Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 1 September 2026
F. Skokan
Okta
28 February 2026

Additional Hash Algorithms for OAuth 2.0 PKCE and Proof-of-Possession

draft-skokan-oauth-additional-hashes-04

Abstract

This document defines SHA-512 as an additional hash algorithm for OAuth 2.0 Proof Key for Code Exchange (PKCE), mutual-TLS certificate-bound access tokens, and Demonstrating Proof of Possession (DPoP), for use in deployments operating under security policies that prohibit the use of SHA-256, which is otherwise mandated or the only option in these mechanisms.

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-skokan-oauth-additional-hashes/.

Source for this draft and an issue tracker can be found at https://github.com/panva/draft-oauth-additional-hashes.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Several OAuth 2.0 mechanisms exclusively mandate the use of SHA-256: Proof Key for Code Exchange (PKCE) [RFC7636], mutual-TLS certificate-bound access tokens [RFC8705], and Demonstrating Proof of Possession (DPoP) [RFC9449].

Security policies, such as the US Commercial National Security Algorithm (CNSA 2.0) Suite [cnsafaq], prohibit the use of SHA-256 and require SHA-384 or SHA-512. This prevents the deployment of these OAuth 2.0 mechanisms in such environments.

This document addresses this gap by defining SHA-512 alternatives for each of these mechanisms, for use in deployments operating under such constrained policies. For PKCE, a new S512 code challenge method is defined. For mutual-TLS certificate-bound access tokens, a new x5t#S512 confirmation method is defined. For DPoP, this document defines SHA-512 alternatives for the JWK Thumbprint confirmation method (jkt#S512) and the access token hash claim (ath#S512), as well as an extensible framework for authorization code binding and access token hash algorithm negotiation.

[[TODO: (#1 (https://github.com/panva/draft-oauth-additional-hashes/issues/1)) The hash algorithm chosen by this document is currently SHA-512. The working group should determine whether to define SHA-384 or SHA-512.]]

2. Conventions and Definitions

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.

All references to "CNSA 2.0" in this document refer to CNSA 2.0 [cnsafaq], unless stated otherwise.

3. Purpose and Scope

The sole purpose of this document is to enable deployments operating under security policies that prohibit SHA-256 to use PKCE, mutual-TLS certificate-bound access tokens, and DPoP. In such constrained deployments, the SHA-512 alternatives defined herein are used in place of their SHA-256 counterparts, since those deployments cannot use SHA-256 at all.

This document does not deprecate the SHA-256 based methods defined in existing specifications. The SHA-256 based methods remain the widely deployed, interoperable and recommended defaults for all mechanisms addressed by this document. Deployments that are not subject to such security policies SHOULD NOT offer or use the SHA-512 based methods defined herein.

The negotiation mechanisms defined herein may however facilitate a broader transition away from SHA-256 in the future, should that become necessary.

4. PKCE

Proof Key for Code Exchange (PKCE) [RFC7636] defines plain and S256 as code challenge methods, with S256 being the only method that applies a cryptographic hash to the code verifier. The specification establishes the "PKCE Code Challenge Methods" registry, which this document uses to register the S512 code challenge method.

4.1. S512 Code Challenge Method

This document defines a new code challenge method for use with PKCE [RFC7636]. The client creates a code challenge derived from the code verifier by using the following transformation on the code verifier:

```
S512:  code_challenge = BASE64URL(SHA-512(ASCII(code_verifier)))
```

The server-side verification of the code verifier follows Section 4.6 of [RFC7636], using SHA-512 as the hash algorithm.

4.2. Authorization Server Metadata

An Authorization Server that supports the S512 code challenge method MUST advertise its support in its Authorization Server metadata (e.g., [RFC8414] or [OpenID.Discovery]) by including S512 in the code_challenge_methods_supported metadata parameter value as defined in [RFC8414].

5. Mutual-TLS

OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens [RFC8705] exclusively uses SHA-256 for certificate-bound access tokens via the x5t#S256 confirmation method. No alternative hash algorithms or extension points for hash algorithm negotiation are defined. This document defines the x5t#S512 confirmation method and a Resource Server metadata parameter for negotiating the confirmation method.

5.1. x5t#S512 Confirmation Method

RFC 8705 [RFC8705] defines the x5t#S256 confirmation method member for binding access tokens to a client certificate using a SHA-256 hash of the DER-encoded X.509 certificate.

This document defines an analogous confirmation method member x5t#S512 that uses SHA-512 as the hash algorithm:

x5t#S512: The value is a base64url-encoded SHA-512 hash of the DER encoding of the X.509 certificate.

When using x5t#S512, the Authorization Server computes the SHA-512 hash of the client certificate presented during mutual-TLS and includes the result as the x5t#S512 member of the cnf claim in the access token (for JWT access tokens) or associates it with the token for later retrieval via token introspection [RFC7662].

The Resource Server MUST compute the SHA-512 hash of the client certificate presented during mutual-TLS and compare it with the x5t#S512 value in the cnf claim. If the values do not match, the Resource Server MUST reject the request.

The choice of x5t#S512 over x5t#S256 is a deployment decision. It can be configured out of band or by the Authorization Server using the Resource Server's metadata (Section 5.2).

[[TODO: (#2 (https://github.com/panva/draft-oauth-additional-hashes/issues/2)) Section 3.1 of [RFC7800] does not preclude the presence of both x5t#S256 and x5t#S512 in the same cnf claim. Including both would not represent confirmations for two different keys but rather two different hash confirmations of the same certificate. This may actually be useful during a transition period in possible future non-constrained deployment scenarios. The working group should determine whether to prohibit or allow this.]]

5.2. Resource Server Metadata

This document defines the mtls_confirmation_methods_supported Resource Server metadata parameter [RFC9728]. Its value is a JSON array containing the mutual-TLS confirmation method names that the Resource Server supports. Defined values are x5t#S256 and x5t#S512. If omitted, the default is ["x5t#S256"].

6. DPoP

OAuth 2.0 Demonstrating Proof of Possession (DPoP) [RFC9449] exclusively uses SHA-256 for all of its hash operations: the jkt confirmation method, the ath access token hash claim, and the dpop_jkt authorization code binding parameter. No alternative hash algorithms or extension points for hash algorithm negotiation are defined.

Section 11.10 of [RFC9449] anticipated the need for hash algorithm agility and foresaw that a future specification would define a new confirmation method, JWT claim, and authorization request parameter for use as alternatives to their SHA-256 counterparts. This document defines those DPoP mechanisms: the dpop_jkt_method authorization request parameter, the jkt#S512 confirmation method, and the ath#S512 JWT claim. In constrained deployments where SHA-256 is prohibited, these are used in place of their SHA-256 counterparts rather than alongside them.

6.1. Authorization Code Binding Methods

6.1.1. dpop_jkt_method Authorization Request Parameter

RFC 9449 [RFC9449] defines the dpop_jkt authorization request parameter as the JWK Thumbprint [RFC7638] of the DPoP public key using SHA-256. This document changes the definition of dpop_jkt to allow alternative hash algorithms indicated by the dpop_jkt_method parameter.

This document defines the dpop_jkt_method authorization request parameter, sent alongside dpop_jkt, to indicate the hash algorithm used to compute the JWK Thumbprint. The following method values are defined:

S256: JWK Thumbprint [RFC7638] using SHA-256, as originally defined in Section 10 of [RFC9449].

S512: JWK Thumbprint [RFC7638] using SHA-512.

For backwards compatibility, when dpop_jkt_method is absent from the authorization request, the Authorization Server MUST assume the value S256.

The value of dpop_jkt MUST be computed using the hash algorithm indicated by dpop_jkt_method.

6.1.2. Authorization Server Metadata

This document defines the dpop_jkt_methods_supported Authorization Server metadata parameter. Its value is a JSON array containing the dpop_jkt_method values that the Authorization Server supports.

An Authorization Server that supports dpop_jkt_method values beyond S256 MUST advertise its support by including the supported values in the dpop_jkt_methods_supported metadata parameter.

6.2. SHA-512 Hash Algorithms

6.2.1. jkt#S512 Confirmation Method

RFC 9449 [RFC9449] defines the jkt confirmation method member for binding access tokens to a DPoP public key using a SHA-256 JWK Thumbprint [RFC7638].

This document defines an analogous confirmation method member jkt#S512 that uses SHA-512 as the hash algorithm:

jkt#S512: The value is the base64url encoding of the JWK Thumbprint [RFC7638] computed using SHA-512 of the DPoP public key (in JWK format) to which the access token is bound.

When using jkt#S512, the Authorization Server computes the SHA-512 JWK Thumbprint of the DPoP public key and includes the result as the jkt#S512 member of the cnf claim in the access token (for JWT access tokens) or associates it with the token for later retrieval via token introspection [RFC7662].

The Resource Server MUST compute the SHA-512 JWK Thumbprint of the DPoP public key and compare it with the jkt#S512 value in the cnf claim. If the values do not match, the Resource Server MUST reject the request.

The choice of jkt#S512 over jkt is a deployment decision. It can be configured out of band or by the Authorization Server using the Resource Server’s metadata (Section 6.2.3).

[[TODO: (#2 (https://github.com/panva/draft-oauth-additional-hashes/issues/2)) Section 3.1 of [RFC7800] does not preclude the presence of both jkt and jkt#S512 in the same cnf claim. Including both would not represent confirmations for two different keys but rather two different hash confirmations of the same key. This may actually be useful during a transition period in possible future non-constrained deployment scenarios. The working group should determine whether to prohibit or allow this.]]

6.2.2. ath#S512 Access Token Hash

RFC 9449 [RFC9449] defines the ath claim in the DPoP proof JWT as the base64url-encoded SHA-256 hash of the ASCII encoding of the access token value.

This document defines an analogous claim ath#S512 that uses SHA-512 as the hash algorithm:

ath#S512: The value is the base64url encoding of the SHA-512 hash of the ASCII encoding of the associated access token’s value.

[[TODO: (#2 (https://github.com/panva/draft-oauth-additional-hashes/issues/2)) Including both ath and ath#S512 in the same DPoP proof JWT would not represent hashes of two different access tokens but rather two different hash confirmations of the same access token. This may actually be useful during a transition period in possible future non-constrained deployment scenarios. The working group should determine whether to prohibit or allow this.]]

The Resource Server MUST compute the SHA-512 hash of the ASCII encoding of the access token value and compare it with the ath#S512 value in the DPoP proof JWT. If the values do not match, the Resource Server MUST reject the request.

A Resource Server MAY signal the acceptable access token hash methods by including the ath_methods parameter in the WWW-Authenticate: DPoP challenge. The value of ath_methods is a space-delimited list of access token hash claim names that the Resource Server supports, analogous to the algs parameter defined in Section 7.1 of [RFC9449]. When ath_methods is absent, the Client MUST use ath. When ath_methods is present, the Client MUST use one of the listed methods. Additionally, Resource Server metadata for the supported access token hash methods is defined in Section 6.2.3.

The following is a non-normative example of an HTTP response signalling the client to use ath#S512:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: DPoP algs="Ed25519", ath_methods="ath#S512"
```

6.2.3. Resource Server Metadata

This document defines the following Resource Server metadata parameters [RFC9728]:

dpop_confirmation_methods_supported: JSON array containing the DPoP confirmation method names that the Resource Server supports. Defined values are jkt and jkt#S512. If omitted, the default is ["jkt"].

dpop_access_token_hash_methods_supported: JSON array containing the access token hash claim names that the Resource Server supports. Defined values are ath and ath#S512. If omitted, the default is ["ath"].

7. Security Considerations

The S512 code challenge method provides the same structural security properties as S256. It is a one-way transformation of the code verifier that prevents an attacker who intercepts the authorization code from computing the code verifier needed to exchange it for tokens.

The x5t#S512 confirmation method provides the same structural security properties as x5t#S256 defined in [RFC8705].

The jkt#S512 confirmation method, dpop_jkt combined with dpop_jkt_method parameter, and ath#S512 claim provide the same structural security properties as their SHA-256 counterparts defined in DPoP [RFC9449].

SHA-512 provides a 256-bit collision resistance and 512-bit preimage resistance, exceeding the 128-bit and 256-bit levels provided by SHA-256. The use of SHA-512 is suitable for deployments with elevated security requirements.

Deployments that do not have restrictions on use of SHA-256 do not need to migrate away from the established SHA-256 based mechanisms.

8. IANA Considerations

8.1. PKCE Code Challenge Method Registration

This document requests registration of the following value in the "PKCE Code Challenge Methods" registry established by Section 6.2 of [RFC7636]:

Code Challenge Method Parameter Name: S512

Change Controller: IETF

Specification Document(s): Section 4.1 of this document

8.2. DPoP Authorization Code Binding Methods Registry

This document establishes the "DPoP Authorization Code Binding Methods" registry for dpop_jkt_method values.

New entries are registered using the Specification Required policy [RFC5226].

The initial contents of the registry are:

Method Name: S256

Change Controller: IETF

Specification Document(s): Section 10 of [RFC9449]

Method Name: S512

Change Controller: IETF

Specification Document(s): Section 6.1.1 of this document

8.3. OAuth Parameters Registrations

This document requests registration of the following value in the "OAuth Parameters" registry established by [RFC6749]:

Parameter Name: dpop_jkt_method

Parameter Usage Location: authorization request

Change Controller: IETF

Specification Document(s): Section 6.1.1 of this document

8.4. OAuth Authorization Server Metadata Registration

This document requests registration of the following value in the "OAuth Authorization Server Metadata" registry established by [RFC8414]:

Metadata Name: dpop_jkt_methods_supported

Metadata Description: JSON array containing a list of the dpop_jkt_method values supported by the Authorization Server

Change Controller: IETF

Specification Document(s): Section 6.1.2 of this document

8.5. JWT Claims Registration

This document requests registration of the following value in the "JSON Web Token Claims" registry established by [RFC7519]:

Claim Name: ath#S512

Claim Description: The base64url-encoded SHA-512 hash of the ASCII encoding of the associated access token's value

Change Controller: IETF

Specification Document(s): Section 6.2.2 of this document

8.6. OAuth Protected Resource Metadata Registrations

This document requests registration of the following values in the "OAuth Protected Resource Metadata" registry established by [RFC9728]:

Metadata Name: dpop_confirmation_methods_supported

Metadata Description: JSON array containing a list of the DPoP confirmation method names supported by the Resource Server

Change Controller: IETF

Specification Document(s): Section 6.2.3 of this document

Metadata Name: dpop_access_token_hash_methods_supported

Metadata Description: JSON array containing a list of the access token hash claim names supported by the Resource Server

Change Controller: IETF

Specification Document(s): Section 6.2.3 of this document

Metadata Name: mtls_confirmation_methods_supported

Metadata Description: JSON array containing a list of the mutual-TLS confirmation method names supported by the Resource Server

Change Controller: IETF

Specification Document(s): Section 5.2 of this document

8.7. JWT Confirmation Methods Registrations

This document requests registration of the following values in the "JWT Confirmation Methods" registry established by [RFC7800]:

Confirmation Method Value: x5t#S512

Confirmation Method Description: X.509 Certificate SHA-512 Thumbprint

Change Controller: IETF

Specification Document(s): Section 5.1 of this document

Confirmation Method Value: jkt#S512

Confirmation Method Description: JWK SHA-512 Thumbprint

Change Controller: IETF

Specification Document(s): Section 6.2.1 of this document

9. References

9.1. Normative References

[OpenID.Discovery]
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0 incorporating errata set 2", December 2023, <https://openid.net/specs/openid-connect-discovery-1_0-errata2.html>.

[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>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/rfc/rfc5226>.

[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>.

[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>.

[RFC7636] Sakimura, N., Ed., Bradley, J., and N. Agarwal, "Proof Key for Code Exchange by OAuth Public Clients", RFC 7636, DOI 10.17487/RFC7636, September 2015, <https://www.rfc-editor.org/rfc/rfc7636>.

[RFC7638] Jones, M. and N. Sakimura, "JSON Web Key (JWK) Thumbprint", RFC 7638, DOI 10.17487/RFC7638, September 2015, <https://www.rfc-editor.org/rfc/rfc7638>.

[RFC7800] Jones, M., Bradley, J., and H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)", RFC 7800, DOI 10.17487/RFC7800, April 2016, <https://www.rfc-editor.org/rfc/rfc7800>.

[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>.

[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>.

[RFC8705] Campbell, B., Bradley, J., Sakimura, N., and T. Lodderstedt, "OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens", RFC 8705, DOI 10.17487/RFC8705, February 2020, <https://www.rfc-editor.org/rfc/rfc8705>.

[RFC9449] Fett, D., Campbell, B., Bradley, J., Lodderstedt, T., Jones, M., and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)", RFC 9449, DOI 10.17487/RFC9449, September 2023, <https://www.rfc-editor.org/rfc/rfc9449>.

[RFC9728] Jones, M.B., Hunt, P., and A. Parecki, "OAuth 2.0 Protected Resource Metadata", RFC 9728, DOI 10.17487/RFC9728, April 2025, <https://www.rfc-editor.org/rfc/rfc9728>.

9.2. Informative References

[cnsafaq] National Security Agency, "The Commercial National Security Algorithm Suite 2.0 and Quantum Computing FAQ", December 2024, <https://media.defense.gov/2022/Sep/07/2003071836/-1/-1/0/CSI_CNSA_2.0_FAQ_.PDF>.

[RFC7662] Richer, J., Ed., "OAuth 2.0 Token Introspection", RFC 7662, DOI 10.17487/RFC7662, October 2015, <https://www.rfc-editor.org/rfc/rfc7662>.

Acknowledgments

TODO acknowledge.

Document History

draft-skokan-oauth-additional-hashes-04

* Opened issues for spec TODOs and inlined their links

draft-skokan-oauth-additional-hashes-03

* Added Document History

* Changed ath_method to ath_methods (plural, space-delimited list), analogous to the algs parameter in Section 7.1 of [RFC9449]

* Removed premature "in place of ath" language for ath#S512, pending resolution of the dual-hash coexistence TODO

draft-skokan-oauth-additional-hashes-02

* Removed client-side MUST NOT requirements for using unadvertised PKCE and DPoP authorization code binding methods

draft-skokan-oauth-additional-hashes-01

* Changed hash algorithm from SHA-384 to SHA-512

* Added Purpose and Scope section

* Added Mutual-TLS section with x5t#S512 confirmation method and mtls_confirmation_methods_supported RS metadata

* Added dpop_confirmation_methods_supported RS metadata for DPoP

* Added WWW-Authenticate challenge parameter for access token hash method signalling

* Added TODO notes for dual-hash coexistence questions

* Expanded Security Considerations

draft-skokan-oauth-additional-hashes-00

* Initial draft

Author's Address

Filip Skokan
Okta
Email: panva.ip@gmail.com