Web Authorization Protocol
Internet-Draft
Intended status: Standards Track
Expires: 12 November 2026

A. Ambekar
eBay

11 May 2026

JSON Web Token (JWT) Profile for OAuth 2.0 Enveloped Proof of Possession (EPOP)
draft-ambekar-oauth-epop-00

Abstract

This specification defines a profile for OAuth 2.0 sender-constrained credentials in which authorization codes, access tokens, and refresh tokens are cryptographically bound to the client's private key as a single inseparable envelope. The profile extends sender-constraining beyond HTTP to non-HTTP transports including MQTT, Kafka, the Model Context Protocol (MCP), gRPC, and SASL-based protocols such as those defined in [RFC7628]. It introduces atomic proof-of-possession key rotation, enabling clients to rotate key pairs without disrupting active sessions, and an offline-derived client nonce (cnonce) that eliminates the server-issued nonce round-trip required by existing mechanisms — enabling stateless proof validation critical for non-HTTP and high-throughput deployments. Authorization servers, resource servers, and clients from different vendors can implement this profile interoperably.

About This Document

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

The latest revision of this draft can be found at https://asambeka.github.io/epop/draft-ambekar-oauth-epop.html. Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ambekar-oauth-epop/.

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/asambeka/epop.

Status of This Memo

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

This Internet-Draft will expire on 12 November 2026.

Copyright Notice

Table of Contents

1. Introduction 4
2. Conventions and Definitions 5
  2.1. Conformance 6
3. EPOP Token Profile 6
  3.1. Header 7
  3.2. Payload 7
4. Issuing and Constructing EPOP Tokens 8
5. Validating an EPOP Token 9
  5.1. Error Responses 10
  5.2. Nested Credential Validation 10
6. OAuth 2.0 Flows with EPOP 11
  6.1. Authorization Server Flows 11
    6.1.1. OAuth 2.0 Protocol Changes Summary 11
    6.1.2. Authorization Code Flow with PKCE 12
    6.1.3. Token Refresh Flow 14
    6.1.4. Token Introspection 15
    6.1.5. Token Exchange 16
    6.1.6. Token Revocation 17
    6.1.7. Pushed Authorization Requests 18
    6.1.8. Unsupported Flows 19

6.2. Resource Server Flows 19
6.2.1. Resource Access 19
6.2.2. Non-HTTP and Agentic Protocols 20
6.2.3. SASL Integration 21
6.2.4. UserInfo Endpoint 22
6.2.5. Resource Server Binding 23
7. Client Nonce (cnonce) 24
7.1. Overview 25
7.2. Notation 25
7.3. Computation 25
7.4. Verification 26
8. Discovery Metadata 26
8.1. Metadata Fields 26
8.2. AS Metadata Example 27
8.3. RS Metadata Example 28
9. Relationship to Other Specifications 28
10. Security Considerations 29
10.1. Token Lifetime 30
10.2. Replay Prevention 30
10.3. Token Substitution 30
10.4. Cross-Protocol Replay 30
10.5. Credential Confidentiality 31
10.6. Key Rotation Security 31
10.7. PKCE Requirement 31
10.8. Authorization Request Key Binding 31
10.9. Algorithm Selection 31
10.10. Intermediary Transparency 32
10.11. Private Key Protection 33
11. Privacy Considerations 33
11.1. Credential Visibility in Logs 33
11.2. Key Identifier as Tracking Vector 33
11.3. Resource URI Disclosure 34
11.4. Minimal Disclosure 34
12. IANA Considerations 34
12.1. HTTP Authentication Scheme Registration 34
12.2. OAuth Grant Type Registration 34
12.3. OAuth Parameters Registration 35
12.4. OAuth Token Type Registration 35
12.5. OAuth Token Type Identifiers 35
12.6. OAuth Token Type Hint Registration 36
12.7. JWT Claims Registration 36
12.8. JWT Type Registration 36
12.9. OAuth Authorization Server Metadata 37
12.10. OAuth Protected Resource Metadata 37
12.11. EPOP Request Context Members Registry 37
12.12. SASL Mechanism Registration 38
13. References 38
13.1. Normative References 38

13.2. Informative References 41
Acknowledgments 41
Author's Address 41

1. Introduction

OAuth 2.0 [RFC6749] access tokens are bearer tokens by default: any party in possession of a token can use it, regardless of whether that party is the legitimate client to which the token was issued. This property makes token theft a practical attack — intercepted tokens can be replayed without further credential material.

Sender-constraining mechanisms address this by cryptographically binding a token to the client's key pair so that possession of the token alone is insufficient to use it. Demonstrating Proof of Possession (DPoP) [RFC9449] introduced sender-constraining for HTTP-based OAuth flows. However, DPoP relies on HTTP-specific request parameters (htm, htu) and a server-issued nonce mechanism that requires an additional round-trip and imposes per-client nonce state management on servers — making it unsuitable for non-HTTP transports such as MQTT, Kafka, gRPC, and SASL-based protocols. No existing specification provides an interoperable sender-constraining profile that operates uniformly across both HTTP and non-HTTP transports.

A further deployment challenge with DPoP is the requirement to propagate two distinct HTTP headers — Authorization: DPoP <token> and DPoP: <proof> — as an inseparable pair through every layer of a distributed system. API gateways, reverse proxies, and service-mesh sidecars must each be updated to recognize and forward the DPoP header; many intermediaries strip non-standard headers by default. Every resource server onboarded to DPoP must separately implement dual-header awareness and proof validation. A single hop that discards the DPoP header silently invalidates the proof-of-possession guarantee for the entire request chain, creating a widespread integration burden across heterogeneous microservice deployments.

This document defines the Enveloped Proof of Possession (EPOP) profile for OAuth 2.0 credentials. In this profile, the OAuth credential — authorization code, access token, or refresh token — is nested within the ntk (Nested Token) claim (Section 3.2) of a signed JSON Web Token (JWT) [RFC7519] envelope. The entire structure, credential and proof together, is signed with the client's private key. The credential and the proof of possession are a single, inseparable cryptographic object: there is no credential without a proof.

The profile introduces a protocol-neutral rctx (Request Context) claim (Section 3.2) that replaces the HTTP-specific htm/htu claims of DPoP, enabling EPOP tokens to operate over any transport without protocol-specific adaptation. An offline-derived client nonce (cnonce) (Section 7) computed from public inputs eliminates the server-issued nonce round-trip required by [RFC9449], enabling stateless proof validation particularly suited to non-HTTP and high-throughput transports. The profile further defines atomic proof-of-possession key rotation, in which a client introduces a new key pair during a token refresh without disrupting the active session, and extends coverage to the full OAuth token lifecycle including token revocation and token exchange.

For SASL-based protocols, this document defines OAUTHEPOP, a new SASL mechanism extending [RFC7628] with sender-constraining support. All behaviors defined in [RFC7628] remain in effect; this document adds only the EPOP-specific authentication type and key binding verification.

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.

Conformance requirements for EPOP-issuing Authorization Servers, EPOP-validating Resource Servers, and EPOP Clients are summarized in Section 2.1.

The following terms are used throughout this document:

EPOP Token: A signed JWT ([RFC7519]) with typ: epop+jwt, signed by the client’s private key. Contains an OAuth 2.0 credential in the ntk claim when used for resource access, token refresh, token revocation, or token exchange. In authorization code exchange and PAR flows, contains only cnf.jkt for key binding; the authorization code travels as the standard code form parameter and is never embedded in ntk.

Nested Token (ntk): The OAuth 2.0 credential (access token, refresh token, or another EPOP token for key rotation) embedded inside an EPOP token’s payload. Not present in EPOP tokens used for authorization code exchange.

Request Context (rctx): A JSON object in the EPOP payload that

identifies the target resource and protocol action, replacing the HTTP-specific htm/htu claims of DPoP.

Client Nonce (cnonce): An HMAC value derived offline from the client's public key, an optional server-supplied seed, and a time-step counter, providing replay resistance without server-issued nonce state.

Authorization Server (AS): A server that issues OAuth 2.0 tokens to clients. As defined in [RFC6749].

Resource Server (RS): A server that hosts protected resources and accepts OAuth 2.0 tokens. As defined in [RFC6749].

Client: An application that requests OAuth 2.0 tokens and uses them to access protected resources. As defined in [RFC6749].

JWK Thumbprint: The SHA-256 thumbprint of a JSON Web Key, computed as defined in [RFC7638].

2.1. Conformance

This specification defines normative requirements for three conformance roles:

EPOP-issuing Authorization Server: An AS that issues EPOP-bound tokens MUST bind all issued tokens to the client's public key via cnf.jkt, MUST validate EPOP tokens presented at the token endpoint per Section 5, and MUST publish EPOP capability metadata per Section 8.

EPOP-validating Resource Server: An RS that accepts EPOP tokens MUST verify the outer envelope signature, MUST validate rctx members when rctx is present, and MUST verify cnf.jkt against the key in the EPOP token header per Section 5.

EPOP Client: A client producing EPOP tokens MUST sign each token with the private key whose public component appears in the EPOP token header, MUST NOT reuse jti values, and MUST derive cnonce per Section 7 when the AS requires it.

3. EPOP Token Profile

An EPOP token is a signed JWT ([RFC7519]) with typ: epop+jwt.

3.1. Header

typ REQUIRED. MUST be epop+jwt.

alg REQUIRED. Asymmetric signature algorithm. Edwards curve algorithms are RECOMMENDED for their superior security, performance, and payload compactness; see Section 10.9. Symmetric algorithms (HS*) and none MUST NOT be used.

jwk REQUIRED. The client's public key as a JWK ([RFC7517]). MUST NOT contain private key material.

Example header:

```
{
  "typ": "epop+jwt",
  "alg": "EdDSA",
  "jwk": {
    "kty": "OKP",
    "crv": "Ed25519",
    "x": "<base64url-encoded-x>"
  }
}
```

3.2. Payload

jti REQUIRED. Unique JWT ID with high entropy (see Section 10.2). Servers MUST maintain a replay cache keyed on jti.

iat REQUIRED. Issued-at Unix timestamp. Servers MUST reject tokens older than the server-defined maximum EPOP lifetime or issued in the future beyond clock skew. EPOP tokens MUST be short-lived, per-request credentials. The exp claim MUST NOT be included; the validity window is controlled entirely by the server's iat-based lifetime policy and, when cnonce is required, by epop_cnonce_step_seconds.

ntk CONDITIONAL. The nested OAuth 2.0 credential. REQUIRED for resource access, token refresh, token revocation, and introspection. OMITTED in authorization code exchange flows; the authorization code travels as the standard code form parameter. JWT credentials (access tokens, refresh tokens, and inner EPOP tokens for key rotation) are encoded as compact-serialized JWTs; opaque credentials are Base64URL-encoded opaque strings.

cnonce RECOMMENDED. Offline-derived client nonce (see Section 7). MUST be included when the server publishes epop_cnonce_required: true.

rctx OPTIONAL. Request context object. When present, all recognized members MUST be validated by the server. Unrecognized members MUST be ignored. Future rctx member names will be registered in the EPOP Request Context Members registry (Section 12.11).

rctx.res OPTIONAL. URI or URN of the target resource or endpoint.

rctx.method RECOMMENDED. Protocol action string. Case-insensitive for HTTP methods; case-sensitive otherwise.

rctx.id OPTIONAL. Client-generated correlation ID for async or multiplexed protocols.

cnf.jkt CONDITIONAL. SHA-256 JWK Thumbprint ([RFC7638]) of the client's public key. REQUIRED in authorization code exchange and in the inner envelope of a key rotation request. SHOULD be omitted on routine resource access and simple refresh where the key is already bound to the token.

Example payload:

```
{
  "jti": "A8B2B026-6C81-4A8C-A403-0F225E3DFEED",
  "iat": 1775749791,
  "ntk": "<credential>",
  "cnonce": "<base64url-hmac-value>",
  "rctx": {
    "res": "https://api.example.com/orders",
    "method": "GET",
    "id": "req_5521"
  },
  "cnf": {
    "jkt": "<sha256-thumbprint-of-public-key>"
  }
}
```

4. Issuing and Constructing EPOP Tokens

The client MUST have an asymmetric key pair (private key held exclusively by the client; public key embedded in every EPOP header), a reliable source of high-entropy identifiers for jti, and a trusted clock for iat.

An EPOP token is a JWS ([RFC7515]) signed with the client's private key, carrying the header and payload claims defined in Section 3.1 and Section 3.2. The compact serialization is transmitted as:

* *Token endpoint and PAR*: the epop form parameter in the POST body, preserving the Authorization header for client authentication.

* *Resource server*: Authorization: EPOP <compact-serialized-epop-token> per [RFC7235].

When the AS issues tokens in response to a valid EPOP request, it MUST bind the issued credential to the client's public key by including cnf.jkt — the SHA-256 JWK Thumbprint ([RFC7638]) of the EPOP token's jwk — in the response. For opaque tokens, the AS MUST record this binding server-side for use by the introspection endpoint. The AS MUST NOT issue EPOP-bound tokens unless the EPOP token has been successfully validated per Section 5.

The token endpoint response MUST include "token_type": "EPOP" for all EPOP-bound credentials, per [RFC6749] Section 5.1. This signals to the client that the issued token must be presented using the Authorization: EPOP scheme at the RS.

5. Validating an EPOP Token

To validate an EPOP token, the receiving server MUST ensure all of the following:

1. The typ header value is epop+jwt.

2. The jwk header is a valid asymmetric public key with no private key material.

3. The JWS signature verifies with the public key in jwk.

4. The iat is within an acceptable window, accounting for clock skew and the server's maximum EPOP lifetime policy.

5. The jti has not been seen before; record it and reject any future token presenting the same value.

6. If the server requires cnonce (i.e., epop_cnonce_required is true in its discovery document), the cnonce claim MUST be present. If cnonce is present, it MUST be valid for the current time-step (see Section 7.4).

7. If rctx is present, its members match the current request context; unrecognized members MUST be ignored.

8. If ntk is present, sha256(jwk) MUST equal cnf.jkt from the nested credential (see Section 5.2). This check MUST be performed after steps 13 pass.

These checks apply at both the AS (token endpoint, PAR) and RS (resource access). The server MUST reject the request with an appropriate error (see Section 5.1) if any check fails.

5.1. Error Responses

When a server rejects an EPOP token, it MUST respond as follows.

At the *token endpoint* (AS), validation failures MUST result in an HTTP 400 response with an OAuth error body per [RFC6749] Section 5.2. The error value MUST be invalid_request for structural failures (e.g., missing typ, invalid jwk format) and invalid_grant for credential failures (e.g., expired iat, replayed jti, invalid cnonce, cnf.jkt mismatch).

At the *resource server*, validation failures MUST result in an HTTP 401 response with a WWW-Authenticate challenge using the EPOP scheme per [RFC7235]:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: EPOP error="invalid_token",
                  error_description="EPOP token validation failed"
```

The error parameter MUST be invalid_token for any EPOP envelope or key binding failure. Servers MAY include error_description with a human-readable explanation. Servers MUST NOT reveal which specific validation step failed, as such information could aid an attacker.

5.2. Nested Credential Validation

When ntk is present and the outer envelope has passed steps 17, the server MUST perform the key binding check (step 8) and then validate the nested credential:

* *JWT (access or refresh token)*: verify signature, iss, exp, aud, and scopes.

* *Opaque token*: introspect with the AS per [RFC7662].

* *Inner EPOP (key rotation)*: apply steps 18 to the inner envelope; cnf.jkt in the inner envelope identifies the new key.

The key binding check requires sha256(outer jwk) == cnf.jkt from the nested credential. If this check fails, the server MUST reject the request with invalid_token.

6. OAuth 2.0 Flows with EPOP

6.1. Authorization Server Flows

6.1.1. OAuth 2.0 Protocol Changes Summary

Element	Change	Impacted Flows
grant_type	New values: epop_code_grant, epop_refresh_token	Authorization code exchange, token refresh
epop	New form parameter; carries compact-serialized EPOP token	Authorization code exchange, token refresh, PAR
actor_token_type	New values: urn:ietf:params:oauth:token-type:epop_access_token, urn:ietf:params:oauth:token-type:epop_refresh_token	Token exchange
token_type_hint	New value: epop_token	Token revocation, token introspection
Issued token response	AS MUST include cnf.jkt binding in all issued tokens	All grant flows
PKCE (code_challenge)	Elevated from RECOMMENDED to REQUIRED	Authorization code exchange, PAR

Table 1

6.1.2. Authorization Code Flow with PKCE

PKCE ([RFC7636]) is REQUIRED in all EPOP authorization code flows. PKCE and EPOP protect complementary attack surfaces: PKCE binds the authorization request to the token request; EPOP binds the authorization code to the client's key pair. Without PKCE, an attacker who intercepts the authorization code could generate their own key pair and wrap the code in a valid EPOP token signed with their key. Together they provide end-to-end chain of trust from the authorization request through token issuance (see Figure 1).

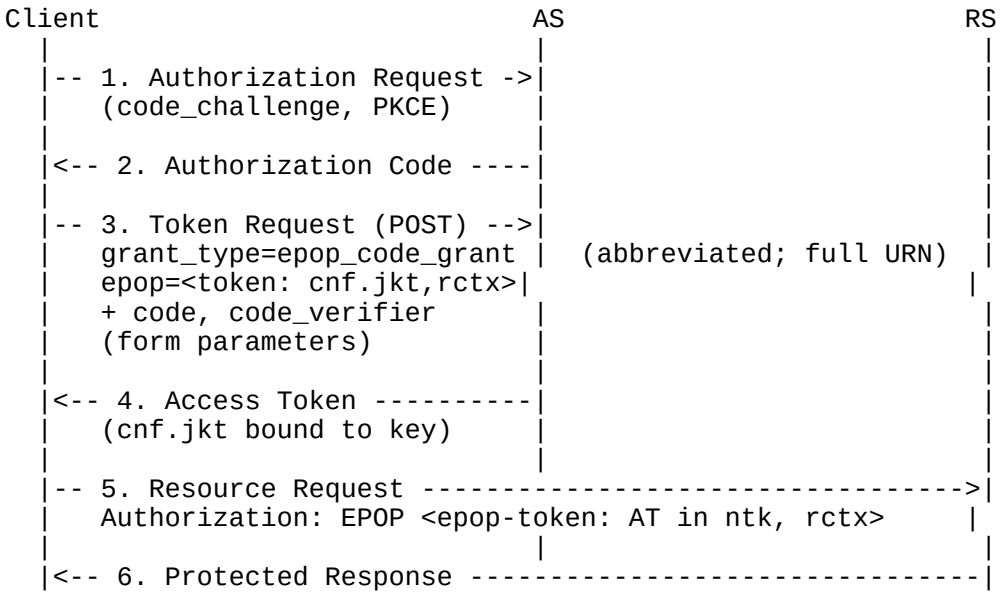

Figure 1: Authorization Code Flow with EPOP and PKCE

Token request (Step 3):

NOTE: '\' line wrapping per {{RFC8792}}

```
POST /token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Aepop_code_grant
&code=SplxlOBeZQQYbYS6WxSbIA
&client_id=s6BhdRkqt3
&redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
&code_verifier=bEaL42izcC-o-xBk0K2vuJ6U-y1p9r_wW2dFWIWgjz-
&epop=eyJ0eXAiOiJlcG9wK2p3dCIsImFsZyI6IkVkRFNBIiwiandrIjp7Imt0eSI6Ik9\
LUCIsImNydiI6IkVkMjU1MTkiLCJ4IjoiMTFxWUFZS3hDcmZWU183VHlXUUhPZzdoY3Z\
QYXBpTWxyd0lhYVBjSFVSbyJ9fQ.eyJqdGkiOiJBOEIyQjAyNi02QzgxLTRBOEMtQTQw\
My0wRjIyNUUzREZFRUQiLCJpYXQiOjE3NzU3NDk3OTEsImNuZiI6eyJqa3QiOiJrUHJL\
X3FteFZXYVlWQTl3d0JGNkl1bzN2Vnp6N1R4SENUd1hCeWdyUzRrIn0sInJjdHgiOnsi\
cmVzIjoiaHR0cHM6Ly9hcy5leGFtcGxlLmNvbS90b2tlbiIsIm1ldGhvZCI6IlBPU1Qi\
fX0.I45kzp-niCPrDZoHUA_n9vor9lqjBD7Pw3hNcaecAVkCKl2yyZIUqZseociCHt_U\
U60NFDLx6kEE8NWIR4aYAQ
```

Decoded EPOP token payload for this request (cnf.jkt declares the client's key binding; ntk is absent because the authorization code travels as the code form parameter per standard OAuth 2.0):

```
{
  "jti": "A8B2B026-6C81-4A8C-A403-0F225E3DFEED",
  "iat": 1775749791,
  "cnf": {
    "jkt": "kPrK_qmxVWaYVA9wwBF6Iuo3vVzz7TxHCTwXBygrS4k"
  },
  "rctx": {
    "res": "https://as.example.com/token",
    "method": "POST"
  }
}
```

Token endpoint response (Step 4):

```
{
  "access_token": "<compact-serialized-jwt-access-token>",
  "token_type": "EPOP",
  "expires_in": 3600
}
```

The issued access token returned by the AS carries the cnf.jkt binding:

```
{
  "sub": "jdoe@acme.org",
  "aud": ["https://api.example.com"],
  "cnf": {
    "jkt": "kPrK_qmxVWaYVA9wwBF6Iuo3vVzz7TxHCTwXBygrS4k"
  },
  "iat": 1775749800,
  "exp": 1775753400
}
```

6.1.3. Token Refresh Flow

The client wraps the refresh token in the ntk claim of an EPOP token and submits it using the epop_refresh_token grant type:

```
POST /token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Aepop_refresh_token
&client_id=s6BhdRkqt3
&epop=<compact-serialized-epop-token>
```

6.1.3.1. Simple Refresh

The EPOP token wraps the refresh token directly in ntk:

```
{
  "jti": "<unique-id>",
  "iat": "<unix-time>",
  "ntk": "<refresh-token-opaque-or-jwt>",
  "rctx": { "res": "https://as.example.com/token", "method": "POST" }
}
```

After standard EPOP validation (Section 5), the AS validates the nested refresh token. For an opaque token, the AS looks it up in the server-side store, verifies it is not revoked or expired, and confirms the associated key matches sha256(outer jwk). For a JWT refresh token, the AS verifies the signature, iss, exp, and jti, then confirms cnf.jkt == sha256(outer jwk).

6.1.3.2. Key Rotation Refresh

Key rotation uses a two-layer structure. The inner envelope is a Simple Refresh EPOP token (signed with the OLD key) with one addition: cnf.jkt set to the thumbprint of the NEW key. The outer envelope, signed with the NEW key, carries the compact-serialized inner EPOP token in ntk:

```
{
  "jti": "<unique-id-outer>",
  "iat": "<unix-time>",
  "ntk": "<compact-serialized-inner-epop-token>"
}
```

Standard EPOP validation (Section 5) applies to both envelopes. Beyond that, the AS verifies the key handoff chain: sha256(inner jwk) MUST match the key previously bound to the refresh token, and sha256(outer jwk) MUST equal inner cnf.jkt. On success, the AS issues new tokens bound to sha256(outer jwk), atomically updates the server-side key binding, and revokes the old key for this session. The old key’s authorization of the new key’s introduction, combined with the new key’s simultaneous proof of possession, provides a non-repudiable chain of custody.

6.1.4. Token Introspection

Token introspection ([RFC7662]) for EPOP tokens supports two modes.

6.1.4.1. Mode 1: Inner Token Introspection

The caller extracts the access token from the ntk claim of the received EPOP token and submits it to the introspection endpoint with the appropriate token_type_hint (e.g., access_token):

```
POST /introspect HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic <rs-credentials>

token=<access-token-extracted-from-ntk>
&token_type_hint=access_token
```

The introspection response includes cnf.jkt. The caller MUST verify that sha256(EPOP token signing key) equals the cnf.jkt returned in the response. This check confirms that the EPOP envelope was signed by the key legitimately bound to the nested token.

6.1.4.2. Mode 2: Full EPOP Token Introspection

The caller sends the entire EPOP token to the introspection endpoint with token_type_hint=epop_token:

```
POST /introspect HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic <rs-credentials>

token=<epop-token-received-from-client>
&token_type_hint=epop_token
```

The AS validates the EPOP envelope signature, extracts the credential from ntk, and validates it. The AS MUST verify that sha256(EPOP token signing key) equals the cnf.jkt in the nested token before returning a response. This check prevents a caller from presenting a valid EPOP envelope wrapping a token not bound to that key.

In both modes the introspection response includes cnf.jkt. For opaque tokens, cnf.jkt is the server-side registered client EPOP public key; for JWT tokens it is extracted from the token's own cnf.jkt claim. The token_type field reflects the type of the inner access token in ntk, not the outer EPOP envelope.

```
{
  "active": true,
  "token_type": "Bearer",
  "sub": "jdoe@acme.org",
  "iss": "https://as.example.com",
  "aud": ["https://api.example.com"],
  "scope": "read:orders",
  "iat": 1775748567,
  "exp": 1775752167,
  "cnf": {
    "jkt": "NLp8qGUJ1ywXs4ayYFLHfh8TA0crUe4g78UyBfx5j0Y"
  }
}
```

6.1.5. Token Exchange

Clients can exchange an EPOP-wrapped token for a new token using the token exchange framework ([RFC8693]). In this flow the client is the actor — it acts on behalf of a subject and proves its identity by presenting the EPOP token as the actor_token. The subject_token carries the token being exchanged on behalf of the subject; subject_token_type identifies its format. The actor_token_type identifies the EPOP token format using the epop_access_token token

type identifier (Section 12.5):

```
POST /token HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

grant_type=urn%3Aietf%3Aparams%3Aoauth%3Agrant-type%3Atoken-exchange
&subject_token=<subject-access-token>
&subject_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aaccess_token
&actor_token=<compact-serialized-epop-token>
&actor_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aepop_access_token
&requested_token_type=urn%3Aietf%3Aparams%3Aoauth%3Atoken-type%3Aepop_access_token
```

The AS MUST validate the EPOP envelope per Section 5 before processing the exchange. The AS MUST also validate the subject_token independently per its token type. The AS issues a new EPOP-wrapped access token bound to the same client public key, or to a new key if the exchange request includes key rotation.

Token exchange response:

```
{
  "access_token": "<compact-serialized-epop-bound-access-token>",
  "issued_token_type": "urn:ietf:params:oauth:token-type:epop_access_token",
  "token_type": "EPOP",
  "expires_in": 3600
}
```

The issued access token carries cnf.jkt bound to the client’s public key, as in all EPOP flows.

6.1.6. Token Revocation

Clients revoke EPOP-wrapped tokens using the token revocation framework ([RFC7009]). The client constructs and signs the EPOP token containing the credential to be revoked, then submits it as the token parameter with a token_type_hint identifying the credential type:

```
POST /revoke HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

token=<compact-serialized-epop-token>
&token_type_hint=epop_token
```

The AS MUST validate the EPOP envelope per Section 5 before processing revocation. The token_type_hint value MUST be epop_token (Section 12.6).

The requirement that the client construct and sign the EPOP envelope provides proof-of-possession on revocation: an attacker who captured a token but does not hold the corresponding private key cannot revoke it. This is a deliberate security improvement over plain RFC 7009 revocation, where any party holding the raw token value can submit a revocation request.

6.1.7. Pushed Authorization Requests

For PAR ([RFC9126]), the client MAY declare its public key fingerprint at the PAR endpoint to pre-bind the authorization code to the client's key before the browser redirect. The client includes the EPOP token as the epop form parameter — with cnf.jkt and rctx but no ntk — alongside the standard PAR request parameters. The Authorization header carries client authentication as usual. PKCE parameters (code_challenge, code_challenge_method) MUST be included in the PAR request (this specification elevates PKCE from RECOMMENDED in [RFC9126] to REQUIRED in all EPOP flows); the subsequent token endpoint request MUST include code_verifier.

EPOP token payload for the PAR request:

```
{
  "jti": "<unique-id>",
  "iat": "<unix-time>",
  "cnf": { "jkt": "<client-public-key-thumbprint>" },
  "rctx": {
    "res": "https://as.example.com/par",
    "method": "POST"
  }
}
```

HTTP request:

```
POST /par HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW

response_type=code
&client_id=s6BhdRkqt3
&redirect_uri=https%3A%2F%2Fclient.example.com%2Fcb
&scope=read%3Aorders
&code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
&code_challenge_method=S256
&epop=<compact-serialized-epop-token>
```

The AS verifies the EPOP signature, extracts cnf.jkt, and records it against the request_uri it returns. Once a cnf.jkt is registered via PAR, that key binding is final for the lifetime of the resulting authorization code. If the EPOP token presented at the token endpoint declares a different cnf.jkt than the one recorded at the PAR endpoint, the AS MUST reject the request.

6.1.8. Unsupported Flows

6.1.8.1. Client Credentials Flow

Client credentials flow is not covered by this specification.

6.1.8.2. Authorization Request Key Binding Not Supported

Binding a public key thumbprint inside the authorization request URL (analogous to dpop_jkt in [RFC9449] Section 10) is not supported; see Section 10.8 for the security rationale.

6.2. Resource Server Flows

6.2.1. Resource Access

The client sends the EPOP token in the HTTP Authorization header using the EPOP authentication scheme ([RFC7235]). No Authorization: Bearer header is used.

EPOP token payload:

```
{
  "jti": "626545DF-CD19-48EA-BB85-974130E012B5",
  "iat": 1775749791,
  "ntk": "<compact-serialized-access-token>",
  "rctx": {
    "res": "https://api.example.com/orders",
    "method": "GET"
  }
}
```

HTTP request:

```
GET /orders HTTP/1.1
Host: api.example.com
Authorization: EPOP <compact-serialized-epop-token>
```

The RS verifies the outer envelope, then extracts and validates the access token from ntk as defined in Section 5.2.

6.2.2. Non-HTTP and Agentic Protocols

The EPOP token structure is identical for non-HTTP protocols; only rctx values differ. The rctx.res field accommodates any URI or URN:

Protocol	rctx.res	rctx.method
HTTPS	https://api.example.com/orders	GET
MQTT	urn:mqtt:broker:sensors/temperature	PUBLISH
MCP	urn:mcp:server:filesystem	tools/call
Kafka	urn:kafka:cluster:orders-topic	Produce
gRPC	urn:grpc:service:helloworld.Greeter	SayHello

Table 2

The rctx.id field lets the server correlate the EPOP token with an asynchronous response — critical in multiplexed or streaming protocols where request/response pairs are not strictly sequential:

```
{
  "jti": "9F3A1C22-4D87-4B3E-BC12-0A5E8D7F1234",
  "iat": 1775750000,
  "ntk": "<compact-serialized-access-token>",
  "rctx": {
    "res": "urn:mcp:server:filesystem",
    "method": "tools/call",
    "id": "req_5521"
  }
}
```

6.2.3. SASL Integration

This section extends [RFC7628] to support Enveloped Proof of Possession. All behaviors defined in [RFC7628] — including the GS2 ([RFC5801]) message structure, connection-establishment scope, server error challenge format, and error handling — apply to OAUTHEPOP unless explicitly stated otherwise in this section.

6.2.3.1. OAUTHEPOP Mechanism

OAUTHEPOP is a new SASL mechanism following the structure of [RFC7628] Section 3. Implementors should note that EPOP compact serializations (header, payload, and signature) may be significantly larger than a plain bearer token; SASL implementations MUST support initial client response buffers large enough to carry the full compact-serialized EPOP JWT. It introduces EPOP as the OAuth authentication type for the auth field of the GS2 initial client response.

The auth field for OAUTHEPOP is defined as:

auth-field = "auth" "=" "EPOP" SP epop-token kvsep

where epop-token is the compact-serialized EPOP JWT as defined in Section 3 and kvsep is %x01 per [RFC7628]. All other fields in the GS2 initial client response (host, port, GS2 header, final kvsep) are unchanged from [RFC7628] Section 3.

Example initial client response (using %x01 represented as <SOH>):

```
n,,<SOH>auth=EPOP <compact-epop-token><SOH>host=mail.example.com<SOH>port=993<SOH><SOH>
```

6.2.3.2. OAUTHBEARER Backward Compatibility

Servers advertising OAUTHBEARER MAY also accept EPOP tokens by recognizing auth=EPOP in the initial client response. When a server receives auth=EPOP over the OAUTHBEARER mechanism, it MUST treat the value as a compact-serialized EPOP JWT and apply the key binding check defined in Section 5.2.

The server determines whether the nested access token in ntk is EPOP-bound as follows:

* *Opaque nested access token*: The server calls token introspection ([RFC7662]). If the introspection response includes cnf.jkt, the token is EPOP-bound; the server MUST verify sha256(outer jwk) == cnf.jkt before accepting the connection.

* *JWT nested access token*: The server inspects the typ claim of the access token extracted from ntk. If typ == "epop+jwt", the token is EPOP-bound; the server MUST apply the same key binding check.

Servers SHOULD advertise OAUTHEPOP in SASL capability responses when EPOP is supported and SHOULD list it ahead of OAUTHBEARER. Clients that support EPOP MUST use OAUTHEPOP when the server advertises it.

6.2.4. UserInfo Endpoint

The client presents an EPOP token wrapping the access token at the UserInfo endpoint:

```
{
  "jti": "<unique-id>",
  "iat": "<unix-time>",
  "ntk": "<access-token>",
  "rctx": {
    "res": "https://as.example.com/userinfo",
    "method": "GET"
  }
}
```

```
GET /userinfo HTTP/1.1
Host: as.example.com
Authorization: EPOP <compact-serialized-epop-token>
```

The AS validates the EPOP token per Section 5, verifying rctx.res matches its UserInfo endpoint URI, then returns UserInfo claims.

6.2.5. Resource Server Binding

6.2.5.1. Key Binding Enforcement

Every JWT access token issued under EPOP carries a cnf.jkt claim — the SHA-256 thumbprint of the client's EPOP public key. When the RS receives an EPOP-wrapped request:

1. It verifies the outer EPOP signature using the jwk embedded in the header.

2. It extracts the access token from ntk and validates it (signature, exp, aud, scopes).

3. It computes sha256(outer jwk header key) and compares it to cnf.jkt inside the access token.

If the two values differ, the RS MUST reject the request with invalid_token. This check prevents an attacker from wrapping a stolen access token in their own EPOP envelope.

6.2.5.2. Early-Exit Pattern

The rctx check (Section 5 Step 7) SHOULD occur before nested credential validation (Section 5.2). If rctx.res or rctx.method does not match the current request, the RS SHOULD reject immediately without decoding or verifying the access token. This is the primary defense against replay of an EPOP token from one endpoint at another within the token's short validity window.

6.2.5.3. RS Handling of Opaque Access Tokens

When the access token in ntk is opaque, the RS MUST introspect the received EPOP token with the AS ([RFC7662]), passing it as the token parameter with token_type_hint=epop_token so the AS can validate the envelope and extract the inner token:

```
POST /introspect HTTP/1.1
Host: as.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic <rs-credentials>

token=<epop-token-received-from-client>
&token_type_hint=epop_token
```

The introspection response includes the cnf.jkt claim:

```
{
  "active": true,
  "sub": "jdoe@acme.org",
  "scope": "read:orders",
  "cnf": {
    "jkt": "NLp8qGUJ1ywXs4ayYFLHfh8TA0crUe4g78UyBfx5j0Y"
  },
  "exp": 1775752167
}
```

6.2.5.4. RS State Management for Opaque Tokens

The RS MUST NOT cache introspection results beyond the EPOP token's validity window. Since EPOP tokens are very short duration per-request credentials, a stale introspection cache could cause the RS to validate a post-rotation request against the pre-rotation key.

When the client performs key rotation (Section 6.1.3.2), the AS MUST atomically update its server-side registry before issuing any new tokens. Introspection responses for access tokens issued after rotation MUST reflect the new client EPOP public key.

Scenario	RS Obligation
Opaque access token, no caching	Call introspection per request; use returned cnf.jkt for binding check
Opaque access token, cached introspection	Cache MUST expire at or before the EPOP token validity window; MUST NOT reuse across key rotation
Key mismatch detected	Reject with invalid_token; MUST NOT fall back to a previously cached key
Opaque refresh token key rotation (AS-side)	AS MUST atomically update server-side key registration; partial update MUST NOT be possible

Table 3

7. Client Nonce (cnonce)

7.1. Overview

cnonce is an offline-derived nonce that time-bounds each EPOP token to a discrete step window. Unlike the DPoP server-issued nonce ([RFC9449] Section 8), it requires no server round-trip and carries no server-side nonce state, making it particularly suited to non-HTTP transports and high-throughput deployments. cnonce reduces the replay opportunity within a window; the jti replay cache (Section 5 Step 5) remains the primary replay prevention mechanism and MUST be maintained regardless.

7.2. Notation

The following notation is used in this section:

SPKI DER-encoded SubjectPublicKeyInfo of the client public key in the EPOP token header.

X Time-step duration in seconds, taken from epop_cnonce_step_seconds.

seed Optional 32-byte deployment seed from epop_cnonce_seed. When absent, treated as the empty octet string.

HKDF-SHA256(ikm, salt, info, len) HKDF with SHA-256 per [RFC5869], producing len octets.

HMAC-SHA256(key, data) HMAC with SHA-256 per [RFC2104].

BASE64URL(octets) Base64url encoding without padding per [RFC4648] Section 5.

UTF8(str) UTF-8 encoding of string str per [RFC3629].

UINT64BE(n) 8-octet big-endian encoding of integer n.

|| Octet string concatenation.

7.3. Computation

The time-step counter is:

T = floor(CurrentUnixTime / X)

The per-client key material is derived once per key pair and re-derived on seed rotation:

```
key_material = HKDF-SHA256(seed || SPKI, SHA256(SPKI), "epop-cnonce-v1", 32)
```

When seed is absent, ikm = SPKI. The optional seed scopes derivation per deployment so that cnonce values from clients under different seeds are non-interchangeable.

The cnonce value is:

```
cnonce = BASE64URL(HMAC-SHA256(key_material, UTF8(jti) || UINT64BE(T)))
```

where jti is the EPOP token's own unique identifier.

7.4. Verification

The server (AS or RS) derives key_material identically and verifies:

```
cnonce == BASE64URL(HMAC-SHA256(key_material, UTF8(jti) || UINT64BE(t)))
```

for any $t \in \{T-1, T, T+1\}$. This window absorbs clock skew of up to one step. If no value of t satisfies the equality, the server MUST reject the token. A cnonce satisfying the time-window check but carrying a replayed jti is caught by the replay cache (Section 5 Step 5).

When epop_cnonce_seed is rotated, the server MUST update epop_cnonce_seed_id simultaneously so that clients re-derive key_material.

8. Discovery Metadata

Authorization Servers that support EPOP MUST publish their capabilities in the OAuth Authorization Server Metadata document ([RFC8414]), available at /.well-known/oauth-authorization-server (or /.well-known/openid-configuration for OpenID Connect providers). Resource Servers publish EPOP capabilities in the OAuth Protected Resource Metadata document ([RFC9728]), available at /.well-known/oauth-protected-resource.

8.1. Metadata Fields

The epop_cnonce_seed, epop_cnonce_seed_id, and epop_cnonce_step_seconds values MUST be identical in the AS and RS discovery documents. Clients SHOULD validate this consistency on startup and after any seed rotation.

epop_supported String; AS and RS. REQUIRED when EPOP is active. Server EPOP posture: "disabled" — clients MUST NOT send EPOP tokens; "recommended" — EPOP accepted but non-EPOP requests also accepted; "required" — requests without a valid EPOP token MUST be rejected.

epop_ntk_types_supported Array of strings; AS and RS. REQUIRED when epop_supported is not "disabled". Credential formats accepted inside ntk: "jwt" and/or "opaque".

epop_key_rotation_supported Boolean; AS only. Default: false. When true, the server supports the two-layer EPOP key rotation flow (Section 6.1.3.2). Clients MUST check this field before attempting key rotation.

epop_cnonce_required Boolean; AS and RS. Default: false. When true, the server MUST reject EPOP tokens that omit cnonce. Clients MUST include cnonce if either the AS or RS sets this to true.

epop_cnonce_step_seconds Integer; AS and RS. REQUIRED when epop_cnonce_required is true. Time-step duration in seconds; both client and server compute T = floor(utc_now() / epop_cnonce_step_seconds). MUST be identical in AS and RS documents.

epop_cnonce_seed String (Base64URL, 32 bytes); AS and RS. OPTIONAL. Namespace discriminator for multi-tenant deployments; mixed into the per-client HKDF derivation so cnonce values across tenants are non-interchangeable. Not a secret. MUST be identical in AS and RS documents when present. Rotated in coordination with epop_cnonce_seed_id.

epop_cnonce_seed_id String; AS and RS. REQUIRED when epop_cnonce_seed is present; OPTIONAL otherwise. Opaque identifier for the current epop_cnonce_seed. Clients MUST cache the discovery document keyed on this value and re-derive key_material only when it changes. MUST be identical in AS and RS documents.

8.2. AS Metadata Example

```
{
  "issuer": "https://as.example.com",
  "token_endpoint": "https://as.example.com/token",
  "epop_supported": "recommended",
  "epop_ntk_types_supported": ["jwt", "opaque"],
  "epop_key_rotation_supported": true,
  "epop_cnonce_required": true,
  "epop_cnonce_step_seconds": 30,
  "epop_cnonce_seed": "<base64url-32-bytes>",
  "epop_cnonce_seed_id": "seed-2026-q2"
}
```

8.3. RS Metadata Example

```
{
  "resource": "https://api.example.com",
  "authorization_servers": ["https://as.example.com"],
  "epop_supported": "required",
  "epop_ntk_types_supported": ["jwt", "opaque"],
  "epop_cnonce_required": true,
  "epop_cnonce_step_seconds": 30,
  "epop_cnonce_seed": "<base64url-32-bytes>",
  "epop_cnonce_seed_id": "seed-2026-q2"
}
```

9. Relationship to Other Specifications

RFC	Title	Relationship
[RFC7628]	A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth	EPOP extends RFC 7628 by introducing EPOP as a new OAuth authentication type for the SASL auth field and defining OAUTHEPOP as a new SASL mechanism. All behaviors defined in RFC 7628 remain in effect; this document adds only the auth=EPOP field value and the key binding check that OAUTHBEARER servers apply when they encounter it.
[RFC9449]	OAuth 2.0 Demonstrating Proof of Possession (DPoP)	EPOP generalizes the DPoP proof model: replaces htm/htu with rctx for protocol agnosticism; replaces the ath hash with ntk embedding so credential and proof travel as one object; extends coverage to authorization codes and refresh tokens; adds atomic key rotation and the offline cnonce. Additionally, DPoP’s requirement to carry two coordinated HTTP headers (Authorization and DPoP) imposes a propagation burden on every intermediary and resource server in a distributed system; EPOP eliminates this by enveloping the proof inside the token, so a single Authorization header carries the

		inseparable credential-and-proof object through all hops without requiring middleware changes.
[RFC7636]	Proof Key for Code Exchange (PKCE)	EPOP elevates PKCE from RECOMMENDED to REQUIRED in all authorization code flows.
[RFC8414]	OAuth 2.0 Authorization Server Metadata	EPOP adds new discovery fields to the well-known document: epop_supported, epop_ntk_types_supported, epop_key_rotation_supported, and the epop_cnonce_* family.
[RFC7638]	JSON Web Key (JWK) Thumbprint	EPOP uses the SHA-256 JWK thumbprint (cnf.jkt) as its primary key binding primitive — embedded in every issued token and checked by the RS as its main defense against token substitution.
[RFC7662]	OAuth 2.0 Token Introspection	EPOP extends introspection responses to include cnf.jkt. For opaque tokens the AS returns the server-side registered client EPOP public key; for JWT tokens it is extracted from the token's own claim.
[RFC9126]	OAuth 2.0 Pushed Authorization Requests (PAR)	EPOP extends PAR to let the client declare cnf.jkt at the earliest point in the flow, pre-binding the authorization code before the browser redirect.
[RFC5869]	HMAC-based Key Derivation Function (HKDF)	EPOP uses HKDF-SHA256 to derive a per-client key from the optional server-controlled epop_cnonce_seed and the client's public key, enabling stateless offline cnonce computation.

Table 4

10. Security Considerations

10.1. Token Lifetime

EPOP tokens MUST be short-lived, per-request credentials. The server defines the maximum acceptable age of the iat claim; when cnonce is required, epop_cnonce_step_seconds imposes an additional validity bound enforced independently by both the AS and RS. Short lifetimes limit the window during which a captured token remains usable and reduce the cost of maintaining the jti replay cache.

10.2. Replay Prevention

Each EPOP token MUST include a jti value of at least 128 bits of entropy (e.g., a UUID v4 or CSPRNG-generated value) to make collisions computationally infeasible. Servers MUST maintain a replay cache keyed on jti with a TTL of at least max_epop_lifetime + clock_skew; any token whose jti appears in the cache MUST be rejected. When cnonce is also required, the time-step window narrows the effective validity period further but does not replace the cache.

10.3. Token Substitution

The key binding check — sha256(outer jwk) == cnf.jkt from the nested credential — is the primary defense against token substitution. An attacker who captures an access token and wraps it in an EPOP envelope signed with their own key will produce a thumbprint that does not match the cnf.jkt embedded by the AS; the RS MUST reject the mismatch.

For opaque access tokens, the RS MUST NOT cache introspection results beyond the EPOP token’s validity window. When a client performs key rotation (Section 6.1.3.2), the AS MUST atomically update its server-side key binding before issuing new tokens; a stale cached result could cause the RS to validate a post-rotation request against the pre-rotation key.

10.4. Cross-Protocol Replay

Without rctx validation, an EPOP token captured from one protocol or endpoint could be replayed at another within the token’s validity window. Servers MUST validate rctx.res and rctx.method when those claims are present. Clients SHOULD always include rctx to maximize replay resistance.

10.5. Credential Confidentiality

The ntk claim embeds the full OAuth 2.0 credential inside the EPOP envelope. Unlike bearer token flows where only the token value is sensitive, the entire compact serialization carries sensitive material. TLS is REQUIRED for all EPOP token transmissions; JWE MAY be applied for additional confidentiality over transports that cannot guarantee channel security. Refresh tokens embedded in ntk are particularly sensitive. Servers and intermediaries MUST NOT log EPOP token values in plaintext; the jti claim SHOULD be used as the audit correlation identifier instead.

10.6. Key Rotation Security

The AS MUST validate both the outer and inner EPOP envelopes completely before issuing new tokens or updating key bindings. Partial validation would allow an attacker who holds a captured refresh token to inject a new key by constructing a valid outer envelope while providing an invalid inner envelope.

10.7. PKCE Requirement

PKCE ([RFC7636]) is REQUIRED in all EPOP authorization code flows. Without PKCE, an attacker who intercepts the authorization code can generate their own key pair and wrap the code in a valid EPOP token signed with that key, bypassing key binding entirely. PKCE and EPOP protect complementary surfaces: PKCE binds the authorization request to the token request; EPOP binds the code to the client’s key pair.

10.8. Authorization Request Key Binding

Binding a public key thumbprint inside the authorization request URL (as defined for DPoP in [RFC9449] Section 10) is not supported by this specification. Authorization requests travel through the browser redirect — an untrusted channel where a parameter can be silently replaced before the AS sees it. EPOP establishes key binding exclusively at endpoints where the client communicates directly with the AS over TLS: the token endpoint and, optionally, the PAR endpoint (Section 6.1.7).

10.9. Algorithm Selection

EPOP tokens are generated per-request at high frequency; algorithm choice directly affects signing latency, token size, and security posture. The jwk embedded in every EPOP header makes key footprint particularly significant for constrained transports. Edwards curve algorithms are RECOMMENDED. Ed25519 is the primary choice — smallest public key, fastest deterministic signing, and 128-bit security

adequate for short-lived credentials. Ed448 is appropriate for high-assurance environments requiring a larger security margin. ES256 is acceptable where Edwards curves are unavailable. RSA algorithms SHOULD NOT be used in new implementations. Implementations MUST follow [RFC8725].

Property	EdDSA / Ed25519	EdDSA / Ed448	ES256 (P-256)	RS256 (RSA-2048)
Security level	128-bit	224-bit	128-bit	112-bit
Signature size	64 bytes	114 bytes	64 bytes	256 bytes
Public key size	32 bytes	57 bytes	64 bytes	~256 bytes
Signing speed	Very fast (deterministic)	Fast (deterministic)	Moderate	Slow
Side-channel resistance	Strong (constant-time)	Strong (constant-time)	Moderate	Weak

Table 5

10.10. Intermediary Transparency

Because the EPOP proof is embedded within the token rather than transmitted as a separate header, EPOP tokens are transparent to intermediaries that forward the Authorization header without modification. Unlike DPoP, where loss of the DPoP header at any hop silently breaks proof-of-possession, EPOP's enveloped structure ensures that the proof travels with the credential through every layer of a distributed system. Resource servers MUST NOT accept EPOP tokens from which the outer envelope signature has been stripped or replaced by an intermediary; the full compact-serialized EPOP token MUST be forwarded unchanged.

10.11. Private Key Protection

The security of EPOP depends entirely on the client's private key remaining secret. Private key material MUST never be logged, serialized into application state, or transmitted over any channel. Implementations MUST verify that no private key fields (d, p, q, dp, dq, qi) are present in the jwk header parameter before accepting or forwarding an EPOP token.

On native and mobile platforms, clients MUST use platform secure storage (e.g., Android Keystore, iOS Secure Enclave), with private key operations performed inside the secure element where available so that key material never enters application memory.

In browser environments, private keys MUST be generated as non-extractable CryptoKey objects via the Web Crypto API (extractable: false). Clients MUST NOT store private keys in localStorage, sessionStorage, IndexedDB, or any JavaScript-readable store; this prevents exfiltration by XSS or injected scripts running in the same origin.

EPOP tokens MUST use asymmetric signature algorithms. Symmetric algorithms such as HS256 require the verifier to hold the same secret as the signer, making independent third-party verification impossible and introducing shared-secret distribution risk.

11. Privacy Considerations

11.1. Credential Visibility in Logs

Because the ntk claim embeds the full OAuth 2.0 credential, an EPOP token is more privacy-sensitive than a typical bearer token: its compact serialization reveals the credential type, issuer, audience, and scope to any party that receives or stores it. Servers SHOULD apply data-minimization practices to audit logs — retaining only the jti and iat claims rather than the full token value. See Section 10.5 for the normative logging and TLS requirements.

11.2. Key Identifier as Tracking Vector

The cnf.jkt claim is a stable, long-lived public key thumbprint. If the same key pair is used across multiple authorization server or resource server deployments, the thumbprint functions as a cross-context tracking identifier. Clients SHOULD use distinct key pairs per authorization server to limit cross-context correlation. Key rotation (Section 6.1.3.2) provides a mechanism for periodic key refresh independent of active sessions.

11.3. Resource URI Disclosure

The rctx.res field encodes the target resource URI or URN and is part of the signed EPOP envelope, visible to any party that receives or logs the token. Clients MUST use TLS for all EPOP token transmissions to limit exposure to on-path observers. Resource identifiers that encode sensitive information (e.g., user identifiers embedded in path parameters) SHOULD be avoided in rctx.res.

11.4. Minimal Disclosure

This profile does not require the EPOP envelope to carry user identity claims. User identity information belongs in the nested access token (ntk), not in the outer EPOP envelope. Implementations MUST NOT add user identity claims to the EPOP token header or payload unless required by a specific profile extension.

12. IANA Considerations

12.1. HTTP Authentication Scheme Registration

This specification requests registration of the following entry in the "Hypertext Transfer Protocol (HTTP) Authentication Scheme Registry" ([RFC7235]):

Authentication Scheme Name: EPOP

Pointer to specification text: Section 6.2.1 and Section 3 of this document.

12.2. OAuth Grant Type Registration

This specification requests registration of the following entries in the "OAuth Parameters" registry (grant type sub-table) established by [RFC6749] Section 11.3:

Grant Type Name: urn:ietf:params:oauth:grant-type:epop_code_grant

Change Controller: IETF

Specification Document(s): Section 6.1.2 of this document

Grant Type Name: urn:ietf:params:oauth:grant-type:epop_refresh_token

Change Controller: IETF

Specification Document(s): Section 6.1.3 of this document

12.3. OAuth Parameters Registration

This specification requests registration of the following entry in the "OAuth Parameters" registry established by [RFC6749] Section 11.2:

Parameter Name: epop

Parameter Usage Location: token request, pushed authorization request

Change Controller: IETF

Specification Document(s): Section 4, Section 6.1.2, Section 6.1.3, and Section 6.1.7 of this document

12.4. OAuth Token Type Registration

This specification requests registration of the following value in the "OAuth Access Token Types" registry established by [RFC6749] Section 11.1:

Type Name: EPOP

Additional Token Endpoint Response Parameters: (none)

HTTP Authentication Scheme(s): EPOP

Change Controller: IETF

Specification Document(s): Section 4 and Section 6.2.1 of this document

12.5. OAuth Token Type Identifiers

This specification requests registration of the following entries in the "OAuth URI" subregistry of the "OAuth Parameters" registry established by [RFC8693] Section 7.1:

URI: urn:ietf:params:oauth:token-type:epop_access_token

Change Controller: IETF

Specification Document(s): Section 6.1.5 of this document

URI: urn:ietf:params:oauth:token-type:epop_refresh_token

Change Controller: IETF

Specification Document(s): Section 6.1.5 of this document

12.6. OAuth Token Type Hint Registration

This specification requests registration of the following value in the "OAuth Token Type Hints" registry established by [RFC7009] Section 4.1:

Token Type Hint Name: epop_token

Change Controller: IETF

Specification Document(s): Section 6.1.4, Section 6.1.6, and Section 6.2.5.3 of this document

12.7. JWT Claims Registration

This specification requests registration of the following JWT claims in the "JSON Web Token Claims" registry established by [RFC7519]:

Claim Name: ntk Claim Description: Nested Token — the OAuth 2.0 credential embedded inside the EPOP envelope.

Change Controller: IETF

Specification Document(s): Section 3.2 of this document

Claim Name: rctx Claim Description: Request Context — a JSON object identifying the target resource and protocol action.

Change Controller: IETF

Specification Document(s): Section 3.2 of this document

Claim Name: cnonce Claim Description: Client Nonce — offline-derived HMAC value for replay resistance without server-issued nonce state.

Change Controller: IETF

Specification Document(s): Section 7 of this document

12.8. JWT Type Registration

This specification requests registration of the following typ header parameter value in the "JSON Web Signature and Encryption Header Parameters" registry established by [RFC7515], in accordance with [RFC8725] Section 3.11:

Type Name: epop+jwt Description: Enveloped Proof of Possession JWT.

Change Controller: IETF

Specification Document(s): Section 3.1 of this document

12.9. OAuth Authorization Server Metadata

This specification requests registration of the following names in the "OAuth Authorization Server Metadata" registry ([RFC8414]):

* epop_supported
* epop_ntk_types_supported
* epop_key_rotation_supported
* epop_cnonce_seed
* epop_cnonce_step_seconds
* epop_cnonce_seed_id
* epop_cnonce_required

12.10. OAuth Protected Resource Metadata

This specification requests registration of the following names in the "OAuth Protected Resource Metadata" registry ([RFC9728]):

* epop_supported
* epop_ntk_types_supported
* epop_cnonce_seed
* epop_cnonce_step_seconds
* epop_cnonce_seed_id
* epop_cnonce_required

12.11. EPOP Request Context Members Registry

This specification requests creation of a new registry, "EPOP Request Context Members", under the "OAuth Parameters" registry group. The registry is to be maintained as Specification Required per [RFC8126].

Initial registrations:

Member Name	Type	Description	Specification
res	String	URI or URN of the target resource or endpoint	Section 3.2 of this document
method	String	Protocol action string	Section 3.2 of this document
id	String	Client-generated correlation ID	Section 3.2 of this document

Table 6

12.12. SASL Mechanism Registration

This specification requests registration of the following entry in the "SASL Mechanisms" registry established by [RFC4422]:

Mechanism Name: OAUTHEPOP

Security Considerations: See Section 6.2.3 of this document.

Published Specification: This document.

Intended Usage: COMMON

Owner/Change Controller: IETF

13. References

13.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/rfc/rfc2104>.

[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>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/rfc/rfc3629>.

[RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <https://www.rfc-editor.org/rfc/rfc4422>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/rfc/rfc4648>.

[RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, DOI 10.17487/RFC5801, July 2010, <https://www.rfc-editor.org/rfc/rfc5801>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/rfc/rfc5869>.

[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>.

[RFC7009] Lodderstedt, T., Ed., Dronia, S., and M. Scurtescu, "OAuth 2.0 Token Revocation", RFC 7009, DOI 10.17487/RFC7009, August 2013, <https://www.rfc-editor.org/rfc/rfc7009>.

[RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/rfc/rfc7235>.

[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>.

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

[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>.

[RFC7628] Mills, W., Showalter, T., and H. Tschofenig, "A Set of Simple Authentication and Security Layer (SASL) Mechanisms for OAuth", RFC 7628, DOI 10.17487/RFC7628, August 2015, <https://www.rfc-editor.org/rfc/rfc7628>.

[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>.

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

[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>.

[RFC8693] Jones, M., Nadalin, A., Campbell, B., Ed., Bradley, J., and C. Mortimore, "OAuth 2.0 Token Exchange", RFC 8693, DOI 10.17487/RFC8693, January 2020, <https://www.rfc-editor.org/rfc/rfc8693>.

[RFC8725] Sheffer, Y., Hardt, D., and M. Jones, "JSON Web Token Best Current Practices", BCP 225, RFC 8725, DOI 10.17487/RFC8725, February 2020, <https://www.rfc-editor.org/rfc/rfc8725>.

[RFC9126] Lodderstedt, T., Campbell, B., Sakimura, N., Tonge, D., and F. Skokan, "OAuth 2.0 Pushed Authorization Requests", RFC 9126, DOI 10.17487/RFC9126, September 2021, <https://www.rfc-editor.org/rfc/rfc9126>.

[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>.

13.2. Informative References

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/rfc/rfc8126>.

[RFC8792] Watsen, K., Auerswald, E., Farrel, A., and Q. Wu, "Handling Long Lines in Content of Internet-Drafts and RFCs", RFC 8792, DOI 10.17487/RFC8792, June 2020, <https://www.rfc-editor.org/rfc/rfc8792>.

Acknowledgments

The author thanks the OAuth Working Group for their foundational work on DPoP ([RFC9449]), PKCE ([RFC7636]), and the related specifications that this document extends.

Author’s Address

Ashwin Ambekar
eBay
Email: ambekar@gmail.com, aambekar@ebay.com