Separating DPoP Bindings for Access and Refresh Tokens

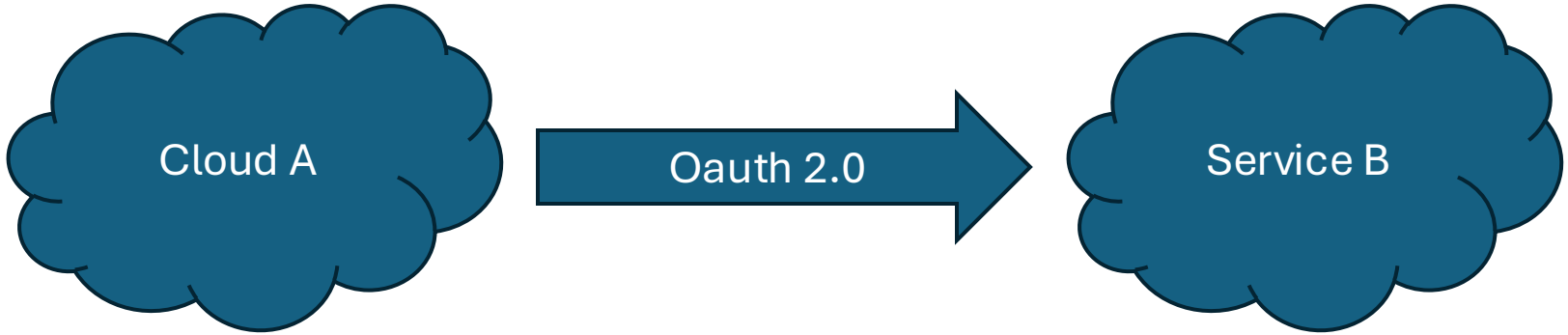
`draft-rosomakho-oauth-dpop-rt-00`

Loren Weith

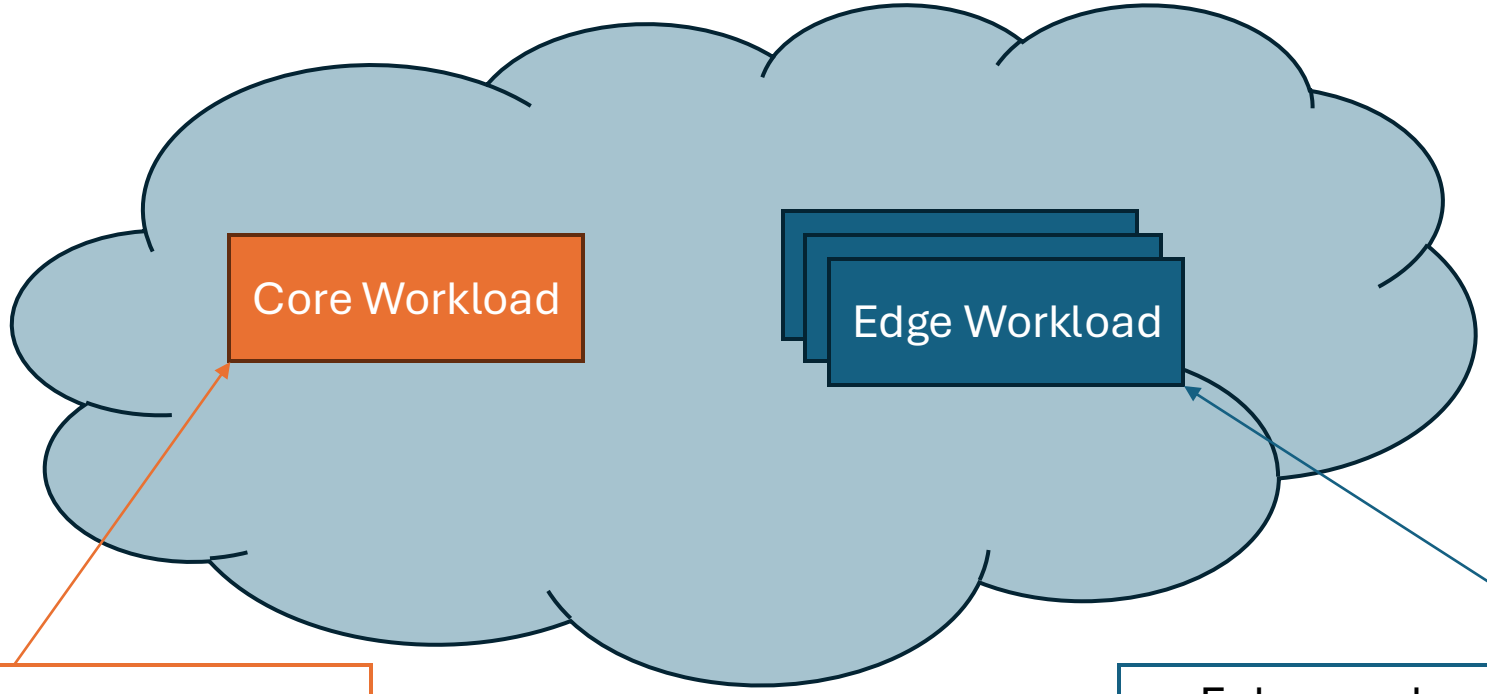
Yaroslav Rosomakho

Oauth
IETF124, November 2025, Montreal

The problem that we are trying to solve



The problem that we are trying to solve



- Long-living
- Write things to disk
- Often backed by an HSM/TPM

- Ephemeral
- Store things in memory
- Do not have access to hardware security

The problem that we are trying to solve

Fast Containment of Leaked Refresh Tokens from Cloud-Based Agents

- **Requirements**
  - Support for distributed architecture
  - Separate key access for RT (long lived in core) and AT (short lived at edge)
  - Cryptographically bound to service components, hard to use outside agent

- **Challenges**
  - Delegated tokens must be revoked by owner
  - Cert bound (RFC 8705) tokens seem reasonable
  - DPoP is simpler with fewer parties to trust

Simple Revocation

- Customers must each do this individually
  - Upon customer recognition, resolution is fast (for that customer)
- With many customers
  - Some will respond immediately
  - Some will respond eventually
  - Some will respond never

Available Solutions

- Confidential client options
  - mTLS with issuance from a likely private CA
    - Customer must have a PKI from which to issue a cert to the agent. Maybe it's a good one.
    - Upon key compromise, days to revoke key.
  - JWKS -> based on HTTPs -> authenticated with cert from public CA
    - Ultimately relies on a highly regulates public CA
    - Upon server cert key compromise, days to revoke key
    - Otherwise, could vary based on caching

- DPoP
    - Key specific to token, not changeable
    - Follows from RT to AT
      - Fine at low rate of signing
      - Access tokens used a lot (HSM / TPM could be bottleneck)
      - Refresh used less (HSM / TPM less of a bottleneck)
      - Limits scalability in distributed / high volume architecture when HSM involved

PKI Based Options: mTLS / JWKS

- mTLS
  - Root of Trust: Private CA (client certificate)
  - Level of Trust: Varies, hard to do well
  - Revocation: CRL, if reachable by the provider
  - Time To Disable: Days up to Never

- JWKS
  - Root of Trust: Web PKI (server cert of the PKIX server)
  - Level of Trust: High – highly regulated and audited operations
  - Revocation: CRL
  - Time to Disable: Around 3.5 days assuming 7-day CRL validity

The proposal

- Separate DPoP for access tokens and DPoP for refresh token
- Introduce a new `DPoP-RT` header with similar structure to regular DPoP
  - typ = “dpop-rt+jwt”
  - rth = hash of refresh token
- Optional `DPoP-RT-Nonce` mechanism similar to regular `DPoP-Nonce`

Overall flow

Client | Server

Authorization code exchange with DPoP and DPoP-RT →

← Access token bound to DPoP, refresh token bound to DPoP-RT

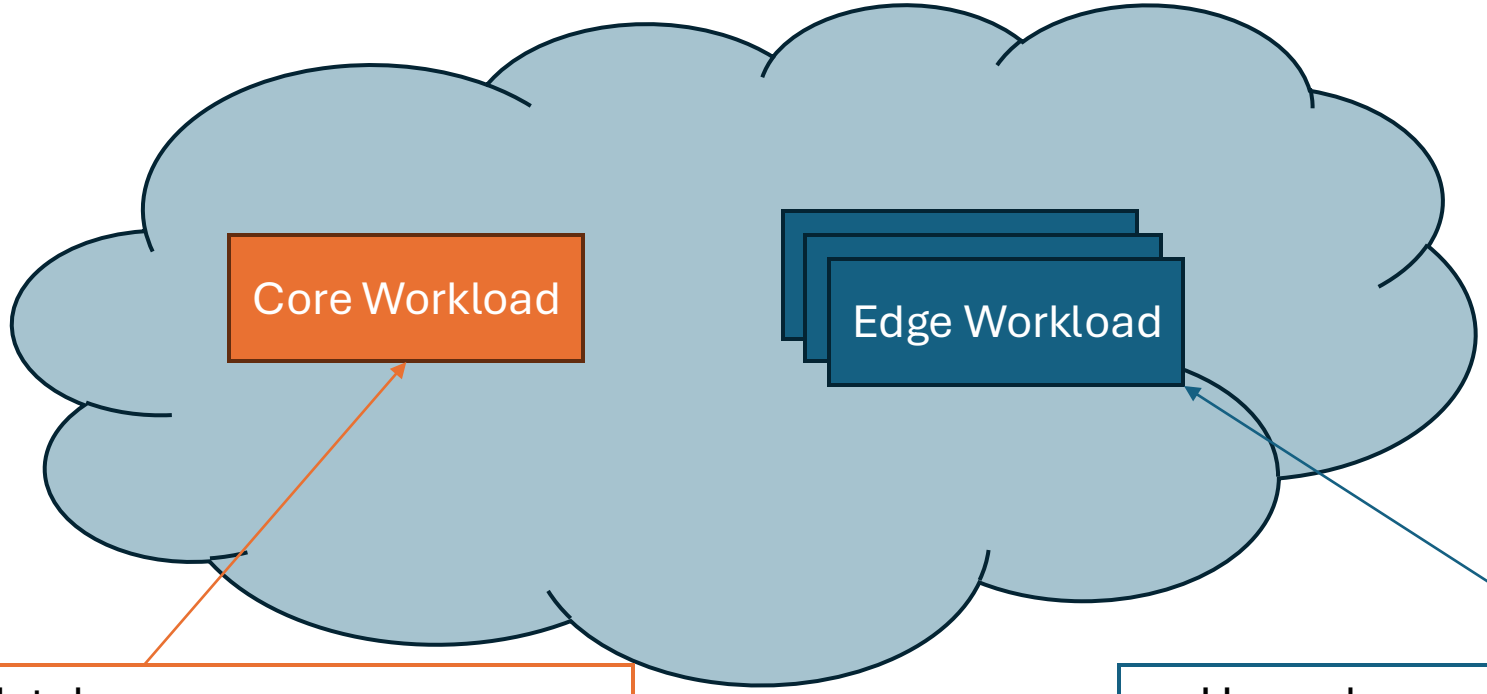
Access with access token and DPoP →
→
→

Refresh token request with **new** DPoP and established DPoP-RT →

← Access token bound to new DPoP

(Optionally) new refresh token bound to established DPoP-RT

The outcome



- Stores refresh token
- Keeps private key of DPoP-RT on HSM/TPM

- Has only access token
- Maintains DPoP
- Has not access to refresh token or DPoP-RT

Next steps

- Questions?
- Feedback?
- Suggestions?