Context Continuity for Web Authentication

Without Token or Identity Provider Modification

Strashko Sergey · plumbus.dev

April 2026 · Version 1.0

Assumed Knowledge

This paper is written for practitioners already familiar with the following standards and frameworks. Their core mechanics are not recapitulated here.

- RFC 6750 - OAuth 2.0 Bearer Token Usage
- RFC 9449 - OAuth 2.0 Demonstrating Proof of Possession (DPoP)
- RFC 9700 - OAuth 2.0 Security Best Current Practice
- RFC 8471 - The Token Binding Protocol Version 1.0
- RFC 8705 - OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens
- FAPI 2.0 Security Profile - Financial-grade API Security Profile (OpenID Foundation)
- OWASP Top 10 - particularly A07:2021 Identification and Authentication Failures

Executive Summary

Authentication and context continuity are two distinct security properties. Authentication answers: *who is this entity?* Context continuity answers: *is this request from the same execution context that established the session?* Authentication systems were designed primarily to answer the first question. Prior systems addressed the second indirectly, but none introduced it as a standalone architectural concern.

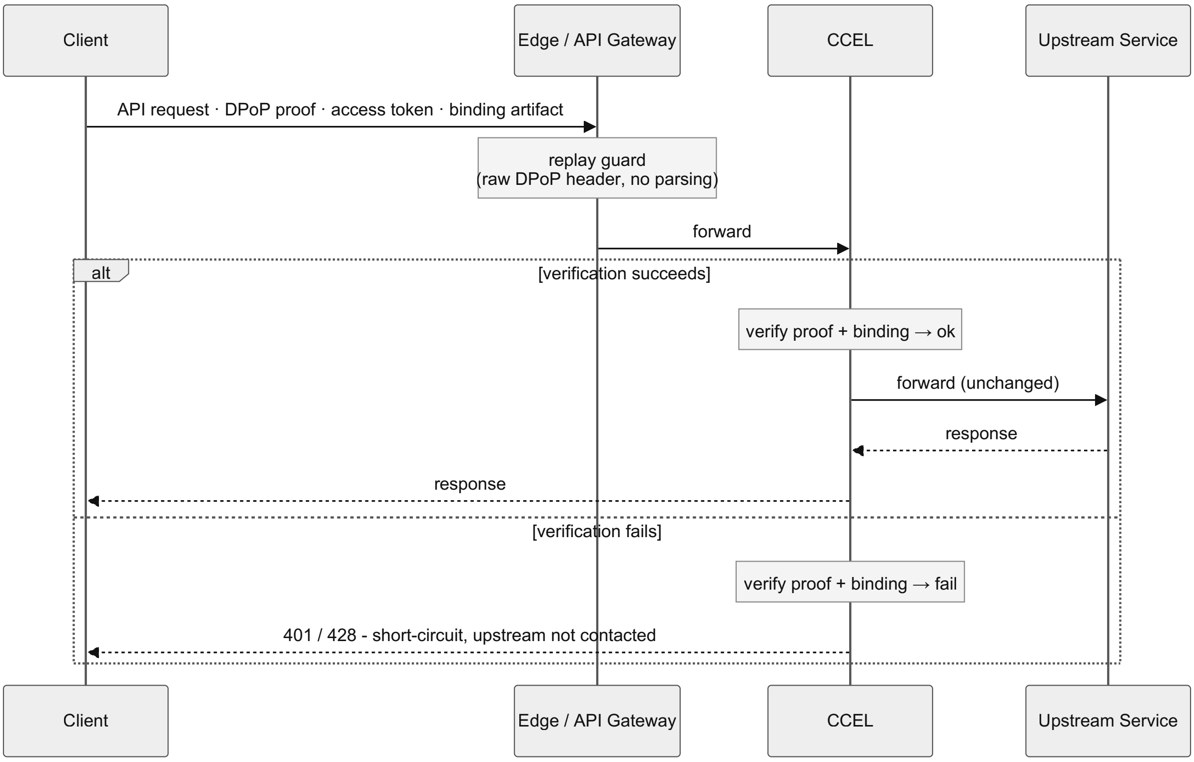
RFC 9449 is prior art that addresses context continuity via token-embedded key binding - a design choice that places binding creation at the identity provider and verification at every resource server. This is an IdP-centric approach: it extends what the identity provider builds and controls, and places the adoption chain on the relying party. Section 2 analyzes the structural consequences of this choice for each actor in the system.

This paper introduces a **context continuity enforcement layer (CCEL)**: an independent actor with sole responsibility for context continuity, deployed at the service boundary by the resource server operator - without changes to the identity provider, the token format, or resource-server application logic. CCEL is relying-party-centric: adoption, enforcement, and audit are owned entirely by the party that bears compliance obligation.

The difference between CCEL and RFC 9449 is not deployment difficulty. It is a fundamentally different allocation of responsibility: which actor owns context continuity, and whether that actor can act unilaterally.

Protocol at a Glance

The diagram below shows the most frequent operation under CCEL: a session-continuation API call. The client presents a fresh DPoP proof, the access token, and the binding artifact established at login. The edge / API gateway tier guards against proof replay using the raw `DPoP` header value as the dedup key. CCEL verifies the proof and the binding before forwarding to the upstream service. The IdP is not in the path; the upstream service is unmodified.



Section 5 develops the full protocol. Appendix B contains the precise sequence diagrams for client logic delivery, the MFA ceremony, and session continuation, along with the verification functions referenced throughout.

1. The Continuity Invariant

All analysis in this paper is measured against the following invariant:

> **The first request of an authentication ceremony - the request whose response produces authentication material - establishes the client context. The public key carried in that request's proof, verified at the enforcement layer, defines the context for the entire ceremony and all subsequent calls. Continuity requires that every subsequent request presenting any material from that ceremony - through intermediate challenge steps and all post-authentication API calls - originates from a party that can prove possession of the corresponding private key.**

This property is termed **context continuity**. It is distinct from authentication strength, token integrity, and transport security. No existing authentication system was designed to enforce context continuity as a standalone concern - because it has never been defined as one.

2. Actors, Responsibilities, and the Continuity Gap

Every web authentication system involves the same actors and a central artifact they exchange. Understanding their roles and the limits of their authority is the foundation for evaluating any approach to context continuity.

2.1 Actors and Artifacts

Identity Provider (IdP). The IdP owns identity: it authenticates the principal and issues authentication material. The IdP may be internal to the organization or an external vendor. It produces the authentication material (AM) that downstream systems rely on. Context continuity in downstream sessions is outside its defined scope.

Authentication Material (AM). Not an actor; an artifact. The AM is produced by the IdP at the conclusion of an authentication ceremony - an access token, a session cookie, or a proprietary credential. The AM is a contract between the IdP and the resource server: the IdP issues it, the RS accepts it. This contract is opaque to CCEL. Modifying or extending the AM - for example, to embed a `cnf` binding claim - requires both parties to the contract to participate. A change made by only one party is not effective.

Resource Server (RS). The RS is the target system. It is not a single server - it is a fleet: every node independently evaluates each incoming request. Under RFC 9449, the RS has an additional responsibility: extract the `cnf` claim from the AM and verify that the DPoP proof key matches. This must be done independently at every node in the fleet.

Client. In bearer authentication, the client is passive - it presents the AM and nothing more. Under both RFC 9449 and CCEL, the client must actively cooperate: it generates an ephemeral key pair per session and produces a cryptographic proof for each request. Under RFC 9449, client implementation is left undefined by the standard. Under CCEL, client implementation can be delivered by the enforcement layer itself - as a service worker or client-side library - consistently with the server-side configuration.

Context Continuity Enforcement Layer (CCEL). CCEL is a new independent actor. Its sole responsibility is to enforce context continuity: to forward only requests that satisfy the continuity invariant. CCEL is owned and operated by the RS operator - not the IdP, not a vendor. It is deployed at the service boundary and is fully configuration-driven. CCEL is not a party to the AM contract; it does not parse or modify the AM. It creates and verifies its own binding artifact, external to the AM, observing both the client key and the AM at session establishment without any participation from the IdP.

2.2 Per-Actor Responsibility Matrix

The following matrix shows what each actor must do to enforce context continuity under RFC 9449 versus under CCEL:

Actor	RFC 9449 requirement	CCEL requirement
IdP	Token format migration + validate proof at issuance + embed `cnf`	No changes
AM	Must carry `cnf` binding - IdP–RS contract must change	No changes
RS (every one, fleet-wide)	Extract `cnf`, verify proof key matches, reject if not - independently, at every node	No changes
Client	Must cooperate - implementation left undefined by standard	Must cooperate - implementation deliverable by CCEL
CCEL	Does not exist	Deploy + configure routes - by RS operator, unilaterally

> **Note on freshness and duplicate-proof suppression:** Proof freshness - that a proof carries a recent `iat` within an allowed skew window - is stateless (see Appendix A) and can be enforced independently by any single instance of RS or CCEL. Stronger uniqueness-based duplicate-proof suppression is a separate concern: it requires coordinated shared state across all fleet instances and is appropriately implemented at the gateway or edge layer that already operates such infrastructure. CCEL's contribution is freshness enforcement; uniqueness-based dedup remains a platform concern.
> **Why `jti` parsing is unnecessary.** The per-request DPoP proof itself is unique, fresh, and bound to method and path - it satisfies the properties of a nonce. The raw serialized proof is directly usable as the opaque dedup key by any platform-layer mechanism the operator chooses to deploy. Parsing the proof to extract `jti` is unnecessary for this purpose.

2.3 RFC 9449's Architectural Choice and Its Consequences

RFC 9449 places key binding inside the authentication material - the `cnf` claim. This is an architectural choice, not a necessity. Placing binding inside the AM has specific, named consequences for each actor.

Consequence 1: Implementation authority resides with the wrong party. For a proprietary or in-house identity system, extending it to support `cnf` binding may be architecturally difficult or require changes the RS operator cannot make unilaterally. For a commercial IdP, the cryptographic strength of the binding key - its generation, rotation policy, and protection - is the vendor's decision, not the relying party's. Yet the compliance obligation sits with the relying party. The relying party holds the compliance obligation for a system whose cryptographic parameters it does not control.

Consequence 2: AM extension requires a coordinated contract change. The AM is a contract between the IdP and the RS. A `cnf` claim that no RS evaluates provides no security; a `cnf` claim that the IdP does not produce cannot be evaluated. Both sides of the contract must change simultaneously. For organizations with third-party IdPs or heterogeneous RS fleets, this is a coordinated ecosystem migration, not a deployment decision.

Consequence 3: Adoption is not auditable. Enforcement under RFC 9449 is distributed: each RS node independently verifies the `cnf` claim. There is no single source of truth for whether the fleet is fully covered. There is no artifact that proves fleet-wide enforcement. Audit requires inspecting every RS individually. A single RS that does not enforce the `cnf` check accepts tokens as plain bearer credentials - silently, without any visible signal.

Consequence 4: Compliance obligation and implementation authority are misaligned. The relying party is the actor that bears audit findings, regulatory pressure, and incident risk. Yet under RFC 9449, implementation authority resides with the identity provider: the IdP must validate proofs at issuance, embed `cnf`, and potentially migrate token formats. For organizations using commercial IdPs, the relying party cannot act unilaterally - adoption waits on a vendor roadmap. The party accountable for compliance cannot act on its own timeline - it must wait for a vendor to move first.

Consequence 5: Freshness enforcement is distributed across RS teams. Under RFC 9449, every RS team independently owns the configuration of the `iat` skew window and is responsible for enforcing proof freshness on every covered request. Under CCEL, proof freshness is enforced once at the continuity boundary, configured and operated in one place. The platform-layer treatment of stronger uniqueness-based duplicate suppression is discussed in the freshness note in §2.2.

Together, these consequences explain why RFC 9449 adoption is structurally constrained for the relying party - and why an alternative allocation of responsibility is worth defining.

3. Prior Art and Related Work

Prior approaches to sender-constrained authentication - Token Binding (RFC 8471), Mutual TLS (RFC 8705), FAPI 2.0, GNAP, and DPoP (RFC 9449) - share a common architectural assumption: that key binding must be expressed either inside the token or at the transport layer. In every case, enforcing the binding requires participation from at least one actor that may not be under the relying party's control: the identity provider (to embed the binding), the resource server fleet (to verify it), or the transport layer (to carry it).

Token Binding embedded key material in the TLS layer - requiring TLS termination points and origin servers to cooperate, with no path for unilateral adoption by the relying party. Mutual TLS requires client certificate infrastructure and RS-level verification at every endpoint. FAPI 2.0 extends DPoP and mTLS requirements to the full authorization flow - a high bar suited to regulated environments with organizational control over the entire token ecosystem. GNAP redesigns the authorization protocol to carry binding natively, requiring adoption of the new protocol by all parties.

The common architectural assumption across all prior approaches: the relying party cannot act unilaterally. Adoption requires either a standards process that moves the IdP, or RS-level changes that span the fleet.

CCEL breaks this assumption. To our knowledge, no prior published work - peer-reviewed paper, IETF draft, or RFC - presents an explicit formulation in which the RS operator, the party accountable for compliance, can adopt context continuity enforcement entirely within their own operational boundary, without any participation from the IdP or changes to the RS fleet.

4. The Missing Layer: Context Continuity

This paper introduces the context continuity layer as a distinct architectural component in web security systems.

4.1 Definition

Context continuity is the security property of an authentication ceremony and its resulting session such that every request - from the first exchange that produces authentication material through all intermediate challenge steps and all subsequent API calls - can be cryptographically verified to originate from the same client context.

“Client context” is defined precisely: the execution environment that holds the private key whose corresponding public key was verified at the moment authentication material was first produced. Continuity is not about device identity, user identity, or network location. It is about key continuity across the entire ceremony and session lifetime. The same key that was verified at first material issuance must prove possession of the key at every subsequent request.

This definition excludes IP binding, device fingerprinting, and behavioral heuristics by construction. Those mechanisms approximate continuity through observable properties that may or may not correlate with context integrity. Context continuity, as defined here, requires cryptographic proof.

4.2 Required Properties

A system that enforces context continuity must satisfy the following invariants:

1. **Binding at establishment.** At session establishment (the authentication event), the enforcement system records a cryptographic relationship between the authentication material and the client’s public key.

2. **Proof on every request.** Every subsequent request in the session must include a fresh cryptographic proof that the requester holds the private key corresponding to the registered public key.

3. **Binding verification.** The enforcement system must verify, on every request, that the public key used in the proof matches the public key registered at session establishment for the presented authentication material.
4. **Proof freshness.** Proofs must carry a recent timestamp within an enforced skew window. The continuity layer enforces freshness through this stateless time bound.
5. **No reliance on transport properties.** The enforcement must not derive its security guarantees from IP addresses, TLS session state, device fingerprints, or other observable-but-forgeable context properties. The guarantee must be derivable from the cryptographic material alone.

4.3 Break Conditions

Context continuity is violated in the following conditions:

Authentication ceremony splicing. An attacker intercepts intermediate authentication material produced during a multi-step ceremony - a challenge state, an MFA session token, or a partial authentication artifact - and attempts to complete the ceremony from a different client context. Because the enforcement layer bound the context to the client public key at the moment that material was first produced, any continuation request carrying a different key fails verification.

Token replay from a different key. An attacker obtains authentication material (a bearer token, session cookie, or other credential) but not the client's private key. Requests constructed with this material but a different key - or no key - fail binding verification. The session is not accessible to the attacker.

Context cloning. An attacker obtains both the authentication material and the binding artifact (the signed artifact recording the established session binding) but cannot produce valid proofs without the private key. Cloning the binding artifact without the key is insufficient.

Proof forgery. An attacker attempts to construct a proof without the private key. Given a sufficiently strong signature scheme, this is computationally infeasible.

Note on scope. Context continuity does not prevent an attacker who has compromised the original client device or execution environment and extracted the private key. In that scenario, the attacker can produce valid proofs indistinguishable from the legitimate client's proofs. This is not a limitation specific to this model - RFC 9449 accepts the same boundary. Key extraction from a compromised client cannot be addressed at the protocol level; it is a property of the underlying cryptographic assumption.

5. Model: Externalized Sender Constraint

The central insight of this model is that **sender constraint does not need to be implemented inside the token**. The token is opaque to the enforcement layer. The enforcement layer creates and verifies its own binding artifact, independently of the token's internal structure. The token's validity is still verified by the upstream service. The context continuity property is enforced by the enforcement layer, at the service boundary, without either party needing to know about the other.

5.1 Key Insight

RFC 9449 places the binding inside the authentication material - the `cnf` claim - because the AM contract is the natural extension point when the IdP is the primary actor. CCEL takes a different approach: the binding artifact is external to the AM contract by design. CCEL is not a party to the AM contract; it does not modify or inspect the AM's internal structure. It creates its own binding artifact, observed at the service boundary, owned by the enforcement layer alone.

This is not an alternative placement for the same idea - it is a different actor taking ownership of a different concern.

This placement is possible because the enforcement layer has access to both sides of the binding at the moment they occur: the client's key (present in the authentication request as a proof) and the authentication material (present in the authentication response). No other party needs to observe or record this relationship.

5.2 Components

The model has three principal artifacts:

Authentication material. An artifact issued by the upstream service during or at the conclusion of an authentication ceremony. This includes intermediate MFA challenge states, partial authentication tokens, and step-up session identifiers, as well as final credentials such as JWT access tokens, opaque session cookies, or proprietary session identifiers. The enforcement layer treats each piece of authentication material as opaque - it does not parse or validate its internal structure. It computes a fixed-size cryptographic hash of the authentication material for use in binding. The content of authentication material is opaque to the enforcement layer, but the layer requires configuration identifying where authentication material appears in the authentication response and how it is presented on subsequent requests, so that the value hashed at binding time matches the value hashed at verification time.

A distinct binding artifact is created for each piece of authentication material the enforcement layer observes. Systems that issue multiple authentication materials receive a corresponding binding for each.

Client key. An ephemeral asymmetric key pair generated by the client. The private key never leaves the client's execution environment. The public key is conveyed with each proof via the JOSE header of the proof JWT. The enforcement layer extracts the public key from the proof during the authentication request and computes a fixed-size cryptographic hash of it for use in binding.

Binding artifact. A signed artifact created by the enforcement layer at session establishment time. It contains the hash of the authentication material and the hash of the client's public key, signed by the enforcement layer's signing key. This artifact binds the authentication material to the client's key in a tamper-evident form. It is delivered to the client (typically as an HttpOnly cookie) and must be presented on all subsequent requests.

5.3 Enforcement Point

The optimal placement for the context continuity enforcement layer is at the network ingress boundary - a WAF, TLS termination point, or API gateway - where all traffic already converges and enforcement requires no additional network hop. For deployments without a centralized ingress, the enforcement layer can equally be positioned as a sidecar or dedicated reverse proxy in front of the upstream service, operating at the individual service boundary. In all cases the logical role is identical: intercept requests and responses on paths designated for continuity enforcement.

The enforcement layer has two operational modes corresponding to two route types:

Authentication routes (context establishment). When a client authenticates, the enforcement layer requires a DPoP-style proof in the request. It calls `DPoPVerify` (Appendix B.1) to validate the proof and extract the client's public key, forwards the request to the upstream service, observes the authentication material in the response, creates a binding artifact, and attaches it to the response before delivery to the client. The upstream service is unmodified; it sees the authentication request and returns the credential as usual.

API routes (context continuation). On all subsequent requests, the enforcement layer requires the client to present both a fresh proof (signed with the same key) and the binding artifact. It calls `BindingVerify` (Appendix B.1). If verification succeeds, the request is forwarded to the upstream service. If any check fails, the request is rejected with an appropriate error response, without being forwarded.

This placement requires no changes to the upstream service or application business logic - the only client-side addition is proof generation, which can be delivered transparently via a service worker or client-side library without modifications to application code.

5.4 Protocol Flow

The following describes the protocol flow at a level of abstraction independent of any specific implementation:

Session establishment:

1. The client generates an ephemeral asymmetric key pair `(pk, sk)`. The private key `sk` is retained by the client and not transmitted.
2. The client constructs an authentication request and attaches a DPoP-style proof: a signed JWT containing the public key `pk`, the request method and URI (`htm`, `htu`), a timestamp (`iat`), and a unique identifier (`jti`). The proof is signed with `sk`.
3. The enforcement layer receives the request and calls `DPoPVerify(proof, request)` (Appendix B.1) to validate the proof and extract `pk`.
4. The enforcement layer forwards the request to the upstream service.
5. The upstream service authenticates the client and returns authentication material `A` in the response.
6. The enforcement layer observes `A`. It computes `H(A)` and `H(pk)`. It creates a binding artifact `B` consisting of the payload `H(A) || H(pk)` together with a signature over that payload produced using its own signing key. The full artifact - payload plus signature - is delivered to the client and verified on subsequent requests.
7. The enforcement layer returns the response to the client with `B` attached as an HttpOnly cookie.

Session continuation:

1. The client prepares a request to an API route. It attaches: the authentication material `A` (as a bearer credential or equivalent), the binding artifact `B`, and a fresh proof signed with `sk` (new `jti`, current `iat`, matching `htu` and `htm`).
2. The enforcement layer calls `BindingVerify(B, A, proof, request)` (defined in Appendix B.1).
3. If verification succeeds, the request is forwarded to the upstream service. If any check fails, the request is rejected.

See Appendix B.2–B.4 for sequence diagrams covering client-logic delivery, MFA establishment, and session continuation. See Appendix D for example HTTP request and response structures.

6. Implementation Requirements for CCEL

A conformant CCEL must provide the following capabilities. This section is prescriptive: it defines what any implementation must support to be considered a context continuity enforcement layer under this model.

6.1 Route Policy Configuration

The operator must be able to declare, per route (method + URI pattern):

- **Authentication route** - proof required.
  - The presented proof is verified via `DPoPVerify` (Appendix B.1) before forwarding.
  - Always: the binding artifact is created from the upstream response and attached before delivery to the client.
  - If a prior ceremony step is declared as a prerequisite: the bound authentication material from that step (AM + proof + binding artifact) is required and verified via `BindingVerify` before forwarding.
  - A token refresh endpoint is an authentication route: it follows the same AM lifecycle as initial issuance, with the same actor analysis. No special case is required.
- **API route** - proof and binding artifact required.
  - The presented proof and binding artifact are verified via `BindingVerify` (Appendix B.1) before forwarding.
  - Verification failure short-circuits the request per §6.5.

Proof validity and freshness enforcement apply to every route in scope - authentication routes and API routes alike. Continuity verification (binding artifact check) applies to every route for which a binding context exists: API routes unconditionally, and authentication routes where a prior ceremony step is declared as a prerequisite.

6.2 Authentication Material Location

For each authentication route, the operator must be able to configure two independent concerns:

AM location in the upstream response (for binding). The operator must be able to specify:

- Which upstream response code(s) indicate that AM was actually produced and should be bound

- Where in the response the AM is located - header, cookie, body

AM location in subsequent requests (for verification). The operator must be able to specify:

- Where the AM is presented in API route requests - header, cookie, query parameter, body

6.3 Cryptographic Policy

CCEL defines the cryptographic policy for the full enforcement boundary. Consistency between proof generation and proof validation is a CCEL responsibility - the operator must be able to configure:

Proof generation and validation:

- The algorithm used for client proof generation - the same algorithm is applied for validation, ensuring consistency across client and server
- Clock skew tolerance (`iat` window)
- Proof freshness: proofs must carry a fresh `iat` within the configured skew window. This stateless time-window check is the CCEL baseline and is sufficient for the core continuity property.

Binding artifact signing:

- The algorithm and key type used to sign binding artifacts
- The signing key source (see §6.4)

> **Uniqueness-based deduplication is out of scope.** Suppressing reuse of the same proof within its validity window is not a CCEL concern. By requiring a DPoP proof on every covered request, CCEL adoption produces a per-request unique, fresh, method- and path-bound token that already satisfies the properties of a nonce. WAF and edge infrastructure can consume the raw proof directly as the opaque dedup key - no parsing, no field extraction, no additional CCEL mechanism or shared state required.

6.4 Key Management

The operator must be able to configure a secure key store. CCEL must be able to load signing keys from the configured store. Key rotation must be supported with overlap: the implementation must accept artifacts signed by either the previous or current key during a configurable rotation window. In multi-instance deployments, all instances must be able to verify artifacts produced by any peer instance. The signing key is the trust anchor for all binding artifacts (see §8.3); its protection is the operator's responsibility.

6.5 Failure Behavior

The response returned on context continuity failure must be configurable by the operator. At minimum the operator must be able to control: the HTTP status code returned per failure class (proof verification failure vs. missing or invalid binding artifact), the response body and content type, and any additional response headers required by the application's error contract. The default behavior should distinguish cryptographic verification failures from missing-prerequisite failures so that clients and observability tooling can react appropriately - typically `401 Unauthorized` for the former and `428 Precondition Required` for the latter.

6.6 Auditability and Telemetry

The implementation should expose:

- The complete set of enforced routes (coverage declaration - verifiable without inspecting application code)
- Rejection counts by failure reason: proof invalid, binding artifact invalid, AM mismatch, key mismatch
- Proof freshness violations as a separate metric or event stream, so dashboards and alerting can target them independently of other rejection reasons

These signals are the operational foundation of the governance guarantees described in §11: coverage is declared in one place; violations are observable from one source.

6.7 Client-Side Delivery

A conformant CCEL must be capable of delivering the client-side proof generation component, parameterized from the server-side configuration. This component must:

- Generate an ephemeral asymmetric key pair scoped to the session, using the algorithm declared in §6.3
- Store the private key in a non-extractable form, scoped to the origin
- Construct a proof JWT for each covered request containing the required `htm`, `htu`, `iat`, and `jti` fields, signed with the session private key
- Present the proof in the location required by the configured route policy

A *session* is defined here as the lifetime of the final binding artifact established at the conclusion of the authentication ceremony: the key pair is generated before the first authentication request, reused across any intermediate ceremony bindings, and discarded only when the session ends (logout, expiry, or explicit termination). The `jti` field is included for RFC 9449 compliance; CCEL itself does not validate it. Where an operator deploys platform-layer duplicate-proof suppression, the dedup key is the raw serialized proof - no `jti` extraction required.

The component may be delivered as a service worker (intercepting covered fetch calls transparently, without application code changes) or as a client-side library (integrated into the application's request path). In both cases the component must be generated or parameterized from the server-side configuration - not hardcoded - so that changes to cryptographic policy are reflected in the delivered client component without requiring separate client updates.

This property - that the client-side component is derived from the server-side configuration - is what makes the algorithm consistency guarantee in §6.3 a CCEL responsibility rather than an application integration concern.

7. Security Properties

7.1 Security Guarantees

The proof validation guarantees of RFC 9449 relevant to context continuity apply unchanged: proof signature validity, `htm`/`htu` binding to the specific request, and `iat` freshness enforcement all hold under the same assumptions as RFC 9449 - specifically, that the client's private key is not extractable by the attacker and that the client generates a fresh proof per request. No new cryptographic properties are introduced; the enforcement layer applies the same validation rules RFC 9449 defines for resource servers.

The binding artifact extends these guarantees in two ways that RFC 9449 does not provide without IdP participation. First, **authentication material and key continuity**: the enforcement layer verifies on every request that the authentication material and the public key match those recorded at context establishment - without parsing the token's internal structure, and without the identity provider embedding a `cnf` claim. Second, **token-format independence**: because the enforcement layer hashes the authentication material rather than inspecting it, these guarantees hold for JWT access tokens, opaque session cookies, session identifiers, and proprietary credentials alike. RFC 9449 commonly relies on token representations that expose binding semantics directly (for example, JWT access tokens), while this model does not depend on token format.

7.2 Non-Goals

Consistent with the threat model adopted in RFC 9449, the following are out of scope: physical device compromise, full client-side code execution by an attacker, identity fraud, and availability attacks. These are conditions outside the protocol layer's scope - either the underlying cryptographic assumption fails (physical device compromise or full client-side code execution by an attacker, both leading to key extraction), or the threat is orthogonal to the proof model (identity fraud, denial of service).

7.3 Comparison to IdP-level Binding

This model achieves the same continuity outcome for protected requests as IdP-level DPoP (RFC 9449, Section 5) - only requests from the client context that established the session are accepted - under the same cryptographic assumptions, and subject to the trust requirement on the enforcement layer's signing key described in Section 8.3. Both models use DPoP proofs; the difference is architectural, not cryptographic. In IdP-level binding, the `cnf` claim is embedded in the token by the IdP and verified by every RS independently - binding creation and enforcement are distributed across actors and teams. In CCEL, the binding artifact is created and verified by the enforcement layer alone - no IdP participation, no RS-level enforcement logic. The architectural difference is not weaker cryptography or reduced guarantees - it is a different allocation of which actor owns the property, with CCEL placing ownership where compliance obligation already sits. Freshness enforcement follows the same allocation: under RFC 9449 it must be implemented wherever proofs are verified - distributed across every RS team - while under CCEL it is enforced once at the continuity boundary.

8. Threat Model

8.1 Attacker Capabilities

The attacker capability assumptions of RFC 9449 apply: token theft by any means, network observation including TLS inspection environments, arbitrary request construction, and proof replay attempts. In all such scenarios the attacker possesses the authentication material but not the client private key. Replay outside the freshness window is rejected by the time-bound check; replay within the window targeting non-idempotent operations is a platform/application concern (see the freshness note in §2.2). This model adds one capability: **binding artifact theft** - the attacker may also obtain `B`, though it is typically protected as an HttpOnly cookie inaccessible to JavaScript.

8.2 What Is Prevented

The proof-level protections of RFC 9449 apply unchanged. Additionally, this model prevents two attack classes that RFC 9449 cannot address without full IdP-level adoption:

Authentication ceremony splicing. An attacker who intercepts intermediate authentication material - a challenge state, an MFA token, or any artifact produced before the final credential is issued - cannot complete the ceremony from a different client context. The enforcement layer bound the context to the client public key at the moment that intermediate material was first produced; continuation requests from any other key are rejected before reaching the upstream service.

Cross-context token replay. An attacker who obtains `A` and `B` but not `sk` cannot construct a valid proof. The enforcement layer requires a fresh proof matching the key registered in the binding artifact on every request; without `sk`, the session is inaccessible from any other context.

8.3 Trust Boundary

Enforcement layer signing key. The enforcement layer is the trusted party in this model. It creates binding artifacts signed with its own key; the security of those artifacts depends on the enforcement layer's signing key not being compromised. Key management, rotation, and protection for the enforcement layer's signing key are operational concerns that should follow standard practices for high-value signing keys: hardware security modules or managed key services, regular rotation, and audited access. Deployments should account for this key as an additional trust anchor: an additional high-value signing key, alongside any IdP-side keys the organization already operates or relies on, requiring equivalent protection.

Client-side code integrity. The security of the model requires that the client-side proof generation component executes without tampering - whether that component was delivered by CCEL or implemented by the application directly. A modified, injected, or replaced client component can suppress proof generation, produce invalid proofs, or expose the private key. This is not a concern specific to CCEL-delivered code; it applies equally to any client-side code regardless of origin. This sits at the same boundary as private key extraction from a compromised device - outside what the protocol layer can guarantee.

9. Deployment Model

9.1 Integration Properties

The actor model explains why each system component requires no changes. CCEL is defined as not being a party to the AM contract - so the IdP's relationship with the RS is unaffected by CCEL's presence. CCEL treats the AM as opaque - so no token format is presumed or required. CCEL sits at the network ingress before the upstream service - so the upstream service sees only requests that have already passed enforcement and remains entirely unmodified. The client must cooperate, but CCEL can deliver the client-side component itself, consistently with the server-side configuration.

Each actor's unchanged role is a direct consequence of CCEL's definition:

Identity provider (IdP). No changes. The enforcement layer forwards authentication requests to the IdP and observes the response. The IdP issues tokens as it always has.

Token format. No changes. The enforcement layer does not parse tokens. It hashes whatever the upstream service returns. JWTs, opaque tokens, session cookies, and proprietary credentials are all handled identically.

Client application logic. No changes to business logic. The only client-side addition is proof generation: a function that creates a signed proof JWT for each request. This can be delivered as a service worker (intercepting all fetch calls from the application) or as a small library injected into the application's request path. Application code does not need to be aware that proofs are being generated.

Upstream service. No changes. The enforcement layer enforces context continuity before forwarding the request. The upstream service sees only requests that have already passed enforcement and responds as normal.

Because the enforcement layer is an independent entity with no coupling to the upstream service, coverage can be applied incrementally - beginning with high-value routes and expanding over time without disruption to existing sessions or clients.

9.2 Operational Impact

The per-request overhead of context continuity enforcement consists of:

- One asymmetric signature verification (proof signature, using the public key extracted from the proof)
- Two cryptographic hash computations (presented authentication material; presented public key)

- One signature verification (binding artifact)
- One additional network hop to the upstream service (eliminated when CCEL is deployed as a co-located sidecar or in-process gateway component)

These operations are computationally lightweight. For latency-sensitive applications, the additional hop can be addressed by co-location (sidecar model) or by selecting enforcement scope carefully. For most web applications, the overhead does not require architectural accommodation.

10. Economic Impact

10.1 Governance Consequences of IdP-Centric Enforcement

The architectural consequences identified in §2.3 manifest at the governance level as the three operational outcomes below. Placing context continuity enforcement inside the AM contract creates these outcomes regardless of whether an organization adopts RFC 9449. They are not design tradeoffs - they follow directly from placing enforcement inside the IdP-controlled AM contract.

Vendor lock-in. When context continuity is a function of the identity provider, it becomes entangled with the IdP as a feature. Replacing the IdP - for performance, cost, compliance, or architectural reasons - is now a continuity decision. Two orthogonal concerns (identity and continuity) are coupled at the vendor boundary. An organization that switches IdPs must account for continuity disruption as part of that migration.

Unauditable adoption. Enforcement distributed across a fleet of RS nodes has no single source of truth. An organization under regulatory pressure cannot produce a single artifact that demonstrates fleet-wide coverage. Compliance requires inspecting every RS individually - a continuous operational burden that grows with the fleet and produces no durable audit trail. A single uncovered RS silently accepts tokens as plain bearer credentials.

Regulatory misalignment. The party accountable for compliance is the relying party. The party that controls implementation under RFC 9449 is the identity provider. For organizations using commercial IdPs, compliance timelines are subject to vendor roadmaps and contractual relationships outside the relying party's authority. The organization that bears the risk does not control the remediation.

10.2 Deployability as Economic Value

This model resolves all three governance consequences. The enforcement layer requires no identity provider changes, no token format migration, no application rewrite, and no coordination across resource servers. The integration surface is any request-intercepting component - a reverse proxy, API gateway, WAF with edge-compute extension, middleware, or serverless function - which most organizations already operate in some form. Adding context continuity enforcement is an incremental configuration change, not an infrastructure project. The security property is available immediately, without waiting for ecosystem alignment.

For organizations under regulatory or compliance pressure, this distinction is particularly material. The relying party is accountable for compliance; an externalized enforcement layer means the relying party satisfies the requirement within its own operational boundary, on its own timeline, without waiting for vendor adoption.

Externalizing context continuity into a dedicated layer also separates operational responsibilities: security teams own and configure the enforcement boundary independently of application development. When enforcement is distributed across individual resource servers - as RFC 9449 requires - compliance becomes unverifiable: it is practically difficult to determine which endpoints enforce key binding and which do not, and a single uncovered server silently creates an exposure risk. The enforcement layer makes coverage explicit: a single configuration declares exactly which routes are subject to enforcement, auditable and verifiable without inspecting application code.

11. Strategic Implications

11.1 Separation of Concerns

The foundational premise of this paper is that identity and context continuity are distinct security properties. Authentication systems answer *who is this entity?*; the continuity enforcement layer answers *is this request from the same context that established the session?* With CCEL, these concerns are cleanly separated: the identity provider handles identity; the continuity enforcement layer handles context continuity. Each is owned by the right actor, operated on its own lifecycle, and replaceable without disrupting the other.

11.2 Compatibility with Existing Systems

Context continuity enforcement is compatible with the full range of authentication patterns in common use:

OAuth 2.0 / OIDC. Bearer access tokens issued by any identity provider can be bound by the enforcement layer. The token format is irrelevant.

Session cookies. Cookie-based sessions - as issued by traditional server-rendered applications - are handled identically. The session identifier (or session cookie value) is hashed and bound to the client key. No session management changes are required.

Proprietary authentication. Applications that use proprietary authentication formats - including legacy systems where authentication material is embedded in request parameters or custom headers - can be covered by the enforcement layer if the enforcement layer can observe the authentication material in the response and its presentation in subsequent requests.

11.3 Toward a Browser-Native Capability

The client-side component of this model - ephemeral key pair generation, proof construction, and automatic proof attachment to covered requests - is currently implemented in application logic or a service worker. This is functional and deployable, but it requires per-application adoption.

A natural extension of this model is browser-native support: a browser API, analogous to the WebCrypto API, that manages ephemeral key pairs for authenticated sessions and automatically attaches proofs to covered requests based on site configuration. This would move proof generation out of application code into the browser platform, reducing per-application integration overhead and enabling continuity enforcement for any web application without code changes. The server-side enforcement model described in this paper is already compatible with such a future: it processes proof headers mechanically, independent of how those headers were generated. Provided a browser primitive emits proofs in the DPoP wire format defined by RFC 9449, no enforcement-layer change would be required and no migration discontinuity would arise.

12. Conclusion

Bearer authentication cannot provide context continuity. This is not an implementation deficiency - it is a property of the bearer model. Any party that possesses a bearer token can use it.

Prior approaches to sender-constrained authentication - mutual TLS, DPoP, Token Binding - address the problem with sound cryptography. Each places the binding where its primary actors have control: inside the token, or at the transport layer. In each case, the relying party inherits the adoption chain. Compliance obligation and implementation authority are misaligned by design.

CCEL inverts this allocation. Context continuity is owned by the enforcement layer, deployed unilaterally by the RS operator within their own operational boundary. Adoption requires no IdP changes, no RS fleet changes, and no coordination with external parties. Audit is a single configuration, not a fleet inspection.

What CCEL introduces

- **Architectural clarity.** Context continuity is defined as an independent security property with a dedicated enforcement point, not a side effect of token design.
- **Relying-party control.** Adoption and policy decisions are executed by the operator that bears compliance responsibility, without IdP coordination.
- **Token-format independence.** Enforcement operates on opaque authentication material; JWTs, cookies, and proprietary session tokens are treated identically.
- **Centralized enforcement.** Continuity policy is applied once at the service boundary rather than replicated across every resource server.
- **Stateless baseline model.** Freshness is enforced structurally, without mandatory shared replay state.
- **Auditable coverage.** Protected routes are declared in a single configuration, making the enforcement boundary inspectable.
- **Incremental deployability.** Coverage can begin with the highest-risk routes and expand without disrupting existing infrastructure.

What CCEL removes

- Mandatory IdP participation in continuity adoption
- Token schema changes or credential format migration
- Per-service continuity implementations across the resource server fleet
- Fleet-wide inspection to verify continuity coverage
- Dependence on external vendor roadmaps for continuity rollout

The question is not whether the cryptography is sound. It is who owns the security property - and whether that owner can act.

Appendix A: Terminology

Bearer token. A security token whose possession is sufficient proof of authorization. Any party holding the token can use it. Defined in RFC 6750.

Authentication ceremony. The sequence of one or more request/response exchanges through which a client establishes an authenticated session. May include credential submission, MFA challenges, passkey flows, and other intermediate steps before final authentication material is issued.

Authentication material (AM). An artifact issued by the upstream service during or at the conclusion of an authentication ceremony. Includes intermediate artifacts such as MFA challenge states and step-up session identifiers, as well as final credentials such as JWT access tokens, opaque session cookies, and proprietary session identifiers. Treated as opaque by the enforcement layer.

Client context. The execution environment that holds the private key whose corresponding public key was verified at the moment authentication material was first produced in a ceremony. Continuity is defined as key continuity: the same key must prove possession at every subsequent request within the ceremony and session.

Context continuity. The security property of an authentication ceremony and its resulting session such that every request - from the first exchange that produces authentication material through all intermediate challenge steps and all subsequent API calls - can be cryptographically verified to originate from the same client context.

Context Continuity Enforcement Layer (CCEL). An independent actor with sole responsibility for context continuity. Deployed at the service boundary by the RS operator. Not a party to the AM contract. Creates and verifies binding artifacts external to the AM, without IdP participation. Delivers the client-side proof generation component, parameterized from the server-side configuration, as a service worker or client-side library.

Stateless (CCEL). CCEL's baseline model is stateless: core continuity verification requires no mutating runtime state across requests. Each request is verified using only configured policy, signing keys, and the request's own contents. Configuration, secrets, and pre-loaded keys are not considered state in this sense.

Binding artifact. A signed artifact created by the enforcement layer at context establishment time, containing the hash of the authentication material and the hash of the client’s public key, signed by the enforcement layer’s key. Delivered to the client and presented on subsequent requests to enable session verification.

DPoP proof (proof JWT). A short-lived JWT signed by the client’s private key, containing the public key via the JOSE header, a unique identifier, a timestamp, and the method and URI of the target request. Per-request uniqueness, where required, is enforced at the platform layer (gateway/edge) rather than within the continuity layer; the raw serialized proof itself serves as the dedup key - no field extraction required. Defined in RFC 9449.

Enforcement layer. A request-intercepting component - such as a WAF, API gateway, reverse proxy, or sidecar - responsible for requiring and verifying DPoP proofs and binding artifacts on covered request paths, and for creating binding artifacts at context establishment.

Authentication route. A request path whose response produces authentication material - the origin point of the client context. The enforcement layer observes the response, extracts the authentication material, and creates a binding artifact that records the relationship between that material and the client’s public key.

API route. A request path that consumes authentication material produced by an authentication route to perform an operation, without producing new authentication material. The enforcement layer verifies, on every such request, that the presented authentication material and the proof’s signing key match the binding artifact established at context establishment.

Appendix B: Protocol Flows

B.1 Verification Functions

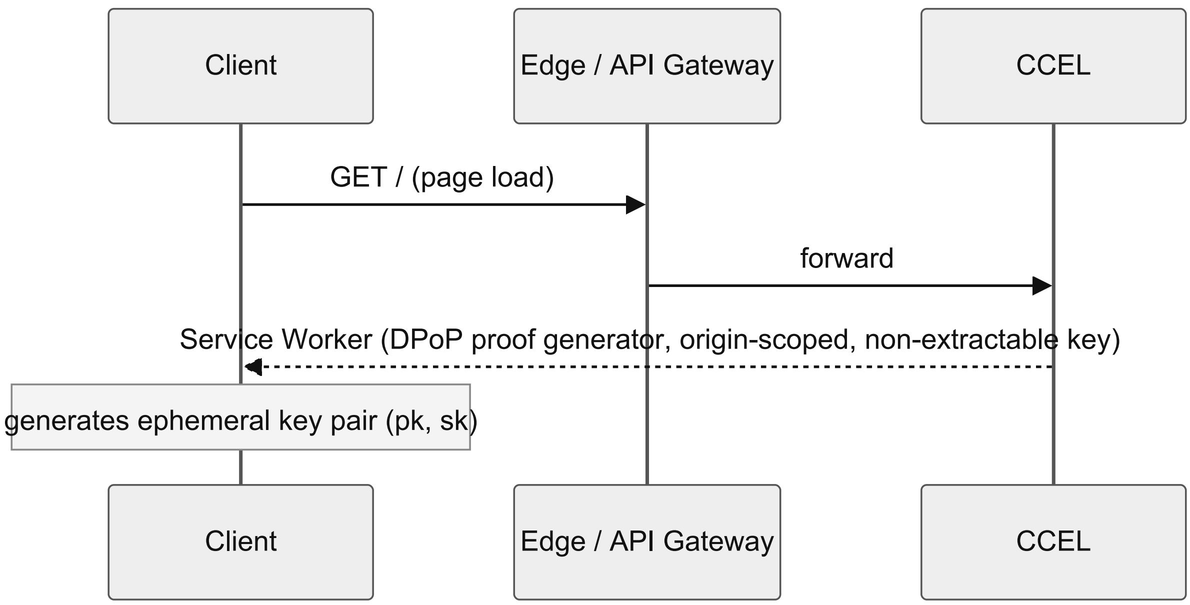
```
DPoPVerify(proof, request):
    verify proof signature
    verify proof.iat within configured skew window
    verify proof.htm == request.method
    verify proof.htu == request.uri
    return pk  // public key extracted from proof
```

```
BindingVerify(B, A, proof, request):
    pk ← DPoPVerify(proof, request)
    verify B signature (enforcement layer key)
    verify H(pk) ∈ B
    verify H(A)  ∈ B
```

`A` denotes whichever AM is bound by `B`: for the final binding it is the access token; for a temporary binding `B_t` (see B.3) it is the intermediate AM (e.g. an MFA session token).

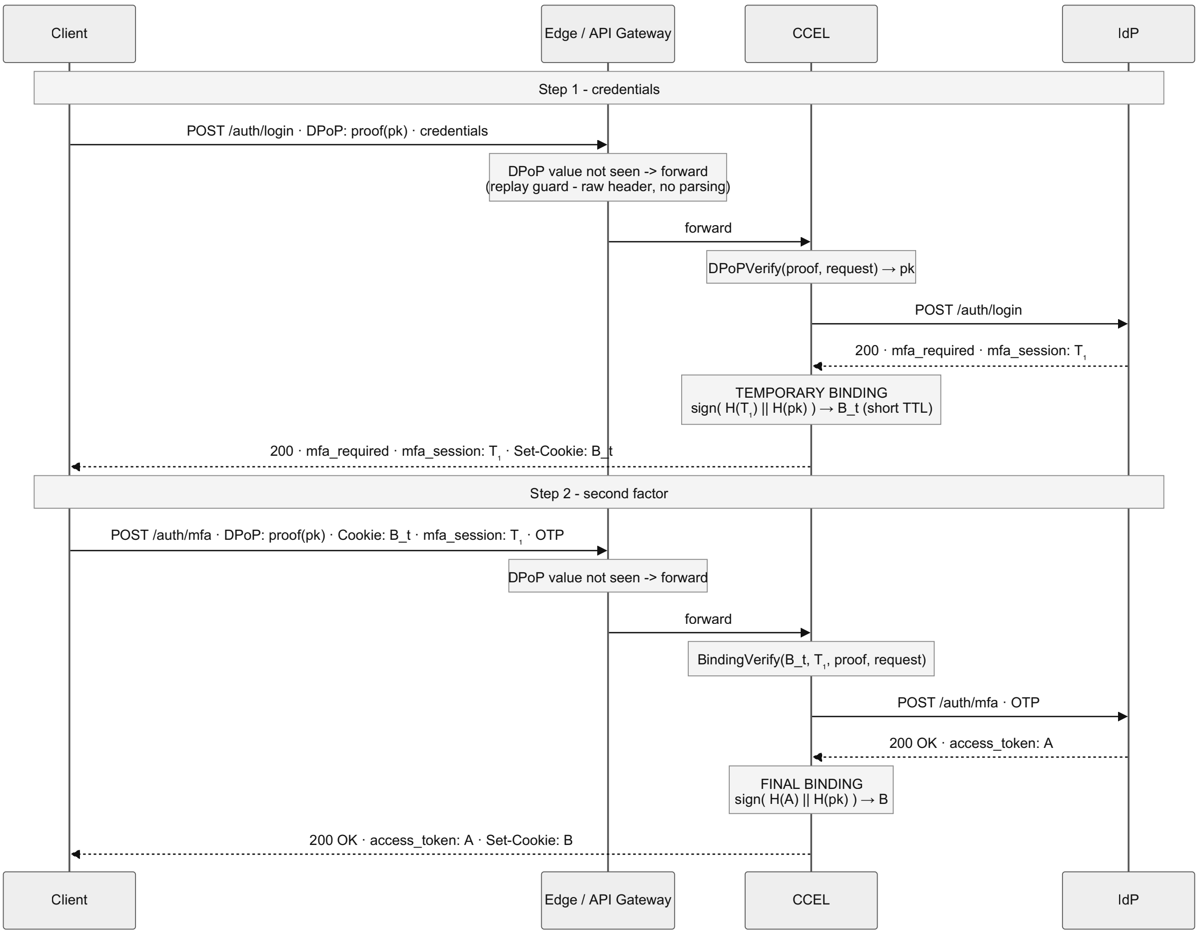
B.2 Client Logic Delivery

CCEL delivers the proof generation component to the client before any authentication takes place. The service worker is parameterized from the server-side configuration and runs in an origin-scoped sandbox; the private key it generates is non-extractable and never leaves the browser.



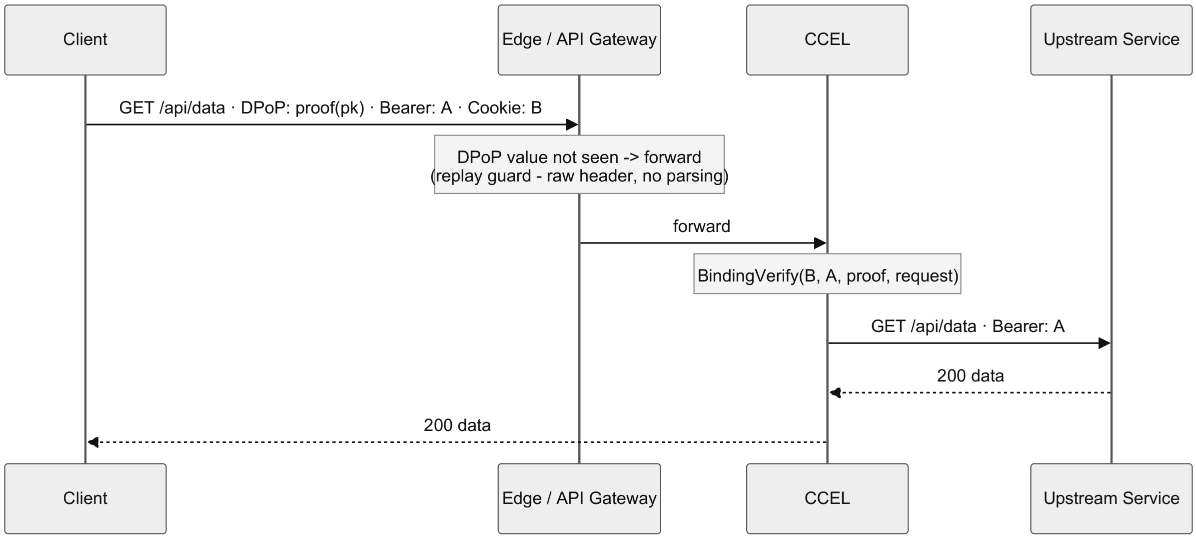
B.3 MFA Ceremony - Temporary and Final Binding

The same ephemeral key pair is used throughout the ceremony. After step 1 (credentials), CCEL creates a short-lived temporary binding on the intermediate session token. After step 2 (second factor), it creates the final binding on the issued access token. The edge / API gateway tier blocks proof replay by deduplicating on the raw `DPoP` header value - no JWT parsing required; the full serialized proof is the replay key.



B.4 Session Continuation - API Call

On every subsequent request the edge / API gateway tier deduplicates on the raw `DPoP` header value to block replay - no JWT parsing needed, the full serialized proof is the key. CCEL then calls `BindingVerify`, which ties the presented authentication material and public key to what was recorded at context establishment.



Appendix C: Mapping to RFC 9449 Concepts

The following table maps the concepts defined in RFC 9449 (DPoP) to their equivalents in the externalized sender constraint model described in this paper.

RFC 9449 Concept	This Model Equivalent	Notes
DPoP proof JWT	Proof JWT	Identical format and validation rules
DPoP public key (`jwk` in proof header)	Client public key `pk`	Same field, same usage
`cnf` claim in access token	Binding artifact `B`	Functionally equivalent; `cnf` is inside the token, `B` is external
`cnf.jkt` (key thumbprint)	`H(pk)` in binding artifact	Same semantic content
IdP embeds `cnf` in token	Enforcement layer creates `B`	Functional equivalence; no IdP participation required
RS verifies `cnf` matches proof `pk`	Enforcement layer verifies `H(pk)` in `B` matches proof `pk`	Same verification logic, different enforcement point
`htu` and `htm` claims	`htu` and `htm` claims	Identical - RFC 9449 format
`jti` nonce field	`jti` nonce field	Same field; CCEL does not validate it. Raw proof is the platform dedup key.
`iat` timestamp	`iat` timestamp	Identical - RFC 9449 requirement
Token is sender-constrained	Session is sender-constrained via `B`	Same property, different binding mechanism

The key structural difference: in RFC 9449, the binding between token and key is encoded inside the token (requires IdP participation). In this model, the binding is encoded in an external artifact maintained by the enforcement layer (requires no IdP participation).

Appendix D: Example Request and Response Structures

The following examples illustrate the HTTP structures used in the protocol flow. Field values are illustrative; actual proof JWTs and binding artifacts are base64url-encoded signed JWTs. Header and cookie names are implementation-defined.

Authentication request (authentication route):

```
POST /auth/login HTTP/1.1
Host: api.example.com
Content-Type: application/json
DPoP: eyJhbGciOiJFZERTQSIsImp3ayI6eyJrdHkiOiJPS1AiLCJjcnYiOiJFZDI1NTE5IiwieCI6Ii4uLiJ9fQ.eyJodG0iOiJQT1NUIiwiaHR1I

{"username": "alice", "password": "..."}
```

The `DPoP` header contains a proof JWT with:

- Header: algorithm (`EdDSA`), public key (`jwk`)
- Payload: `htm` (POST), `htu` (https://api.example.com/auth/login), `iat` (current timestamp), `jti` (unique identifier)
- Signed with the client's private key

Authentication response (with binding artifact):

```
HTTP/1.1 200 OK
Content-Type: application/json
Set-Cookie: __continuity=eyJ...<binding_artifact>...; HttpOnly; Secure; SameSite=Lax; Path=/

{"token": "eyJ...<access_token>...", "expires_in": 3600}
```

The binding artifact contains:

- `H(A)` - hash of the authentication material (the access token, in this example)
- `H(pk)` - hash of the client's public key
- Signed by the enforcement layer's key

API request (API route):

```
GET /api/resource HTTP/1.1
Host: api.example.com
Authorization: Bearer eyJ...<access_token>...
DPoP: eyJhbGciOiJFZERTQSIsImp3ayI6eyJrdHkiOiJPS1AiLCJjcnYiOiJFZDI1NTE5IiwieCI6Ii4uLiJ9fQ.eyJodG0iOiJHRVQiLCJodHUiO
Cookie: __continuity=eyJ...<binding_artifact>...
```

The enforcement layer calls `BindingVerify(B, A, proof, request)` as defined in Appendix B.1, with `A` taken from the `Authorization: Bearer` header, `B` from the `__continuity` cookie, and `proof` from the `DPoP` header.

If verification succeeds, the request is forwarded to the upstream service. If any check fails, the configured failure response (see §6.5) is returned immediately without upstream forwarding - typically `401 Unauthorized` for cryptographic verification failures and `428 Precondition Required` when the binding artifact is missing.