Operational Realities of Sender-Constrained OAuth: Replay-State Scaling Thresholds and Silent Binding Failures in Distributed Resource-Server Deployments

Sergio Emiliano Mancinas Fontalvo
Communication Systems and Networking
TH Köln, Cologne, Germany
sergio_emiliano.mancinas_fontalvo@smail.th-koeln.de

***Abstract*—Sender-constraining binds an OAuth 2.0 access token to a key or certificate held by the legitimate client, defending against the token-replay threat behind recent large-scale incidents. The OpenID FAPI 2.0 Implementation Advice Sec. 2.3 (April 2026) permits two mechanisms, Demonstrating Proof-of-Possession (DPoP, RFC 9449) and mutual TLS (RFC 8705), and advises against deploying both. To our knowledge no peer-reviewed measurement compares them as sender-constraining mechanisms in a distributed resource-server topology. We derive, from the published source of the Spring Security framework, the request-rate threshold above which a bounded process-local DPoP replay cache admits proof replay, together with the closed-form sizing rule a correct cache must satisfy. For the shipped defaults of Spring Security 6.5 a single node admits replay of an honestly issued proof above approximately seventeen requests per second, and is safe against a client that chooses its own issued-at time only below approximately eight. This is a single-node correctness defect, distinct from the known multi-node limitation, and it admits replay silently on the default code path. We confirm the predicted behaviour on a single-host deployment of Spring Security 6.5 and Keycloak 26.6, observing rejection at and below 17 requests per second and admission at and above 18, and we show that the eviction budget is controlled by the attacker rather than by deployment load. As a secondary contribution we document a taxonomy of *silent binding failures* in widely-deployed OAuth implementations, grounded in public defects in Spring Security and Keycloak. We further specify a pre-registered protocol for the broader empirical comparison across bearer, DPoP, and mTLS configurations, which remains future work. The analysis is consistent with the FAPI 2.0 guidance against simultaneous deployment, though it does not measure the operational cost on which that guidance rests, and it frames the bounded-replay-cache pattern as a deployment-grade attack surface beyond the multi-node case named in RFC 9449. The measurement protocol and the reproducibility harness, including the single-node validation of this threshold, are available in the artifact repository.**

***Index Terms*—OAuth 2.0, DPoP, mutual TLS, sender-constrained access tokens, FAPI 2.0, replay defense, distributed systems, Spring Security framework, Keycloak IAM**

I. Introduction

In August 2025, the threat actor UNC6395 exfiltrated OAuth access and refresh tokens from the Salesloft Drift third-party integration to Salesforce, a widely deployed customer-relationship management platform, enabling unauthorized access to more than seven hundred Salesforce customer environments [1]. The defense specified for this attack class, sender-constraining the access token to a key or certificate held only by the legitimate client, has been available as a standards-track mechanism for over five years. RFC 8705 (Mutual TLS Client Authentication and Certificate-Bound Access Tokens) was published in 2020 [2]; RFC 9449 (Demonstrating Proof-of-Possession) followed in 2023 [3]. Most affected organizations could have deployed one of these mechanisms. The empirical question is why they did not, and what it would cost to do so at scale.

The OpenID Foundation's *FAPI 2.0 Implementation Advice* Sec. 2.3 addresses this question directly [4].[1] Its DPoP considerations (Sec. 2.3.1) name the operational costs this paper is concerned with: per-request cryptographic operations with possible performance implications, protocol complexity including HTTP URL normalization and optional server-provided nonces, and less mature library support than mTLS. The mTLS considerations (Sec. 2.3.2) note that mTLS authenticates and sender-constrains in one step, constrains the whole request, and amortizes its asymmetric cost over the TLS handshake. The Selection Guidance (Sec. 2.3.3) concludes that both are valid and directs that *"Implementers should not use both DPoP and MTLS simultaneously for sender constraining."*

To our knowledge, no peer-reviewed measurement compares DPoP and mTLS as OAuth sender-constraining mechanisms in a distributed resource-server topology. The closest formal work, the FAPI 2.0 security analysis by Hosseyni, Küsters, and Würtele [5], identifies the *DPoP Proof Replay* attack at the protocol level but does not measure operational cost; the closest measurement work targets service-mesh mTLS overhead in general (e.g., Bremler-Barr et al. [24]) rather than sender-constrained OAuth tokens specifically. Vendor and practitioner literature [6]–[8] provides qualitative comparison

[1] A working-group draft, cited at the snapshot current at the time of writing. This paper writes *mTLS* throughout; quotations and reference titles preserve the spelling of their source.

but no rigorous benchmarking against a controlled distributed topology. This paper derives the bounded-cache replay threshold analytically and pre-registers the protocol to close the empirical gap.

A second class of operational concerns emerges from the integration of sender-constrained tokens with intermediate infrastructure. RFC 8705 Sec. 6.5 states that "an authorization server or resource server MAY choose to terminate TLS connections at a load balancer, reverse proxy, or other network intermediary. How the client certificate metadata is securely communicated between the intermediary and the application server, in this case, is out of scope of this specification" [2]. RFC 9449 Sec. 11.1 similarly leaves single-use `jti` replay defense as out-of-spec when "multiple servers behind a single endpoint have no shared state" [3]. Both gaps are operationally realized in production deployments and constitute a family of what we call *silent binding failures*: configurations in which a sender-constrained access token traverses a boundary whose semantics admit replay or impersonation undetectably.

This paper makes two contributions.

1) *An analytical replay-threshold and its demonstration.* We derive, from the published source of the Spring Security framework, the request-rate threshold above which a bounded process-local replay cache admits DPoP proof replay on a single node, give the closed-form rule a correct cache must satisfy, and confirm both on a runnable stack. We further show the eviction budget is controlled by an attacker or a co-tenant, not only by honest load.
2) *A taxonomy of silent binding failures.* We organize three publicly-disclosed defects in widely-deployed OAuth implementations into deployment failure classes, each an instance where a sender-constrained token is admitted without its binding being enforced.

Contribution 1 answers *which mechanism to deploy and at what cost*; contribution 2 answers *how the chosen mechanism fails silently after deployment*. We also specify a pre-registered protocol, that is, hypotheses and an analysis plan committed publicly before any data is collected, so results cannot be fitted to the plan after the fact, for the broader DPoP-versus-mTLS cost comparison; that comparison is future work.

The analysis targets the Spring Security framework version 6.5 as resource server and Keycloak as identity and access management (IAM) system. Both are open source, so the replay cache is auditable from published source rather than inferred from black-box behaviour, and each is widely deployed in its role. A defect in a shipped default of either therefore affects a large installed base and is a property of that default, not of a misconfiguration.

II. Background

A. OAuth 2.0 Bearer Tokens and the Replay Problem

OAuth 2.0 [13] specifies the bearer access token: a string credential whose possession is sufficient for use. Token theft at rest is therefore equivalent to authentication of the legitimate client until the token expires. Short token lifetimes reduce but do not eliminate this risk; refresh-token rotation [14] propagates the risk to a longer-lived credential.

Sender-constraining binds the access token to a cryptographic artifact held by the legitimate client, a private key (DPoP) or a TLS client certificate (RFC 8705), and is layered on top of OAuth 2.0 issuance and resource-access flows without modifying them.

B. DPoP Mechanics (RFC 9449)

A DPoP proof is a JSON Web Signature [15] carried in the `DPoP` HTTP header. Its JOSE (JavaScript Object Signing and Encryption) header includes `typ=dpop+jwt`, an algorithm identifier (typically `ES256`), and the public JWK (JSON Web Key) of the client's key pair. Its payload includes:

- `jti`, unique identifier per proof (for replay defense)
- `htm`, HTTP method
- `htu`, HTTP URI (without query or fragment, per spec)
- `iat`, issued-at timestamp
- `ath`, base64url-encoded SHA-256 of the access token, when the proof accompanies a resource-server request
- Optionally, a server-provided `nonce`

At issuance, the authorization server embeds `cnf.jkt` (the SHA-256 thumbprint of the client JWK) in the access token per RFC 7800 [16]. At use, the resource server verifies the proof signature against the embedded JWK, checks `htm`, `htu`, `iat`, and `ath`, looks up `jti` in a replay-prevention cache, and confirms that the recomputed thumbprint matches `cnf.jkt`.

RFC 9449 Sec. 11.1 names the `jti` single-use check as the critical replay defense, and explicitly acknowledges its operational difficulty in deployments with multiple resource-server replicas lacking shared state.

C. mTLS-Bound Tokens (RFC 8705)

RFC 8705 binds the access token to the SHA-256 thumbprint of the client's TLS certificate, embedded as `cnf.x5t#S256`. The resource server extracts the certificate from the TLS layer at request time, computes the thumbprint, and compares against the bound value. No per-request signing is required; the binding is amortized across the TLS handshake and any session reuse [17].

Sec. 6.5 of the specification leaves out of scope how the certificate metadata is conveyed to the resource server when TLS terminates at an intermediary, a deployment topology common in modern cloud architectures.

D. FAPI 2.0 Sender-Constraining Mandate

The OpenID FAPI 2.0 Security Profile [18] requires sender-constrained access tokens for confidential clients in high-value deployments and permits mTLS or DPoP as the binding mechanism. The Implementation Advice document [4] cited above expands on the selection trade-offs. The FAPI 2.0 Attacker Model [45] defines six attacker types A1–A6 of increasing power, with sender-constraining specifically defending against A5 (an attacker with full read access to an honest client's storage, the Salesloft archetype).

III. Related Work

A. Formal Analysis

Hosseyni, Küsters, and Würtele [5] published a formal security analysis of the FAPI 2.0 family of protocols in *ACM Transactions on Privacy and Security* (Volume 28, No. 1, November 2024). The paper identifies three new attacks: *DPoP Proof Replay*, *Attacker Token Injection*, and *Client Impersonation*. The verbatim statement load-bearing for the present work, from their treatment of DPoP Proof Replay (Sec. 3.3), is: *"neither FAPI 2.0++, nor DPoP itself, nor the specifications on which DPoP is built mandate for DPoP proofs to be strictly one-time use. Hence, the specifications do not mandate the RS to reject a replayed proof."* Their analysis is symbolic and does not address deployment cost; the present work is complementary, addressing the operational envelope their analysis assumes. A more recent result from the same group, audience injection attacks [36], affecting signature-based client authentication across OAuth, OpenID Connect, FAPI, and related flows, further underscores that protocol-level soundness and deployment-level correctness are distinct concerns.

B. Measurement-Adjacent OAuth Security

Static and dynamic analyses of OAuth implementations are extensive: Cerberus [19] (47 violations across 10 libraries), OAuthLint [20] (316 Android apps), OAuch [21] and its follow-up [22] (deployed IdPs miss 20% of required and 33% of recommended controls), and Innocenti et al. [23] (249 brokered SSO providers). None measures the runtime cost of sender-constrained variants. On the performance side, Bremler-Barr et al. [24] measure mutual TLS across service meshes (up to 166% p99 latency growth under load), but for service-mesh mTLS in general, not OAuth tokens and not DPoP; Cloudflare's Privacy Pass work [25] and Authrim's load tests [26] report shared-cache and token-exchange throughput but not RFC 9449 or RFC 8705 sender-constraining specifically. The IETF draft *TLS-Session-Bound Access Tokens* [11], binding tokens to the TLS session via RFC 5705 exporters [27] to avoid per-request DPoP cost, shows the standards community is actively debating that cost envelope even absent a measurement.

C. Research Gap

To our knowledge, no peer-reviewed study has combined the FAPI 2.0 attacker-model reasoning with a multi-replica analysis of DPoP and mTLS comparative cost. This paper contributes the analytical model and a pre-registered protocol for the empirical comparison.

The gap is not closing on the standards side. RFC 9700 [40], the current best-current-practice document, recommends sender-constraining without addressing the state a resource server must hold to enforce it. The OAuth working group's active update to that document [41], authored in part by the same group as the FAPI 2.0 formal analysis [5], mentions DPoP once and does not discuss replay caches or the `jti` claim at all. Replay-cache behaviour therefore remains outside both the deployed best practice and its pending revision.

IV. Methodology

This section specifies the measurement protocol committed to public pre-registration prior to any data collection. The analytical result of Sec. V is established independently of this protocol; the protocol below governs the empirical validation of that result and the broader DPoP-versus-mTLS cost comparison, the full distributed portion of which is future work (Sec. V-D).

A. Stack and Conditions

The analytical result of Sec. V needs no measurement hardware; its single-host validation (Sec. V-D) runs on one machine, since the threshold concerns cache eviction rather than network latency. The benchmark stack is Docker Compose-orchestrated: Keycloak 26.6.1 (DPoP realm, `ES256`, `cnf.jkt`) as authorization server, Spring Boot 3.x with Spring Security 6.5 resource servers, the Spring default `LinkedHashMap`-backed `jti` cache for single-node conditions and Redis 7 for distributed ones, and NGINX-terminated mTLS with explicit X-Forwarded-Client-Cert propagation. The full stack definition is in the artifact repository [37]. Seven conditions are defined: C0 bearer baseline; D1 single-node default cache; D2 three-replica Redis cache; D4 server-provided nonces (RFC 9449 Sec. 8); M1 mTLS certificate-bound tokens; and X1/X2 for correct and broken X-Forwarded-Client-Cert (XFCC) propagation. The distributed latency comparison, deferred to future work, adds a two-host network path and a cloud VM pair for cross-environment validation, on commodity hardware under exclusive control per Maricq et al. [9].

B. Statistical Pipeline

The binary single-node outcome of Sec. V-D needs no distributional statistics. The latency comparison, when run, generates load with `wrk2` and `k6` under HdrHistogram capture [28] and analyses the resulting p99 latencies with standard non-parametric methods, chosen because latency distributions are skewed and not normal. In outline: replication count is sized by CONFIRM-style pilot analysis [9] so the bootstrap confidence interval on median p99 is within $\pm 5\%$; the Kruskal-Wallis test asks whether the conditions differ at all; Dunn post-hoc tests, with Holm correction for the multiple pairwise comparisons, identify which pairs differ; and Cliff's δ reports how large each difference is, read against the Romano-Kromrey magnitude thresholds [46]. The full plan is in `PROTOCOL.md` [37].

Implementing this pipeline against synthetic data before any measurement exposed a defect in the pre-registered stationarity rule. Split-half Kolmogorov-Smirnov assumes independent samples, and latency series are autocorrelated: on 400 AR(1) series stationary by construction it rejects 3.0% at $\varphi = 0$, 22.0% at $\varphi = 0.5$ and 88.5% at $\varphi = 0.95$, against a nominal 5%, so applied literally it would discard most valid windows. We record this in `docs/DEVIATIONS.md` rather than silently amend the plan, since no latency data has yet been

collected. A pre-registered analysis plan is itself an artifact that can be tested, and testing it early is cheaper than discovering its defects afterward.

C. Pre-Registration

The pre-registered hypothesis document is committed to gi thub.com/sergioemancinas/dpop-replay-threshold before the first measurement run. Falsifiable hypotheses include:

- **H1:** Distributed cache (D2) increases p99 by $\geq$1 ms over D1 below 1000 req/s.
- **H2 (headline):** There exists a request rate λ^* per resource-server connection at which D2 p99 crosses M1 p99 from below, given session reuse, or no such crossover exists.
- **H3:** D4 reduces p99 variance compared to D2 by $\geq$20%.
- **H4:** A closed-form prediction `p99(λ, r) = a + b·λ + c·(r−1)·log(λ)` fits observed data within $\pm$15%, where r is replica count.

D. Correctness Suite

Prior to any benchmark measurement, a correctness suite validates:

- `jti` rejection on second use, within and across replicas;
- stale-proof rejection under clock skew of $\pm$2 seconds;
- `iat` window-edge behavior at expiry-minus-1-second and expiry-plus-1-second;
- `ath` and `htm`/`htu` mismatch rejection;
- partition behavior under Redis primary loss, fail-closed expected.

On the executed single-host configuration the suite runs 20 checks, of which 18 pass and 2 are skipped as requiring Redis or multiple replicas [37]; the cross-replica and Redis cases run in the D2 stack of Sec. VI-A. The bounded-cache behavior itself is the subject of Sec. V.

V. Primary Contribution: The Bounded-Cache Replay Threshold

The primary contribution is an analytical result, derived from the published source of a Tier-1 OAuth library and reproducible against it: the sustained per-node request rate above which a bounded process-local `jti` replay cache admits DPoP proof replay. The result is established without measurement; we state it as a closed-form model, instantiate it for the default configuration of Spring Security's resource-server implementation, and give the sizing relationship a correct cache must satisfy. The pre-registered protocol of Sec. IV governs its empirical validation and the broader DPoP-versus-mTLS cost comparison, the execution status of which is stated in Sec. V-D.

A. Model

A process-local replay cache is characterized by its capacity bound `N` (entries) and by the `iat` validation that bounds how long a proof is accepted. The check is two-sided: with clock-skew tolerance `S`, a proof carrying issued-at `iat` is accepted whenever the current time lies in `[iat − S, iat + S]`, a span of `2S`. How much of that span a replay can use depends on the `iat` the proof carries, and this distinction turns out to matter, so we keep the two cases separate.

TABLE I
Honest-traffic replay threshold $R^* = N/S$ and safe rate $N/2S$ for the default clock-skew tolerance $S = 60$ s.

N	S (s)	$R^* = N/S$ (req/s)	Safe $N/2S$ (req/s)
1000 †	60 †	≈ 17	≈ 8
1000	30	≈ 33	≈ 17
10,000	60	≈ 167	≈ 83
100,000	60	$\approx 1{,}667$	≈ 833

† Spring Security shipped default.

For a proof issued honestly, `iat` is the moment of first use, so a replay is accepted only in the forward half, up to `S` seconds after first use. Under an insertion-order ("evict-eldest") overflow policy, an entry inserted at sustained arrival rate `R` is displaced after approximately `N / R` seconds. Replay of an honestly issued proof is therefore admitted when

`N / R < S` $\Leftrightarrow$ `R > R⋆ = N / S`.

This `R⋆` is the threshold the single-host validation of Sec. V-D measures, because that experiment issues proofs with an honest `iat`. It is independent of any per-entry time-to-live longer than `S`: above `R⋆`, the capacity bound governs eviction at any rate. The exploitable interval, between eviction at `N/R` and `iat` expiry, widens as `R` grows: at `R = 2R⋆` the original `jti` is evicted after `S/2`, so severity increases with load.

The defensive bound is different, and looser, because the client controls `iat`. A proof dated `iat = now + S` is accepted for the full `2S` after its first presentation, so a cache that must remain correct against a client that is merely clock-skewed, let alone hostile, has to retain each `jti` for `2S`. The safe threshold is therefore `N / 2S`, half the honest-traffic figure. We carry both forward: `R⋆ = N/S` as the rate above which ordinary traffic begins to admit replay, and `N/2S` as the rate below which the cache is safe against any `iat` a client may choose.

B. Threshold under Spring Security defaults

Spring Security's `JtiCache`, added in release 6.5.0 [42], fixes `N = 1000`, and its `JwtIssuedAtValidator` sets a clock-skew tolerance `S = 60 s` by default, so the acceptance span is `2S = 120 s`. The resulting thresholds:

At the shipped defaults a single node admits replay of an honestly issued proof above approximately seventeen requests per second, and is safe against a client that chooses its own `iat` only below approximately eight. Both figures sit within enterprise OAuth deployment ranges and below the load at which an operator would suspect a replay-defense failure. We do not have an operator survey to quantify "within range" and state it as context, not as a measured claim.

C. Safe-Sizing Corollary

Inverting the threshold gives the minimum capacity a correct implementation must provision for a target sustained peak rate

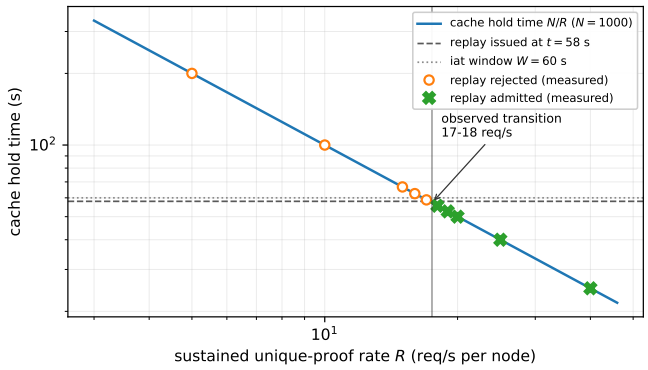

Fig. 1. Cache hold time N/R versus sustained request rate R for the default 1000-entry cache. A replay of an honestly issued proof is admitted where the hold time falls below the $S = 60$ s forward half of the acceptance span, i.e. above $R^* \approx 17$ req/s/node.

TABLE II
MINIMUM CACHE CAPACITY $N \geq P \cdot 2S$ FOR A TARGET AGGREGATE PEAK RATE P (DEFAULT $S = 60$ S, SO $2S = 120$ S).

Aggregate peak P (req/s)	Required cache $N \geq P \cdot 2S$
8	$\approx$960 ($\approx$ default)
50	6,000
100	12,000
250	30,000
500	60,000

`P`. The defensive form uses the full acceptance span `2S`, not its forward half, because a correct cache must resist a client that dates `iat` anywhere in the tolerated range, not only a client whose clock matches the server:

`N ≥ P · 2S`.

Here `P` is the peak *aggregate* insertion rate the node sustains, summed over every client and tenant sharing the process, since the cache is a single JVM-wide instance with no per-client partition (Sec. VI-A). `P` is not any one client's rate.

This is the relationship RFC 9449 Sec. 11.1 implies but does not state: a replay cache must retain at least one full acceptance span of proofs at peak aggregate load. The fixed 1000-entry bound in the Spring Security framework satisfies it only up to ≈ 8 aggregate req/s.

D. Empirical Validation

The model predicts a sharp, binary, directly testable outcome, replay is admitted if and only if `R > R*`, which a minimal single-host configuration suffices to confirm: one authorization server, one resource-server node at the default cache, a unique-`jti` load swept across rates spanning `R*`, and a replay probe issued within the `iat` window at each rate.

We executed this configuration. The stack is Keycloak 26.6.1 issuing ES256 DPoP-bound tokens to a single Spring Boot 3.5 resource server on Spring Security 6.5.5 at its default cache, with no Redis and no replicas, since either would replace the cache under test. Before measurement a correctness suite of 20 checks confirmed the replay defense operates as documented at idle: 18 passed, 2 were skipped as requiring Redis or multiple replicas, and none failed.

TABLE III
SINGLE-HOST REPLAY-ADMISSION SWEEP, PROBE REPLAYED AT $t = 58$ S. TWO REPETITIONS PER RATE, BOTH AGREEING.

R (req/s)	Insertions	Replay
5	290	rejected
10	580	rejected
15	870	rejected
16	928	rejected
17	986	rejected
18	1,044	**admitted**
19	1,102	**admitted**
20	1,160	**admitted**
25	1,450	**admitted**
40	2,320	**admitted**

One qualification governs the timing. The probe is evicted once `R × t > N`, where `t` is the delay before replay, and the replay must still satisfy `iat` validation, which requires `t < W`. The observable crossover is therefore `N/t`, and `R* = N/W` is its limiting case as `t` approaches `W`. Replaying at `t = 58` s rather than at the window edge avoids a decision determined by sub-second clock drift, and predicts a crossover at `1000/58 = 17.2` req/s.

Table III reports ten rates with two repetitions each. All twenty runs agreed with their repeat, every load request was accepted, and every replay fell inside the `iat` window. Replay was rejected at and below 17 req/s and admitted at and above 18 req/s, bracketing the predicted 17.2. The insertion counts show the mechanism directly: 986 insertions leave the probe inside the 1000-entry bound, and 1044 displace it.

This validates the analytical result for condition D1 on a single host. It is not a performance benchmark and yields no latency or capacity figures. The broader mechanism comparison specified in Sec. IV, DPoP distributed-cache and mTLS cost, together with hypotheses H1 through H4, requires the full distributed topology and remains future work (Sec. IX). Deviations from the pre-registered plan, including the replay-timing choice above and the use of a purpose-built load driver rather than `wrk2` for this binary outcome, are recorded in the artifact repository [37].

VI. SECONDARY CONTRIBUTION: SILENT BINDING FAILURE TAXONOMY

A *silent binding failure* is a deployment configuration in which a sender-constrained access token is presented at a boundary whose semantics admit replay or impersonation without protocol-level detection. The failure is silent because the request appears valid at every checkpoint at which validation occurs, and detection requires correlation across multiple system components or empirical workload simulation. We identify three classes, each illustrated with a publicly-disclosed instance in a widely-deployed open-source OAuth implementation.

A class qualifies when the underlying gap *can* manifest silently in a realistic deployment, not when it must. This distinction is not pedantry. Class B2 below has a publicly disclosed silent instance, and we also show an implementation of the same gap that fails loudly instead. Which behaviour a deployment gets is decided by an implementation choice the relevant standard declines to constrain, so the gap is properly classified here while the outcome is not a property of the class.

A. Class B1, Bounded Process-Local Replay Caches

Definition. A sender-constraining implementation maintains a replay-prevention cache local to a single process, sized to a bounded number of entries, and evicting on insert by a policy that does not respect the declared time-to-live of each entry. At sustained request rates above the threshold where cache insertions exceed eviction-window depth, previously-rejected proofs are no longer present in the cache and are accepted on re-submission.

Worked example (publicly disclosed). Spring Security's `JtiClaimValidator`, located in oauth2/oauth2-jose/src /main/java/org/springframework/security/oauth2/jwt/ DPoPProofJwtDecoderFactory.java of the Spring Security open-source repository [29], implements the RFC 9449 `jti` single-use defense via:

```
// static: one cache per JVM. putIfAbsent rejects a
    repeated jti.
private static final Map<String,Long> JTI_CACHE =
    Collections.synchronizedMap(new JtiCache());

private static final class JtiCache extends
    LinkedHashMap<String,Long> {
    private static final int MAX_SIZE = 1000;
    protected boolean removeEldestEntry(Map.Entry<
    String,Long> e) {
        return size() > MAX_SIZE            // evict
     on capacity, or
            || Instant.now().isAfter(       // on
    per-entry expiry
                Instant.ofEpochMilli(e.getValue()
    ));
    }
}
```

The threshold above which this cache admits proof replay, and the sizing relationship a correct cache must satisfy, are derived in Sec. V: for Spring's shipped defaults (`MAX_SIZE = 1000` entries, a 60-second `iat` acceptance window enforced by `JwtIssuedAtValidator`), replay is admitted above $R^\star \approx 17$ req/s per node.

What makes this a security finding rather than a tuning note is its *silence*, its *default*, and its *sub-detection threshold*. The admitted replay is indistinguishable from a legitimate request. In the single-host validation of Sec. V-D, the replayed proof returned HTTP 200 with the same response body as its first use, carried no `WWW-Authenticate` header, produced no log output at the resource server, and left no trace of the reused `jti` in the server logs. The behaviour occurs on the shipped default code path and requires no misconfiguration. It occurs at a request rate far below where an operator would suspect a replay-defense failure. "Provision a larger cache" presumes the operator knows the default is unsafe, and nothing in the library tells them.

RFC 9449 Sec. 11.1 already acknowledges that strict single-use "may not always be feasible ... when multiple servers behind a single endpoint have no shared state," language the community reads as a steer toward shared replay state. Our contribution is not to rediscover that steer but to quantify it: the bound $\texttt{cache\_size} \geq \texttt{peak\_rate} \times \texttt{iat\_window}$ (Sec. V-C) states *how large* a local cache must be, equivalently, the rate below which a local cache is safe at all, which the standard implies but never states, and which practitioner guidance [39] states only as a capacity-provisioning heuristic rather than as a security bound.

Scope precision matters here. With multiple resource-server replicas and no shared state, a replayed proof routed to a *different* node is accepted at any rate. That case is the multi-node limitation already filed as Spring Security #18120 [30]. That issue was closed as a duplicate of the DPoP customization enhancement #16940, with the maintainer stating that `JtiClaimValidator` "is intended to be a basic implementation for a single node setup" and directing deployments that need shared state to supply their own validator through `DPoPProofJwtDecoderFactory.setJwtValidatorFactory()`. No released fix changes the default behaviour at the time of writing.

The threshold of this section is the distinct *single-node* consequence of the same code path. It admits replay even on a single node, or under session-affinity routing, where the multi-node escape does not apply. Practitioner guidance states the provisioning form of the same arithmetic: implementation notes for a .NET DPoP library size a replay cache as request rate multiplied by the acceptance window [39]. That guidance assumes a store bounded only by time-to-live, does not treat a fixed-capacity cache whose eviction policy overrides its declared lifetime, and derives no rate above which replay is admitted. To our knowledge the single-node consequence of that overflow case has not been separately characterized. It is observable on public source and falls under the already-public root cause of #18120, requiring no further coordinated disclosure.

The eviction budget is attacker-influenceable, not a property of load. The threshold $R^\star$ describes when *honest* traffic begins to admit replay. It understates the exposure, because the insertions that evict a target `jti` need not come from honest traffic and need not be validated. Three facts about the shipped code path, all read from Spring Security 6.5.5 source, combine. First, `JtiClaimValidator` runs `JTI_CACHE.putIfAbsent` at the top of its `validate` method, so a proof is inserted before any later check can reject it. Second, that validator sits in a `DelegatingOAuth2TokenValidator` that invokes every delegate without short-circuiting, so the `ath` and `jkt` checks that would reject a proof run only after the insertion has already happened. Third, `JTI_CACHE` is `private static final`: one instance per JVM, shared across every client and tenant the process serves, with no partition.

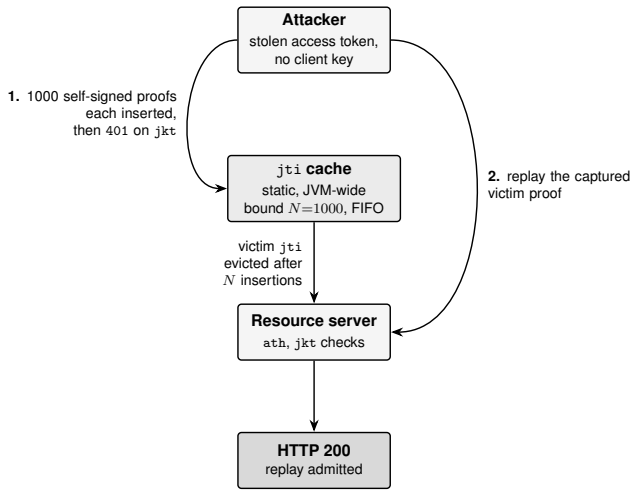

Fig. 2. Attacker-driven eviction on a single node. The `jti` is inserted before the binding check runs, so a flood of self-signed proofs that are each rejected still fills the bounded cache and evicts a captured victim proof, which then replays. Confirmed on the single-host stack [37].

Two consequences follow, both confirmed on the single-host stack [37]. In the first (Fig. 2), an attacker who holds a stolen access token but not the client's key mints proofs under a key of his own: correct `htm`, `htu` and `iat`, unique `jti`. Each is rejected with a `jkt` mismatch, and each nonetheless consumes a cache slot before that rejection. A thousand such proofs, sent in about a second at idle, evict a previously captured victim proof, which then replays successfully. $R\star$ is in this sense a budget the attacker spends, not a rate the deployment must reach. This variant is loud: the thousand rejections are exactly what anomaly detection is positioned to see.

The second consequence needs no attacker and produces no rejection. Because the cache is JVM-wide and unpartitioned, the ordinary successful traffic of one client evicts another client's `jti`. In the demonstration a second, fully legitimate client issues a thousand correctly bound requests, every one returning HTTP 200, and a low-traffic client's captured proof becomes replayable as a result. No stolen credential, no forged proof, and no rejected request appear on the eviction path. This is the security-relevant reading of the sizing rule: the P in $N \geq P \cdot 2S$ is the aggregate insertion rate across every client and tenant on the node, so on a busy multi-tenant node a low-traffic client can be forced below its own safe threshold by neighbours it does not control.

The threshold is a function of two parameters, not one. The sweep of Sec. V-D fixes the replay delay and varies the rate, which cannot by itself separate the rate from the insertion count. Fixing the rate and sweeping the delay does separate them: the admission crossover moves with the rate as $t\star = N/R$, observed at approximately 50 s for 20 req/s and 25 s for 40 req/s, with the boundary at about N insertions in both cases [37]. The outcome is governed by the product $R \cdot t$ meeting the capacity bound N, and the acceptance span $2S$ bounds the delay t over which a replay remains valid at all; the two-sided width of that span is confirmed directly by sweeping `iat` offset [37].

Mechanism affected. DPoP as implemented with a bounded process-local `jti` cache, which is the Spring Security default. The protocol is not at fault, and the defect is specific to the fixed-capacity in-process choice rather than to DPoP: RFC 9449 permits a strict single-use check, and a contemporaneous Go implementation [43] backs its `jti` store with a time-to-live cache bearing no capacity bound, evicting only on expiry and so never reaching the overflow condition analyzed here. Spring's fixed 1000-entry map cannot sustain a single-use check above $R\star$. mTLS is unaffected because its binding is at the transport layer and does not rely on a per-request `jti` replay cache.

B. Class B2, TLS Termination With Lossy Metadata Propagation

Definition. mTLS terminates at a network intermediary (load balancer, reverse proxy, API gateway, service mesh egress) ahead of the consuming component, and the intermediary either does not propagate client certificate metadata to the application layer, propagates it in a format the application cannot parse correctly, or accepts incoming `X-Forwarded-Client-Cert` headers from untrusted networks.

Worked example (publicly disclosed). CVE-2024-10039 ("Keycloak mTLS Authentication Bypass via Reverse Proxy TLS Termination") [31] is an instance: Keycloak deployments using a reverse proxy without pass-through TLS termination, with mTLS enabled, are vulnerable to authentication as any user by an attacker on the local network. The advisory was published November 25, 2024. The defect was patched in Keycloak 26.0.6.

RFC 8705 Sec. 6.5 explicitly places the secure propagation of client-certificate metadata across TLS-terminating intermediaries out of scope of the specification. Because the specification leaves the mechanism open, propagation across a TLS-termination boundary is implementation-specific: it can be done correctly, as in a vendor reference example that forwards the client certificate to a reverse proxy which introspects the token and verifies the `cnf` thumbprint [44], or incorrectly, as in CVE-2024-10039. The absence of a normative mechanism is what makes both outcomes possible from a conformant reading.

Silence is not intrinsic to this class. The same architectural gap produces opposite outcomes depending on how the consuming component treats missing metadata, and the standard does not say which to choose. Spring Security 6.5.5 makes the contrast concrete. Its `X509CertificateThumbprintValidator`, installed by default through `JwtValidators.createDefault`, obtains the certificate from the servlet attribute `jakarta.servlet.request.X509Certificate`, which a terminating proxy leaves unset unless application code restores it. We verified the consequence on a live Envoy deployment terminating mTLS ahead of a stock resource

server [37]. With the certificate propagated intact, the bound certificate is accepted and a different one rejected. With the `X-Forwarded-Client-Cert Cert` field absent, or present but percent-encoded such that the PEM armour does not parse, the same stock configuration returns HTTP 401 with `invalid_token` and the description "Unable to obtain X509Certificate from current request." Propagation loss therefore yields a conspicuous authentication outage whose cause is unobvious, not a silent acceptance. The Keycloak defect above sits at the opposite pole, resolving absent metadata in the attacker's favour on the client-authentication path.

The distinction matters for deployment guidance. A fail-closed consumer converts this class into an availability problem, which monitoring will surface within minutes. A fail-open consumer converts it into an authentication bypass, which monitoring will not surface at all. Because RFC 8705 Sec. 6.5 declines to specify the propagation mechanism, it also declines to specify which of these an implementation must exhibit.

Mechanism affected. mTLS-bound tokens. DPoP is not affected because its binding is at the application layer above any TLS termination point.

C. Class B3, Authorization Server Audience Bypass on Specific Flows

Definition. An authorization server enforces sender-constraining for some response types but not others. When a client is configured to require DPoP-bound tokens, the server issues bearer access tokens lacking `cnf` binding claims for response types where the implementation bypasses the enforcement check.

Worked example (publicly disclosed). Keycloak issue #48428 [32], reported April 23, 2026 by security researcher Evan Hendra via the Keycloak bug bounty program. When `dpop.bound.access.tokens=true` is configured on a client, Keycloak still issues front-channel bearer access tokens for OIDC implicit (`response_type=token`) and hybrid (`response_type=code token`) flows. The issued tokens carry `token_type=Bearer` without a `cnf` claim, silently bypassing the configured DPoP requirement. The issue was filed against Keycloak 26.5.3.

A related earlier disclosure, Keycloak #38333 [33], identified the inverse defect at the user-info endpoint: the DPoP proof was not bound to the access token, allowing a self-generated proof to be used. Fixed in Keycloak 26.2.0.

The default resource server does not compensate. An authorization-server defect of this shape is only exploitable because the other end of the deployment accepts a token that carries no binding, and the shipped defaults do. The two Spring Security 6.5.5 validators behave oppositely on an absent claim, which is worth stating precisely because the difference is the crux. `X509CertificateThumbprintValidator`, wired by default through `JwtValidators.createDefault`, reads `cnf.x5t#S256` and, when the claim is absent, returns success without further checking. `JwkThumbprintValidator`, nested in `DPoPAuthenticationProvider`, does the reverse: an absent `cnf.jkt` yields `jkt claim is required.` and the request fails closed. So the mTLS resource-server path accepts an unbound token outright, while the DPoP path rejects one, but only when the token is presented under the `DPoP` scheme. A token issued as `token_type=Bearer` with no `cnf` claim, which is the shape Keycloak issue #48428 [32] produces, is presented on the bearer path, where no binding validator runs at all.

We observed the mTLS case on two independently constructed stacks [37]: an unbound token is accepted with HTTP 200 both at an endpoint fronted by Envoy under strict validation and at a conventional mTLS endpoint. The composition follows. An authorization server that omits the binding claim and a resource server that only checks a binding the token declares together admit an unbound token end to end, with no error at either party.

The default does not compensate, but a deployment can. A small application-level guard that rejects any token lacking the expected `cnf` claim closes the gap; the artifact includes one, and its test records an unbound token receiving HTTP 403 at that endpoint while the default endpoint returns 200 [37]. What no stock configuration flag we are aware of does is make the default validators demand a binding the token did not declare.

This is the sense in which the failure is silent. Neither component is individually misconfigured, neither logs an error, and the deployment presents as sender-constrained throughout.

Mechanism affected. The composition is reachable whenever a token is issued without a binding claim and reaches a resource server that only enforces bindings that are declared. It is not specific to DPoP or mTLS, since the bearer path applies no binding check for either.

D. Discussion of the Taxonomy

B1 and B2 are *mechanism-specific*: each defeats one binding mechanism while the other remains an effective defense, B1 through a DPoP replay cache correct only below a load threshold and B2 through loss of the mTLS certificate binding across a terminating intermediary. B3 is not: it turns on a token issued without any binding claim reaching a resource server whose default path checks a binding only when the token declares one, which holds on the bearer path regardless of the intended mechanism, so switching mechanisms is not a mitigation. The classes also differ in visibility. B1 and B3 are silent. B2 need not be: whether it surfaces as a bypass or as an outage depends on how the consumer resolves absent certificate metadata, a choice RFC 8705 Sec. 6.5 leaves open with the propagation mechanism, and CVE-2024-10039 resolved it toward acceptance where stock Spring Security resolves it toward a loud 401.

Each surface is named only partially by the standards: RFC 9449 Sec. 11.1 acknowledges the multi-node cache but not the single-node sizing bound of B1, RFC 8705 Sec. 6.5

places B2's propagation out of scope along with the fail-open decision, and no standard requires a resource server to demand a binding its authorization server never supplied, which is what makes B3 composable across two individually defensible implementations. Sender-constraining is therefore necessary but not sufficient: correct deployment also requires cache sizing to the acceptance window, fail-closed certificate propagation across TLS termination, and uniform enforcement of the binding at both ends.

VII. Threats to Validity

We adopt the four-fold framework of Feldt [10]. Two categories of claim must be separated. The analytical threshold of Sec. V and the single-host validation of Sec. V-D are established here. The distributed latency comparison is not, so threats to it are stated as commitments for when it is run rather than as accounts of what was done.

Internal validity, executed work. The single-host result is a binary outcome, so it is insensitive to latency artefacts such as coordinated omission, garbage-collection pauses and just-in-time compilation warm-up. Its own principal threat is timing: the probe must be replayed while still inside the `iat` window, which the harness records for every run, and the effective crossover depends on the replay delay as `N/t`, discussed in Sec. V-D. A second threat is that the observed admission might reflect a broken deployment rather than cache eviction, which the pre-measurement correctness suite addresses by confirming that the same proof is rejected at idle. A third is that both repetitions of a rate could fail identically for a common reason; the insertion counts bracketing the 1000-entry bound argue against this, but only two repetitions per rate were run.

External validity. The single-host validation runs on one commodity machine over loopback, which is adequate because cache eviction is not a network property, but says nothing about production traffic patterns. The analytical threshold of Sec. V is independent of hardware.

Construct validity. The validation operationalizes "replay admitted" as HTTP 200 to a verbatim-replayed proof. This is the property that matters to an attacker, but does not by itself separate capacity eviction from any other validator failure; the correctness suite rules out the latter.

Planned work. Threats specific to the distributed latency comparison, coordinated omission, warm-up and GC artefacts, and the inferential statistics for H1 through H4, are stated with that comparison in Sec. IV and Sec. IX rather than repeated here, since it has not been run.

VIII. Discussion

A. The FAPI 2.0 Implementation Advice Claims

The Implementation Advice Sec. 2.3 [4] makes three operational claims about DPoP and three about mTLS. Our analysis bears on the DPoP claims as follows. The "per-request cryptographic overhead" and "protocol complexity" claims are the subject of the pre-registered cost measurement specified in Sec. IV, of which the bounded-cache threshold of Sec. V is the first analytical result. The "ecosystem maturity" claim is evidenced directly by Class B1 (Spring Security single-node bounded cache) and Class B3 (Keycloak #48428 flow-specific bypass): the largest-deployed Java and Java/Quarkus OAuth implementations exhibit specification-deviating behaviors in current releases.

The Implementation Advice further states explicitly: *"Implementers should not use both DPoP and MTLS simultaneously for sender constraining."* We do not test the combined deployment, and we have not measured the operational cost of either mechanism, so we offer no evidence for or against this guidance on cost grounds. What our results do show is that each mechanism carries its own distinct failure surface, B1 for DPoP and B2 for mTLS, so deploying both would require operating both correctly rather than obtaining redundancy for free. Each mechanism is appropriate for distinct client classes: mTLS for confidential server-to-server clients with stable TLS topology, DPoP for browser-based and public clients where transport-layer integration is impractical.

B. Availability of the Proposed Remedy

The formal analysis that identified DPoP Proof Replay also proposed a fix. Hosseyni et al. state that they "proposed to fix this attack by mandating the use of resource server-provided nonces with strict one-time use enforcement by the RS" [5]. A natural reading is that Class B1 is a solved problem awaiting adoption. Examining what implementations actually offer shows otherwise, in two distinct ways.

First, the remedy is stricter than the standard that names it. RFC 9449 Sec. 8 specifies server-provided nonces, but Sec. 11.1 permits nonce reuse "as long as the `jti` value is tracked and duplicates are rejected," which presumes exactly the sound `jti` tracking that Sec. V shows a bounded cache does not provide. A conformant nonce deployment therefore only replaces the window `W` with the nonce lifetime; the property that removes the threshold, strict single use, is the one RFC 9449 declines to require. Second, neither reference implementation offers nonces at all: the term does not appear in the Spring Security 6.5.5 DPoP surface, whose only error code is `invalid_dpop_proof`, and Keycloak issue #39042 [38] has been open and unmilestoned since April 2025. Third, closing the gap locally is not cheap: our single-use nonce implementation [37] runs to roughly 270 lines of security-critical Java because four extension points are closed (a replaced validator chain drops the private `ath`/`cnf.jkt` checks, `DPoPAuthenticationProvider` is `final` and flattens error codes, the entry point is private, and `ProviderManager` falls through to the library provider). The consequence is that the sizing rule of Sec. V-C is not a stopgap; for current versions of both implementations it is the only available control.

C. Model Context Protocol Authorization

The Model Context Protocol authorization specification version 2025-11-25 [12] mandates OAuth 2.1 with Proof Key for Code Exchange (PKCE) and RFC 8707 resource indicators for agent access to protected resources, and explicitly forbids

token passthrough. It does not mandate sender-constraining. As of May 2026, the modelcontextprotocol/ext-auth extensions repository [34] contains only two extensions, *Client Credentials* and *Enterprise-Managed Authorization*, and no extension specifying DPoP or mTLS binding. An MCP DPoP extension would be operationally valuable if MCP-mediated agentic workloads carry high-value access tokens whose theft would parallel the Salesloft archetype. We make no claim here about the implementation effort such an extension would require.

D. The Bounded-Cache Pattern Beyond Spring

Class B1 is illustrated in Spring Security but is structurally generic: any implementation with a bounded process-local replay cache under insertion-order or least-recently-used (LRU) eviction is exposed to the same defect. RFC 9449 errata or implementation guidance could specify the minimum cache-size-versus-window relationship of Sec. V-C, or require the eviction policy to reject acceptance while the eldest entry is still within its acceptance window.

IX. Future Work

The correctness conditions of the matrix were executed (Sec. VI); what remains is the *latency* comparison and its hypotheses. Hypotheses H1 through H4 are untested and no outcome is claimed for any of them; H2, the rate λ^* at which distributed-cache DPoP p99 meets mTLS p99, is the headline question of the pre-registered design and the reason the two-host topology is needed. The per-request cost claims quoted in Sec. I, and the statistical pipeline of Sec. IV-B that serves them, likewise await that comparison. Two smaller questions are open: whether the threshold differs between a thread-per-request container and an event-loop runtime (no difference is predicted, since the bound is a property of the validator, not the container), and the extension to bursty agentic workloads whose concurrent audience-bound tokens the WIMSE architecture [35] and the Model Context Protocol [12] frame.

X. Conclusion

Sender-constrained access tokens narrow the operational impact of token theft from a critical incident to a recoverable inconvenience, provided the binding mechanism is correctly implemented and operationally maintained. The OpenID Foundation's FAPI 2.0 Implementation Advice Sec. 2.3 names three operational claims about DPoP and mTLS, for which this paper derives the bounded-cache replay threshold analytically from the published library source and specifies a pre-registered protocol for empirical measurement. Beyond this threshold analysis, we document a taxonomy of silent binding failures grounded in publicly-disclosed defects across widely-deployed enterprise OAuth implementations, including a bounded process-local replay cache in Spring Security that admits proof replay on a single node at modest request rates. The findings refine deployment guidance and identify the bounded-cache pattern as an attack surface meriting protocol-level remediation guidance. They are consistent with the FAPI 2.0 recommendation against simultaneous DPoP and mTLS deployment, though we measure neither mechanism's operational cost and so offer no evidence on the cost grounds that recommendation rests on. The measurement protocol and reproducibility harness are available in the artifact repository [37]; future work executes the empirical comparison and extends it to agentic workload patterns, integrating the findings with the Model Context Protocol authorization specification.

Appendix A
Pre-Registration Statement

The measurement design and falsifiable hypotheses (H1–H4, Sec. IV-C) were committed to the public repository [37] before the first measurement run. The pre-registration commit `53e4ae2`, tagged `preregistration` and dated 2026-07-27 12:34:48 UTC, contains the pre-registration document and the protocol and nothing else: no stack definition, no measurement code, and no results. The first commit containing measured data, `bbd4d22`, is dated 46 minutes later, so the ordering is verifiable from the repository history. Deviations from the pre-registered plan are recorded in `docs/DEVIATIONS.md` and discussed in Sec. VII.

Appendix B
Reproducibility

The artifacts are available at [37] under the Apache License 2.0. From a clean checkout, `./reproduce.sh` builds the stack and regenerates the Sec. V-D result; each condition of Sec. IV has its own stack and committed reference outputs, and `check_results.py` checks a fresh run against the model, distinguishing a host too slow to sustain the load from a genuine mismatch. A Zenodo DOI will be assigned at publication, and the artifact is cited as [37].

References

[1] Google Threat Intelligence Group, "Widespread Data Theft Targets Salesforce Instances via Salesloft Drift," August 26, 2025. https://cloud.google.com/blog/topics/threat-intelligence/data-theft-salesforce-instances-via-salesloft-drift

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

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

[4] D. Tonge, *FAPI 2.0 Implementation Advice*, OpenID Foundation FAPI Working Group working copy, snapshot of April 15, 2026 (document revised 26 June 2026; accessed 27 July 2026). https://openid.bitbucket.io/fapi/fapi-2_0-implementation_advice.html

[5] P. Hosseyni, R. Küsters, and T. Würtele, "Formal Security Analysis of the OpenID FAPI 2.0 Family of Protocols: Accompanying a Standardization Process," *ACM Trans. Priv. Sec.*, vol. 28, no. 1, pp. 1–36, 2025 (online-first Nov. 2024), doi:10.1145/3699716. https://doi.org/10.1145/3699716

[6] I. Kawalec, "Sender Constrained Access Tokens: mTLS vs DPoP," SecureAuth (formerly Cloudentity) developer blog, Apr. 15, 2025. https://docs.secureauth.com/ciam/en/sender-constrained-access-tokens-mtls-vs-dpop.html

[7] Auth0, "Demonstrating Proof-of-Possession (DPoP)," developer documentation, 2025. https://auth0.com/docs/secure/sender-constraining/demonstrating-proof-of-possession-dpop

[8] Connect2id Server v19.2 release notes, September 2025. https://connect2id.com/blog/connect2id-server-19-2

[9] A. Maricq et al., "Taming Performance Variability," in *Proc. OSDI 2018*. https://www.usenix.org/conference/osdi18/presentation/maricq

[10] R. Feldt and A. Magazinius, "Validity Threats in Empirical Software Engineering Research – An Initial Survey," in *Proc. 22nd Int. Conf. Software Engineering and Knowledge Engineering (SEKE)*, 2010, pp. 374–379. https://www.cse.chalmers.se/~feldt/publications/feldt_2010_validity_threats_in_ese_initial_survey.pdf

[11] R. Krishnan, A. Prasad, D. Lopez, and S. Addepalli, "TLS-Session-Bound Access Tokens for OAuth 2.0," Internet-Draft draft-mw-oauth-tls-session-bound-tokens-07, Jun. 20, 2026. https://datatracker.ietf.org/doc/draft-mw-oauth-tls-session-bound-tokens/07/

[12] Model Context Protocol Authorization Specification, version 2025-11-25. https://modelcontextprotocol.io/specification/2025-11-25/basic/authorization

[13] D. Hardt (ed.), "The OAuth 2.0 Authorization Framework," RFC 6749, Oct. 2012. https://www.rfc-editor.org/rfc/rfc6749.html

[14] D. Hardt, A. Parecki, and T. Lodderstedt, "The OAuth 2.0 Authorization Framework," Internet-Draft draft-ietf-oauth-v2-1-15, Mar. 2, 2026. https://datatracker.ietf.org/doc/html/draft-ietf-oauth-v2-1-15

[15] M. Jones, J. Bradley, N. Sakimura, "JSON Web Signature (JWS)," RFC 7515, May 2015. https://www.rfc-editor.org/rfc/rfc7515.html

[16] M. Jones, J. Bradley, H. Tschofenig, "Proof-of-Possession Key Semantics for JSON Web Tokens (JWTs)," RFC 7800, Apr. 2016. https://www.rfc-editor.org/rfc/rfc7800.html

[17] E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.3," RFC 9846, Jul. 2026 (obsoletes RFC 8446). https://www.rfc-editor.org/rfc/rfc9846.html

[18] OpenID Foundation, "FAPI 2.0 Security Profile," Final Specification, February 2025. https://openid.net/specs/fapi-security-profile-2_0-final.html

[19] T. Al Rahat, Y. Feng, and Y. Tian, "Cerberus: Query-driven Scalable Vulnerability Detection in OAuth Service Provider Implementations," in *Proc. ACM CCS 2022*, arXiv:2110.01005. https://doi.org/10.1145/3548606.3559381

[20] T. Al Rahat, Y. Feng, and Y. Tian, "OAuthLint: An Empirical Study on OAuth Bugs in Android Applications," in *Proc. ASE 2019*. https://doi.org/10.1109/ASE.2019.00036

[21] P. Philippaerts et al., "OAuch: Exploring Security Compliance in the OAuth 2.0 Ecosystem," in *Proc. RAID 2022*. https://doi.org/10.1145/3545948.3545955

[22] P. Philippaerts, D. Preuveneers, and W. Joosen, "Revisiting OAuth 2.0 Compliance: A Two-Year Follow-Up Study," in *Proc. IEEE European Symp. Security and Privacy Workshops (EuroS&PW)*, 2023, pp. 521–525, doi:10.1109/EuroSPW59978.2023.00064. https://doi.org/10.1109/EuroSPW59978.2023.00064

[23] T. Innocenti, L. Jannett, C. Mainka, V. Mladenov, and E. Kirda, "“Only as Strong as the Weakest Link”: On the Security of Brokered Single Sign-On on the Web," in *Proc. IEEE Symp. Security and Privacy (S&P)*, 2025, pp. 1009–1027, doi:10.1109/SP61157.2025.00024. https://doi.org/10.1109/SP61157.2025.00024

[24] A. Bremler-Barr, O. Lavi, Y. Naor, S. Rampal, and J. Tavori, "Performance Comparison of Service Mesh Frameworks: The MTLS Test Case," in *Proc. IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2025. (Extended technical report: arXiv:2411.02267.). https://doi.org/10.1109/NOMS57970.2025.11073712

[25] Cloudflare, "Reducing double spend latency from 40 ms to < 1 ms on privacy proxy," engineering blog, Aug. 5, 2025. https://blog.cloudflare.com/reducing-double-spend-latency-from-40-ms-to-less-than-1-ms-on-privacy-proxy/

[26] Authrim, "Token Exchange Load Test," vendor load-test report, Dec. 2025. https://authrim.com/operations/load-testing/token-exchange/

[27] E. Rescorla, "Keying Material Exporters for Transport Layer Security (TLS)," RFC 5705, Mar. 2010 (updated by RFC 9846). https://www.rfc-editor.org/rfc/rfc5705.html

[28] G. Tene, "How NOT to Measure Latency," Strange Loop, Sep. 2015. https://www.infoq.com/presentations/latency-response-time/ Tooling: `wrk2` https://github.com/giltene/wrk2 and HdrHistogram https://github.com/HdrHistogram/HdrHistogram

[29] Spring Security, `DPoPProofJwtDecoderFactory.java`, tag 6.5.5. https://github.com/spring-projects/spring-security/blob/6.5.5/oauth2/oauth2-jose/src/main/java/org/springframework/security/oauth2/jwt/DPoPProofJwtDecoderFactory.java

[30] Spring Security issue #18120, "DPoP JtiClaimValidator does not prevent replay attack on multi-node setup," spring-projects/spring-security, 2025. https://github.com/spring-projects/spring-security/issues/18120

[31] CVE-2024-10039, "Keycloak mTLS Authentication Bypass via Reverse Proxy TLS Termination," advisory GHSA-93ww-43rr-79v3, November 25, 2024. https://github.com/advisories/GHSA-93ww-43rr-79v3

[32] Keycloak issue #48428, "Keycloak issues tokens via implicit and hybrid flow when DPoP is enforced for the client," April 23, 2026, version 26.5.3. https://github.com/keycloak/keycloak/issues/48428

[33] Keycloak issue #38333, "When calling the user info endpoint, the DPoP is not bound to the access token," closed 2025-03-31, release/26.2.0. https://github.com/keycloak/keycloak/issues/38333

[34] MCP Authorization Extensions repository, accessed 27 July 2026. https://github.com/modelcontextprotocol/ext-auth

[35] J. Salowey, Y. Rosomakho, and H. Tschofenig, "Workload Identity in a Multi System Environment (WIMSE) Architecture," Internet-Draft draft-ietf-wimse-arch-07, Mar. 2, 2026. https://datatracker.ietf.org/doc/html/draft-ietf-wimse-arch-07

[36] P. Hosseyni, R. Küsters, and T. Würtele, "Audience Injection Attacks: A New Class of Attacks on Web-Based Authorization and Authentication Standards," IACR Cryptology ePrint Archive 2025/629, Apr. 2025; in *Proc. IEEE Symposium on Security and Privacy (S&P)*, 2026. https://eprint.iacr.org/2025/629

[37] S. E. Mancinas Fontalvo, *dpop-replay-threshold*: artifact repository for the bounded-cache DPoP replay threshold, 2026. Apache License 2.0. https://github.com/sergioemancinas/dpop-replay-threshold

[38] Keycloak issue #39042, "[DPoP] Implementing DPoP nonce," open and unmilestoned since Apr. 17, 2025. https://github.com/keycloak/keycloak/issues/39042

[39] J. DeCock, "DPoP security for .NET APIs with JwtBearer extensions," Duende Software, Feb. 2, 2026. https://duendesoftware.com/blog/20260202-dpop-security-for-dotnet-apis-with-jwtbearer-extensions-v1

[40] T. Lodderstedt, J. Bradley, A. Labunets, and D. Fett, "Best Current Practice for OAuth 2.0 Security," RFC 9700 (BCP 240), Jan. 2025. https://www.rfc-editor.org/info/rfc9700

[41] T. Würtele, P. Hosseyni, K. Luo, and A. Fung, "Updates to OAuth 2.0 Security Best Current Practice," Internet-Draft draft-ietf-oauth-security-topics-update-03, Jul. 6, 2026. https://datatracker.ietf.org/doc/draft-ietf-oauth-security-topics-update/03/

[42] Spring Security issue #17107, "Implement internal cache in JtiClaimValidator," milestone 6.5.0, May 14, 2025. https://github.com/spring-projects/spring-security/issues/17107

[43] ConductorOne, *DPoP*: a Go implementation of OAuth 2.0 DPoP (RFC 9449), 2025. In-memory `jti` store is TTL-based with no capacity bound (`pkg/dpop/jti.go`). https://github.com/ConductorOne/DPoP

[44] Curity, *Mutual TLS API Example*: RFC 8705 certificate-bound access tokens with reverse-proxy TLS termination and `cnf` thumbprint verification. https://github.com/curityio/mutual-tls-api-example

[45] OpenID Foundation, "FAPI 2.0 Attacker Model," Final Specification, Feb. 2025. https://openid.net/specs/fapi-attacker-model-2_0-final.html

[46] J. Romano, J. D. Kromrey, J. Coraggio, and J. Skowronek, "Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen’s d for evaluating group differences on the NSSE and other surveys?," in *Annual Meeting of the Florida Assoc. of Institutional Research*, 2006.