```

This format allows the server to interpret the algorithm dynamically and accept or reject hashes according to its policies.

6.2. Format

hash = "<algorithm>:<hash_value>"

* <algorithm>: The cryptographic hash algorithm, using standardized names such as sha-256, sha-512, sha3-256.
* <hash_value>: The lowercase hexadecimal representation of the hash of the request body. Hexadecimal encoding is REQUIRED for interoperability.

6.2.1. Examples

```
"hash": "sha-256:04ef96c3bc56b3c4...200a5353"
"hash": "sha-512:f8f24b01df358f10...df03db5cf"
```

6.3. Server-side Processing

1. Parse the hash string to separate algorithm and value.

```
[algo, value] = hash.split(":")
```

2. Validate the algorithm against the server's allowed list.
3. Compute the hash of the received payload using algo.
4. Compare computed hash with value.
    * If they match → payload integrity verified.
    * If they do not match → reject the webhook request.

6.3.1. Recommendations

* Algorithm Registry: Use standardized algorithm names aligned with the IANA Named Information Hash Algorithm Registry [RFC6920] to avoid ambiguities.
* Extensibility: New hash algorithms may be introduced without changing the JSON schema.
* Security Note: Do not accept weak or deprecated hash algorithms (e.g., MD5, SHA-1).

7. Security Considerations

1. *Signature Verification*: Receivers MUST verify the JWT signature before processing any claims. Failure to verify signatures exposes the system to token forgery attacks. The signature algorithm specified in the JWT header MUST be validated against an allowlist of permitted algorithms to prevent algorithm substitution attacks.

2. *Algorithm Allowlist*: Implementations MUST maintain an allowlist of acceptable signing algorithms and MUST reject tokens signed with algorithms not on this list. The alg header parameter MUST be checked against this allowlist before signature verification. Implementations MUST NOT accept the "none" algorithm.

3. *HTTPS Transport*: It is REQUIRED that SWT transmissions occur over HTTPS to ensure the token’s confidentiality and protect against man-in-the-middle attacks.

4. *Replay Protection*: The jti claim MUST be checked for uniqueness on the server side to prevent replay attacks. Each jti value MUST only be accepted once within the token’s validity period. Implementations SHOULD maintain a cache or registry of seen jti values.

5. *Expiration and Time-based Claims*: Systems MUST validate the exp, nbf, and iat claims to ensure tokens are used only within their intended validity period. Implementations SHOULD allow for a small clock skew tolerance (e.g., 60 seconds) when validating time-based claims to account for clock differences between systems. Token lifetime (exp - iat) SHOULD be kept as short as practical for the use case, typically not exceeding 15 minutes for most webhook scenarios.

6. *Key Management*: Symmetric keys used for HMAC algorithms MUST be generated using a cryptographically secure random number generator and MUST have sufficient entropy (at least 256 bits for HS256). Keys SHOULD be rotated regularly. For asymmetric algorithms, private keys MUST be protected with appropriate access controls and SHOULD be stored in hardware security modules (HSMs) or key management systems (KMS) when available.

7. *Hash Algorithm Security*: When validating request body hashes, implementations MUST reject weak or deprecated hash algorithms (e.g., MD5, SHA-1). Only cryptographically strong hash algorithms (SHA-256 or stronger) SHOULD be accepted.

8. *Respect Header Limits*: Excessively large JWT tokens SHOULD be avoided to ensure compatibility with HTTP header size limits imposed by web servers and proxies.

9. *Retry Count Validation*: When the retry_count field is present, receivers SHOULD validate that its value is reasonable to prevent abuse. Implementations MAY enforce maximum retry limits (e.g., rejecting requests with retry_count exceeding a configured threshold) to protect against resource exhaustion attacks. The retry_count value MUST be a non-negative integer. Receivers

SHOULD consider implementing exponential backoff expectations and MAY reject retry attempts that arrive too quickly relative to the retry_count value. Additionally, the combination of jti and retry_count should be considered together: each retry attempt MUST use a unique jti value, as retries represent distinct delivery attempts even when delivering the same event.

8. Interoperability Considerations

SWT's use of standard HTTP POST requests with JWT-based authorization ensures broad compatibility with existing web infrastructure. The structure is simple to parse and easy to integrate with existing JWT validation libraries. The separation of token metadata and event data provides flexibility for various payload sizes and content types.

9. Signing and Encryption Recommendations

To maintain a balance between security and usability, SWTs are RECOMMENDED to be used as JSON Web Signatures (JWS) [RFC7515], specifically signed JWTs. This provides data integrity and allows the recipient to verify the token's authenticity. The following signing algorithms are suggested for SWTs, as they offer a secure yet practical approach:

* HS256, HS384, and HS512: Symmetric algorithms using HMAC with SHA-256, SHA-384, and SHA-512, respectively. These algorithms are considered secure and performant for most applications, especially when using shared symmetric keys between the issuer and the receiver.

While these symmetric algorithms are RECOMMENDED for ease of use and efficiency, other supported algorithms, including asymmetric methods such as RS256 (RSA with SHA-256) or ES256 (ECDSA with SHA-256), MAY also be used if needed based on security requirements or system constraints.

9.1. Encrypted JWT Support (JWE)

In addition to JWS, SWT allows the use of the JSON Web Encryption (JWE) [RFC7516] standard if encrypted JWTs are required to protect sensitive data within the token. JWE adds encryption layers to JWTs, providing confidentiality as well as integrity protection, which may be appropriate in contexts where tokens contain sensitive information.

10. IANA Considerations

10.1. Media Type Registration

This specification registers the application/swt+jwt media type in the IANA "Media Types" registry in the manner described in [RFC6838].

The following entries should be registered:

* Type name: application
* Subtype name: swt+jwt
* Required parameters: n/a
* Optional parameters: n/a
* Encoding considerations: 8-bit; application/swt+jwt values are encoded as a series of base64url-encoded values (some of which may be the empty string) separated by period ('.') characters
* Security considerations: See Security Considerations section of this document
* Interoperability considerations: n/a
* Published specification: This document
* Applications that use this media type: Webhook delivery systems, event notification platforms, and applications requiring secure webhook authentication
* Fragment identifier considerations: n/a
* 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: Stephan Knauer, mail@knauer.consulting
* Intended usage: COMMON
* Restrictions on usage: none
* Author: Stephan Knauer, mail@knauer.consulting
* Change controller: IESG
* Provisional registration? No

10.2. JSON Web Token Claims Registration

This document updates the IANA "JSON Web Token Claims" registry for JWT Claim Names. Following the format in Section 10.1.1 of [RFC7519], the following should be registered.

* Claim Name: "webhook"
* Claim Description: Webhook
* Change Controller: IESG
* Specification Document(s): Section 10.3 of this document

10.3. JWT webhook claim

The webhook claim is a JSON object that describes the event to be triggered and optionally includes metadata about the event data sent in the request body.

It consists of the following fields:

* event: A case-sensitive string describing the name of the webhook event. It should be concise and specific (e.g., "order.created").
* hash (OPTIONAL): The hash algorithm and the hash digest of the request body payload (hex-encoded) (see Section 6).
  - This field MUST be present when the request body is non-empty (Content-Length > 0).
  - This field SHOULD be omitted when the request body is empty (Content-Length = 0).
* retry_count (OPTIONAL): A non-negative integer indicating the delivery attempt number for this webhook event.
  - A value of 0 indicates the initial delivery attempt (no retry).
  - A value of 1 or greater indicates a retry attempt (e.g., 1 = first retry, 2 = second retry).
  - This field SHOULD be included when the webhook system implements retry logic for failed deliveries.
  - Receivers MAY use this value to implement progressive backoff strategies, rate limiting, or to decide whether to accept further retries.

Implementers MAY restrict the set of supported algorithms based on their security posture.

| Senders and receivers SHOULD agree on supported algorithms in advance.
|
| Although NOT RECOMMENDED, it may be necessary to send non-JSON event data via SWT. The HTTP _Content-Type_ header SHOULD be set accordingly.

10.3.1. Examples of JWT webhook claim

Initial delivery attempt:

```
"webhook": {
  "event": "issue.opened",
  "hash": "sha-256:04ef96c3bc56b3c4...200a5353",
  "retry_count": 0
}
```

Retry attempt:

```
"webhook": {
  "event": "issue.opened",
  "hash": "sha-256:04ef96c3bc56b3c4...200a5353",
  "retry_count": 2
}
```

11. Validation

To validate an SWT, the receiving system MUST perform the following checks:

1. *JWT Validation*: The token in the Authorization header MUST be a valid JWT with all required claims.
2. *Webhook Claim Validation*: The JWT MUST contain a webhook claim with an event field.
3. *Body and Hash Validation*:
   * *If the request body is non-empty* (Content-Length > 0):
     - The webhook claim MUST contain a hash field.
     - The computed hash of the request body MUST match the hash value specified in the webhook claim using the specified algorithm.
   * *If the request body is empty* (Content-Length = 0):
     - The webhook claim MUST NOT contain a hash field.

If any validation check fails, the server SHOULD respond with an appropriate HTTP error code:

* *401 Unauthorized*: If JWT signature verification fails, the token is expired, or authentication credentials are invalid.
* *403 Forbidden*: If the token is valid but the issuer lacks permission to trigger the webhook or access the resource.
* *400 Bad Request*: If the request is malformed (e.g., missing required claims, invalid hash format, or hash mismatch).

11.1. Examples

11.1.1. Example 1: Empty body (notification only)

Webhook claim:

```
{
  "webhook": {
    "event": "ping"
  },
  "exp": 1733987961,
  "nbf": 1733987661,
  "iat": 1733987661,
  "iss": "swt.example.com",
  "jti": "2020B14D-C365-4BCF-84CD-5D423E0C6687"
}
```

POST request:

```
POST /webhook HTTP/1.1
Content-Length: 0
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
Host: example.com
```

11.1.2. Example 2: Non-empty body with hash

Webhook claim:

```
{
  "webhook": {
    "event": "issue.opened",
    "hash": "sha-256:04ef96c3bc56b3c4...200a5353",
    "retry_count": 0
  },
  "exp": 1733987961,
  "nbf": 1733987661,
  "iat": 1733987661,
  "iss": "swt.example.com",
  "jti": "2020B14D-C365-4BCF-84CD-5D423E0C6687"
}
```

POST request:

```
POST /webhook HTTP/1.1
Content-Length: 6123
Content-Type: application/json
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9...
Host: example.com

{
   "action": "opened",
   "issue": {
     "url": "https://api.github.com/repos/octocat/...",
     "number": 1347,
     ...
   },
   "repository" : {
     "id": 1296269,
     "full_name": "octocat/Hello-World",
     "owner": {
       "login": "octocat",
       "id": 1,
       ...
     },
     ...
   },
   "sender": {
     "login": "octocat",
     "id": 1,
     ...
   }
 }
```

12. Conclusion

The Secure Webhook Token (SWT) format provides a secure, interoperable, and efficient solution for webhook authentication. Its focus on signed payloads, minimal overhead, and clear transport guidance makes it suitable for modern web service ecosystems.

13. References

13.1. 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, <https://www.rfc-editor.org/info/rfc2119>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[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>.

[RFC9110] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "HTTP Semantics", STD 97, RFC 9110, DOI 10.17487/RFC9110, June 2022, <https://www.rfc-editor.org/info/rfc9110>.

13.2. Informative References

[RFC9562] Davis, K., Peabody, B., and P. Leach, "Universally Unique IDentifiers (UUIDs)", RFC 9562, DOI 10.17487/RFC9562, May 2024, <https://www.rfc-editor.org/info/rfc9562>.

Author’s Address

Stephan Knauer
Leipzig, SN
Germany
Email: mail@knauer.consulting