

LEXCAP: Transaction-Consistent State-Bound Capabilities for Auditable Legal-Resource Access

Nuha Mansour Alsaeed^{1,*}

¹ College of Business, Imam Mohammad Ibn Saud Islamic University, Riyadh, Saudi Arabia.

* Corresponding author. Nmsaeed@imamu.edu.sa

Received: 24 June 2026 Accepted: 9 September 2026 Published: 20 October 2026

DOI: 10.15849/ijasca.202 License: Open access under CC BY 4.0. Authors retain copyright.

Publisher: Al-Zaytoonah University of Jordan (ZUJ), Amman, Jordan.

Cite this article: N. M. Alsaeed, "LEXCAP: transaction-consistent state-bound capabilities for auditable legal-resource access," *Int. J. Advance Soft Compu. Appl.*, vol. 18, no. 3, pp. 499–510, 2026, doi: <https://doi.org/10.15849/ijasca.202>.

ABSTRACT

Digital legal proceedings require authorization decisions that remain valid while participant status, matter association, privilege, protective-order restrictions, and document classifications change. A signed bearer token can be authentic yet stale, while a matter-wide revocation epoch prevents stale use only by invalidating unrelated grants. This paper presents LEXCAP, a state-bound capability architecture that evaluates identity, association, privilege, restriction, policy, and nonce state under one optimistic serializable transaction. It derives an authorization-state fingerprint (ASF) from deterministic material projections, authorization-relevant versions, request binding, policy and dependency digests, and a one-time issuance nonce. The fingerprint is embedded in an Ed25519-signed capability, re-created before access, and linked to an encrypted, quorum-signed audit chain. We implement a controlled Python prototype with SHA-256, AES-256-GCM, four ledger validators, and a synthetic legal-document corpus containing 600 participants, 80 matters, 8,000 resources, and 20,120 state records. Across five fixed seeds, LEXCAP rejected all 1,500 capabilities affected by relevant post-issuance revocations while retaining all capabilities after administrative or unrelated same-matter changes. It detected every one of 1,250 issuance races, rejected every duplicate-nonce issuance, reconstructed all 1,500 audited decisions, and localized every injected state divergence. Median end-to-end issue-and-validation latency was 1.165 ms, with 2.001 ms at the 95th percentile and 639 request pairs/s on a five-vCPU host.

Keywords: access control; capability security; legal technology; authorization provenance; audit logging; duplicate-issuance prevention; tamper-evident audit logging; digital evidence.

1 Introduction

Digital legal proceedings exchange pleadings, exhibits, discovery productions, transcripts, expert reports, settlement communications, sealed filings, and court orders among organizations with different duties and conflicts. Access is consequently state dependent. A participant may be active in one matter but screened from another; an expert may have a bounded review window; a protective order may prohibit export; and a privilege decision may change after review, waiver, clawback, or judicial determination. A role alone does not capture these conditions.

Operational systems often separate identity, matter membership, policy evaluation, privilege review, restriction enforcement, token issuance, and audit logging. This decomposition creates two assurance problems. First, a request can observe incompatible record versions when updates race with authorization. Second, a previously issued bearer artifact can remain cryptographically valid after the state that justified it has changed. A later log may show that access was granted without proving which record versions and policy produced that decision.

Classical RBAC, ABAC, and usage-control models express increasingly rich policy conditions [1]–[4]. Capability systems and sender-constrained tokens attenuate authority and reduce token theft or replay [5]–[7]. Distributed authorization systems address consistency at large scale [11], and tamper-evident logs preserve forensic history [15], [16]. These mechanisms are complementary, but none alone binds a legal access artifact to a reconstructable, versioned authorization snapshot and then revalidates that snapshot with material-change precision.

This paper develops LEXCAP, a prototype implementation of state-bound capability authorization. The main contributions are: a normalized legal authorization model that distinguishes repository storage

versions from authorization-relevant versions, preventing administrative metadata changes from needlessly invalidating capabilities; an optimistic serializable issuance protocol that binds policy inputs, request context, a one-time nonce, and a deterministic state fingerprint before capability commitment; an Ed25519-signed capability and a privacy-reduced audit record containing pseudonymous identifiers, encrypted lineage fields, and a three-of-four quorum-signed hash-chain commitment; and a reproducible controlled implementation and evaluation against a role cache, a signed ABAC bearer token, and a matter-wide epoch token under policy, state-change, race, replay, substitution, audit, and performance workloads.

The evaluation is deliberately scoped. It measures a synthetic policy service and a single-host permissioned-ledger emulator. It does not claim legal sufficiency, production blockchain finality, or field performance.

2 Related Work

2.1 Access-Control and Capability Models

RBAC assigns permissions through organizational roles [1], [2]. ABAC evaluates subject, object, action, and environmental attributes against policy [3]; UCON further incorporates mutable attributes, obligations, and ongoing control [4]. These models are appropriate for matter roles, resource classifications, temporal windows, and court-imposed restrictions, but the model definition does not require a decision to carry the exact versions of its inputs.

Macaroons provide contextual caveats and decentralized attenuation through chained message authentication codes [5]. JWT defines a compact signed claims format [6], while DPoP sender-constrains OAuth tokens and addresses replay of stolen tokens [7]. LEXCAP addresses a different replay surface: duplicate capability issuance with the same transaction nonce, as well as continued use after authorization state changes. Its state fingerprint can coexist with caveats or proof-of-possession. NIST zero-trust guidance emphasizes per-resource decisions and continual evaluation rather than implicit long-lived trust [8]. W3C decentralized identifiers and verifiable credentials provide interoperable identity and signed claims [9], [10]. Such credentials can populate LEXCAP's identity and association records, but credential authenticity does not establish current matter privilege or restriction state.

2.2 Consistency, Provenance, and Audit

Zanzibar demonstrates a globally consistent relationship-based authorization service with causal consistency requirements [11]. Database literature shows why weak isolation can expose inconsistent state and how serializable snapshot techniques prevent anomalies [12], [13]. LEXCAP applies an optimistic compare-and-commit pattern to the smaller transaction that spans the records used for one legal-resource decision.

Data provenance records why and where a result was produced [14]. Secure audit-log research provides forward integrity and tamper-evident history [15], [16]. Permissioned ledgers such as Hyperledger Fabric provide endorsement and append-only coordination across organizations [17]. LEXCAP combines these ideas by linking each capability to an archived authorization projection and a quorum-signed event chain. Full policy evaluation remains off-chain to limit disclosure and computational cost.

2.3 Research Gap

Prior work provides expressive policies, attenuated tokens, consistent relationship checks, and verifiable logs. The unresolved integration problem is to issue a low-latency capability whose continued validity depends on the same material identity, association, privilege, restriction, and policy state that justified issuance, while preserving sufficient lineage to reconstruct the decision. LEXCAP targets this integration for legal-document workflows.

3 System Model and Requirements

3.1 Entities and State

Let an access request be

$$q = (p, r, c, a, t_q), \quad (1)$$

where participant p requests action a on resource r in legal context c at time t_q . The authorization processor reads four record classes:

$$S_q = (I_p, A_{p,c}, P_{r,c}, R_{r,c}), \quad (2)$$

Table 1. Authorization-state records and material fields.

Record	Material projection examples	Primary effect
Identity	active status, credential expiry, organization, jurisdiction, clearance, MFA result	actor validity and conflict checks
Association	matter, role, active/screened state, assignment interval, scope	participant-context binding
Privilege	classification, permitted roles, challenge, waiver, clawback state	disclosure entitlement
Restriction	allowed actions, export flag, court hold, seal, approval, blocked organizations, interval	procedural override
Nonce	digest, request binding, unused/consumed state	duplicate-issuance prevention

representing identity, matter association, privilege, and restriction state. A versioned policy Π^{v_π} evaluates the records. A nonce record N_η binds one issuance attempt.

Each source record X has a storage version $sv(X)$, an authorization-relevant version $av(X)$, and a material projection $M_{v_\pi}(X)$. Any repository edit increments $sv(X)$; only a change to a field consumed by policy version v_π increments $av(X)$. For example, changing a review comment is administrative, whereas revoking association status or adding an export prohibition is material. The full storage versions remain in encrypted lineage for audit.

3.2 Adversary and Trust Assumptions

The adversary may present stale or modified capabilities, substitute a resource or context, repeat an issuance nonce, race a revocation against issuance, roll a record back to an earlier value, alter audit events, or compromise one of four audit validators. The adversary cannot break SHA-256, Ed25519, or AES-GCM; extract issuer or encryption keys from the trusted authorization service; or control two or more validators. Repositories may be mutable but must provide monotonic authorization versions and historical records for auditable decisions. Denial of service, malicious policy authors, compromised issuer keys, and the legal correctness of human classifications are outside the cryptographic guarantee.

3.3 Security and Functional Requirements

The design requires: (R1) transaction-consistent policy inputs; (R2) request, resource, context, action, and time binding; (R3) rejection after a relevant state or policy change; (R4) no invalidation after a nonmaterial or unrelated change; (R5) atomic nonce consumption; (R6) capability unforgeability; (R7) reconstructable decision lineage; (R8) tamper-evident audit linkage; and (R9) minimized public audit metadata.

4 LEXCAP Architecture

Figure 1 separates the issuance path, validation path, repositories, and audit plane. A state adapter converts heterogeneous case-management records into deterministic material projections. The policy engine evaluates a snapshot. The fingerprint and capability services produce the state-bound artifact. The audit plane stores public commitments and an encrypted envelope rather than plaintext matter metadata.

4.1 Deterministic State Fingerprint

The prototype uses a JSON Canonicalization Scheme-compatible subset with sorted object keys and no floating-point fields [18]. For each record X ,

$$d_X = H\left(\text{type}(X) \parallel \text{key}(X) \parallel \text{av}(X) \parallel \text{Canon}\left(M_{v_\pi}(X)\right)\right). \quad (3)$$

Let σ_c denote the fixed digest order and g the digest of the dependency graph linking identity, association, privilege, restriction, and decision nodes. With nonce digest $h_n = H(n)$ and request binding $b_q = \text{Canon}(p, r, c, a)$,

$$D_q = H(d_I \parallel d_A \parallel d_P \parallel d_R \parallel \sigma_c), \quad (4)$$

$$\text{ASF}_q = H(D_q \parallel g \parallel b_q \parallel h_n \parallel t_i \parallel v_\pi \parallel \text{grant}), \quad (5)$$

where t_i is issuance time. Including $\text{av}(X)$ detects a rollback sequence even when record values later return to their original representation. Including v_π invalidates a capability when policy semantics change.

Schema and dependency versioning. The current dependency graph, canonicalization profile, material-field mapping, and policy-input schema are treated as versioned authorization artifacts. Their identifiers and digests are obtained from the trusted policy/schema registry used by the validator. A change that can alter the interpretation or material projection of a policy input advances the corresponding schema or policy version and therefore changes the reconstructed ASF, invalidating previously issued capabilities unless an explicitly reviewed compatibility rule is defined. Schema-version changes are themselves recorded in the audit lineage so that historical reconstruction uses the exact definitions effective at issuance.

4.2 Atomic Issuance and Nonce Consumption

Algorithm 1 is implemented as optimistic snapshot evaluation followed by a compare-and-commit critical section. The processor reads S_q and the authorization-version vector V_q , evaluates Π^{v_π} , and computes ASF_q outside the commit lock. It then commits only if the nonce remains unused and current versions still equal V_q . A mismatch returns a conflict; retrying against current state either produces a new fingerprint or denies the request.

Partial-failure and recovery semantics. In the single-process prototype, nonce consumption, lineage insertion, capability construction/signing, and audit-event append are staged under the same commit lock. The capability is not released to the caller until the audit event has been appended successfully. If an exception occurs after the nonce is tentatively consumed but before audit commitment, the prototype restores the nonce to unused state, removes any staged lineage entry, and discards the signed capability object. A retry therefore begins from the pre-commit state. This is application-level atomicity rather than a distributed atomic-commit proof: a production implementation spanning a database and a networked ledger would require a durable transaction protocol, transactional outbox, or equivalent recovery record so that crash recovery cannot leave a consumed nonce or externally usable capability without a corresponding committed audit event.

4.3 Capability and Audit Record

The capability body is

$$C_q = (p, r, c, a, \text{ASF}_q, h_n, [t_s, t_e], v_\pi, \lambda_q, s_q, \sigma_c), \quad (6)$$

where λ_q references archived lineage and s_q is the expected audit sequence. The issuer returns

$$\text{CBCap}_q = (C_q, \text{Sign}_{sk_I}(\text{Canon}(C_q)), kid_I). \quad (7)$$

The audit event publicly exposes a capability hash, ASF_q , nonce digest, HMAC-pseudonymous identifiers, lineage commitment, sequence, and predecessor hash. Participant, resource, context, authorization versions, source storage versions, and the decision are encrypted under AES-256-GCM

[20]. Four Ed25519 validators sign the event digest; three signatures form a quorum [21]. The event hash chains to its predecessor.

4.4 Validation and Divergence Localization

The enforcement point verifies issuer signature, validity interval, participant/resource/context/action binding, consumed nonce, capability hash, and audit sequence. It then reads current records and recomputes ASF'_q using the capability's nonce digest and issuance time:

$$Accept(q) = Verify(CBCap_q) \wedge t \in [t_s, t_e] \wedge ASF'_q = ASF_q. \quad (8)$$

On mismatch, the validator compares the four archived material digests with current digests and records the affected record classes. Exact equality is conservative but auditable. A production deployment may define signed compatibility predicates for monotonic changes, but the prototype does not.

5 Security Analysis

Scope of the security analysis. The statements below are conditional arguments under the adversary model and the stated cryptographic and trust assumptions; they are not a formal proof or formal verification. Cryptographic properties are derived from assumptions about SHA-256, Ed25519, AES-GCM, and key protection. Transaction and audit behaviors are properties of the implemented prototype and are tested only in the controlled experiments described in Sections 7 and 8. Monotonic authorization versions, correct materiality mappings, correct historical archives, and externally retained audit checkpoints remain operational assumptions unless a specific experiment exercises them. Accordingly, 0% or 100% results in the evaluation describe deterministic outcomes for the tested implementation and generator, not field attack probabilities or production-level guarantees.

Capability unforgeability. An adversary that changes a signed binding, fingerprint, expiration, lineage reference, or audit sequence must forge an Ed25519 signature, assuming issuer-key protection.

Stale-state rejection. If any fingerprint-contributing material projection, authorization version, dependency graph, or policy version changes, then $ASF'_q \neq ASF_q$ except with a SHA-256 collision. The validator therefore rejects a previously issued capability. Administrative metadata is excluded from M_{v_π} and does not alter $av(X)$, so it does not cause rejection.

Issuance-race safety. A revocation between snapshot evaluation and commit changes V_q . The final compare fails, no nonce is consumed, and no capability is emitted. If the revocation commits after capability issuance, later revalidation detects the new fingerprint. Thus the accepted access is serializable either before the update or after it, not across mixed versions.

Duplicate-issuance and substitution resistance. The serializable unused-to-consumed nonce transition permits at most one committed issuance per nonce. The signed capability and fingerprint include p, r, c, a , preventing cross-resource, cross-context, or cross-action substitution. Repeated presentation of the same valid capability within its validity interval remains permitted. The nonce mechanism prevents duplicate issuance using the same nonce, while request binding prevents reuse under an altered resource, context, or action.

Audit integrity and reconstruction. A modified event invalidates its digest or quorum signatures. Internal deletion or reordering breaks sequence and predecessor linkage. Given archived material projections, the auditor recomputes the historical fingerprint and compares it with both capability and ledger commitments. Tail truncation requires an external expected-height checkpoint, as in other append-only logs.

Tail-truncation checkpoints. A deployment should periodically create a signed checkpoint containing at least the ledger identifier, committed sequence height, terminal event hash, checkpoint time, and signer/key identifier. Independent clients, auditors, or a separate transparency service should retain or cross-publish these checkpoints. Verification compares a presented log tip with the latest trusted

checkpoint. Without an externally retained expected height and terminal hash, a cryptographically valid truncated prefix cannot be distinguished from a genuinely shorter log.

Privacy. Public events contain pseudonyms and commitments rather than plaintext matter identifiers. AES-GCM protects the lineage envelope, but event timing, frequency, event size, and stable pseudonyms remain observable. Key compromise, cross-dataset correlation, and access-pattern privacy require separate controls.

6 Prototype Implementation

The artifact implements LEXCAP in 1,300 lines of Python 3.13.5 using cryptography 46.0.4. SHA-256 follows the Secure Hash Standard [19]; Ed25519 signs capabilities and validator endorsements; AES-256-GCM encrypts audit envelopes. The state store is in memory. Each state bundle has a reentrant lock, and the issuance service uses optimistic version comparison plus a commit lock for nonce consumption, lineage insertion, capability signing, and ledger append.

The ledger is a local state-machine emulator, not Hyperledger Fabric or a networked consensus deployment. It contains four generated validator keys and requires three valid signatures.

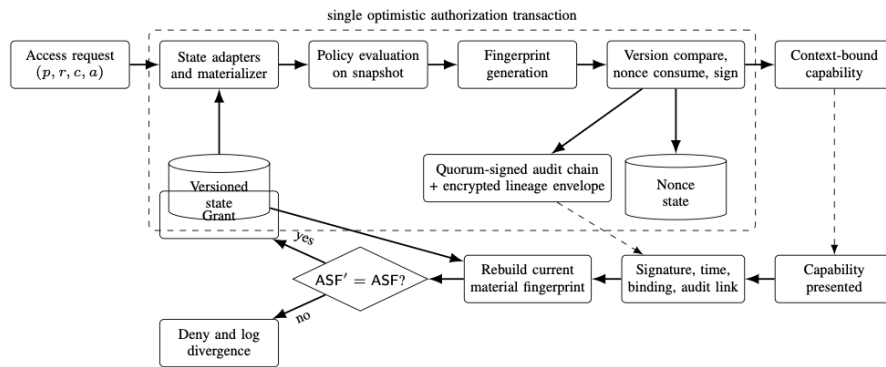


Fig. 1. LEXCAP architecture. Issuance snapshots material authorization state, evaluates policy, compares versions, consumes a nonce, signs the capability, and commits an audit event. Validation reconstructs the current fingerprint before resource access.

The 3-of-4 construction therefore demonstrates quorum-signature and hash-chain verification logic only. It does not evaluate independent validator crashes, network partitions, equivocation, Byzantine coordination, distributed ordering, consensus finality, durable recovery, or cross-organization key management. Claims concerning distributed-ledger behavior are limited to this single-host emulator.

Algorithm 1. State-bound capability issuance

1. Generate or receive 128-bit nonce n bound to q .
2. Snapshot S_q , V_q , v_π , and dependency schema.
3. If $Eval(\Pi^{v_\pi}, S_q, q) = deny$, return deny.
4. Compute ASF_q and prepare lineage projection L_q .
5. Enter commit section. Reject if N_n is consumed.
6. Compare current (V_q, v_π) with snapshot; on mismatch return conflict.
7. Mark N_n consumed; archive L_q ; construct and sign $CBCap_q$.
8. Append quorum-signed audit event and return $CBCap_q$.

This isolates application-level cryptographic and serialization costs while excluding network propagation, disk durability, ordering-service delay, and recovery.

6.1 Synthetic Legal-Document Corpus

Each fixed-seed corpus contains 600 participants, 80 matters, 8,000 resources, 3,520 participant-matter associations, and 20,120 total state records. Roles are attorney, paralegal, expert, clerk, judge, auditor, and custodian. Resources include pleadings, exhibits, discovery, transcripts, expert reports, and court orders with public, confidential, privileged, sealed, or expert-only classifications. Policies combine active identity, credential validity, MFA, association status, screening, temporal windows, role-action permissions, privilege status, clearance, export restrictions, court holds, seals, supervisory approval, and blocked organizations.

6.2 Baselines

The comparison includes: (1) RBAC role cache: a signed, context- and action-bound cached role grant that omits resource binding and does not evaluate privilege or restriction state after issuance; (2) ABAC bearer: the full prototype policy evaluated at issuance, with a signed participant/resource/context/action token but no current-state revalidation; (3) matter-epoch token: the ABAC bearer plus a context-wide epoch checked at access; every matter update advances the epoch; (4) LEXCAP: record-scoped material fingerprint, nonce transaction, audit linkage, and current-state reconstruction. These are concrete experimental baselines, not claims about every possible RBAC, ABAC, JWT, or epoch implementation.

These experiments are controlled mechanism comparisons, not competitive benchmarks against the strongest contemporary authorization architectures. Several outcomes follow directly from the selected baseline mechanisms: for example, a bearer token without current-state revalidation cannot observe a later revocation, and a resource-agnostic role cache cannot reject same-context resource substitution. The results therefore verify the implemented design differences and quantify their consequences under a common workload; they do not establish general superiority over RBAC, ABAC, JWT, epoch-based, or other state-aware authorization systems.

7 Experimental Methodology

The experiments address five questions. RQ1: Does record-scoped state binding prevent stale access without broad false invalidation? RQ2: Does the transaction prevent race-induced issuance, duplicate nonce use, tampering, and substitution? RQ3: Can an auditor reconstruct and localize decisions and detect log alteration? RQ4: What latency and storage overhead does the prototype add? RQ5: Which fingerprint components are necessary?

The current evaluation does not include a separate crash/failure-injection campaign for failures after tentative nonce consumption, during lineage archival, during validator signing, or immediately before ledger append. The prototype rollback path described in Section 4.2 is therefore an implementation design property rather than an independently measured distributed-recovery result. Fault injection across durable multi-node components is left for future work.

Five independent fixed seeds (11, 23, 37, 53, and 71) were executed. Each seed includes 8,000 static policy decisions; 300 trials for each state-change scenario and method; 250 race trials and 250 replay/substitution trials per method; 300 reconstruction/localization trials; 80 event and chain tamper trials; 500 issue-validation performance pairs per method; and 300 ablation trials. Reported means and population standard deviations are across seeds. Security outcomes with zero observed failures are deterministic for the tested implementation and generator; they are not estimates of field attack probability. The host exposed five Intel Xeon E5-2673 v4 virtual CPUs at 2.30 GHz and 5.9 GiB RAM. Measurements exclude network and persistent-storage latency. Performance therefore represents a single-host computational floor.

Because network transport, persistent database I/O, distributed validator communication, ordering, consensus/finality, and recovery are excluded, all reported latency and throughput values should be interpreted as a single-host computational lower bound rather than production end-to-end performance.

8 Results

8.1 Policy Fidelity and Post-Issuance State Changes

The coarse role cache achieved $69.78\% \pm 0.45\%$ decision accuracy and a $48.40\% \pm 1.04\%$ false-grant rate among ground-truth denials because it ignored privilege and procedural restrictions. The three full-policy methods matched the policy oracle by construction.

This agreement is an implementation-consistency check rather than an independent validation of policy correctness. The oracle and the full-policy implementations encode the same synthetic policy semantics, so 100% agreement does not show that LEXCAP captures the substantive authorization law, privilege doctrine, protective-order practice, or procedural requirements of any jurisdiction.

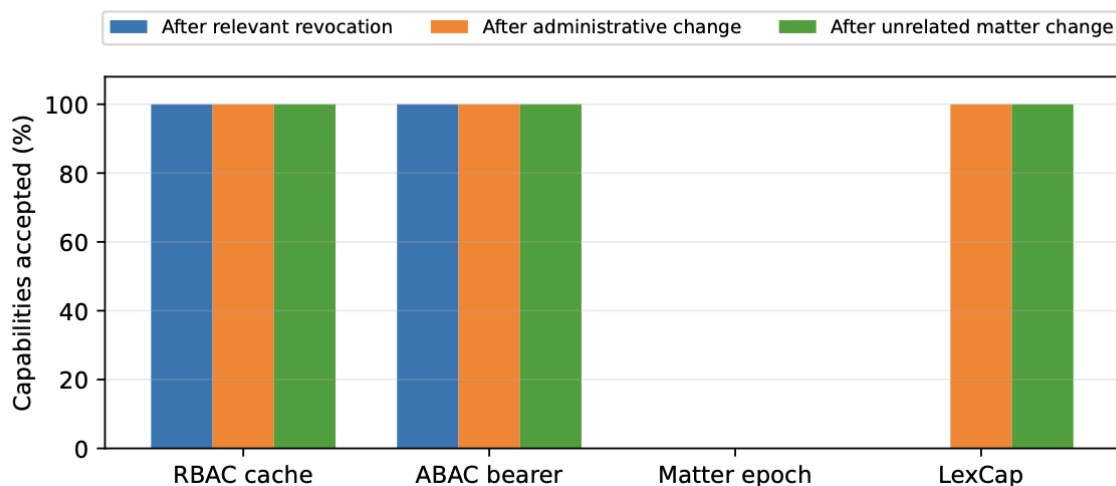


Figure 2. Capability acceptance after three update classes, averaged across five seeds. Correct behavior is 0% acceptance after a relevant revocation and 100% after administrative or unrelated changes. Source: Created by the author.

Table 2. Capability acceptance after state changes (%). Note: Percentages are deterministic controlled-trial outcomes under the implemented generator and threat model, not estimates of real-world attack probabilities.

Method	Relevant revocation	Administrative metadata	Unrelated same matter
RBAC role cache	100	100	100
ABAC bearer	100	100	100
Matter epoch	0	0	0
LEXCAP	0	100	100

Figure 2 and Table 2 show post-issuance behavior. RBAC and ABAC retained valid artifacts after harmless changes but also accepted every capability after a relevant revocation. The matter epoch blocked all relevant revocations, but invalidated every capability after either an administrative edit to a relevant record or an unrelated update elsewhere in the same matter. LEXCAP blocked all relevant revocations and retained all harmless or unrelated grants.

8.2 Race, Replay, Tampering, and Substitution

A revocation was injected after evaluation but before token commitment. RBAC, ABAC, and the matter-epoch method emitted stale capabilities in all 1,250 trials per method. RBAC and ABAC also accepted all of them. The epoch validator rejected stale access, but the artifact had already been emitted. LEXCAP detected a version conflict in every trial, emitted no capability, and denied on retry. Across 1,250 duplicate-nonce trials, only LEXCAP rejected every second issuance. All four methods rejected direct token modification because all tokens were signed. The resource-agnostic RBAC cache accepted every same-context resource substitution; the three resource-bound methods rejected all substitutions.

Table 3. Adversarial workflow results (% of trials). Note: Percentages are deterministic controlled-trial outcomes under the implemented generator and threat model, not estimates of real-world attack probabilities.

Method	Stale emitted	Stale accepted	Duplicate nonce	Cross-resource
RBAC role cache	100	100	100	100
ABAC bearer	100	100	100	0
Matter epoch	100	0	100	0
LEXCAP	0	0	0	0

8.3 Audit Reconstruction and Tamper Detection

LEXCAP reconstructed all 1,500 tested decisions from archived projections and matched the embedded fingerprint. After one material record was altered, it localized the changed record class in all 1,500 trials. It detected all 400 payload/digest/quorum-signature modifications and all 400 internal deletion or reordering attempts. A plain decision log containing only actor, resource, action, and outcome could not reconstruct exact state or detect a semantically valid edit without an external trusted copy. These perfect controlled outcomes validate code paths rather than proving resistance to every operational failure.

8.4 Latency and Storage Overhead

Figure 3 and Table 4 report issue-plus-validation latency. LEXCAP's median was 1.165 ms and its 95th percentile was 2.001 ms, compared with 0.292 ms and 0.530 ms for the ABAC bearer. The absolute median increment was 0.873 ms. The local prototype sustained 639 ± 45 issue-validation pairs/s, versus $1,899 \pm 128$ for ABAC. The 99th-percentile LEXCAP latency was 5.112 ms; the wider tail came primarily from four validator signatures, AES-GCM envelope generation, and repeated canonicalization.

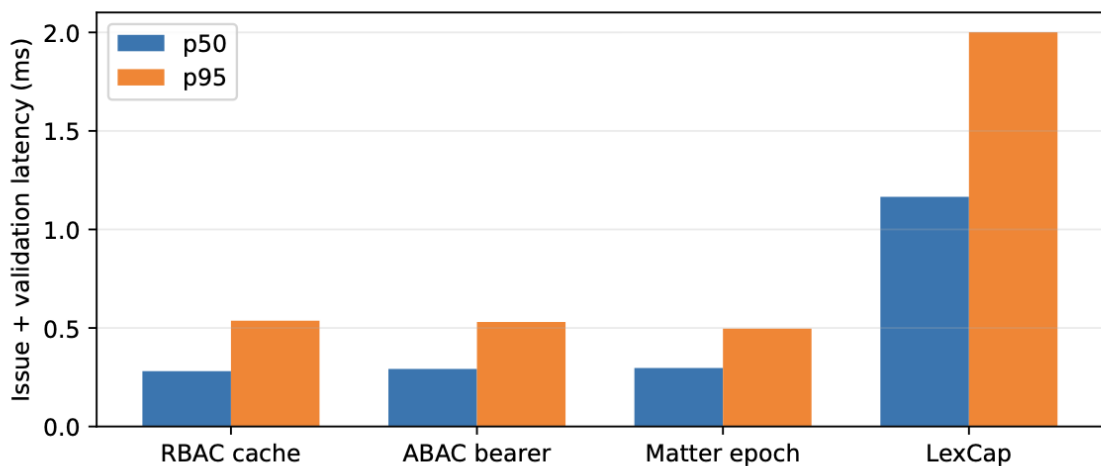


Figure 3. Single-host issue-plus-validation latency. Bars show the mean per-seed p50 and p95 across 500 pairs per method and seed. Source: Created by the author.

Table 4. Issue + validation performance. Note: p50, p95, and p99 are means of the corresponding per-seed percentiles across the five fixed seeds; pairs/s is the mean throughput across seeds. Measurements exclude network and persistent-storage costs.

Method	p50 (ms)	p95 (ms)	p99 (ms)	Pairs/s mean
RBAC role cache	0.281	0.537	0.924	1,939
ABAC bearer	0.292	0.530	0.932	1,899
Matter epoch	0.297	0.497	0.980	1,883
LEXCAP	1.165	2.001	5.112	639

A LEXCAP capability averaged 657 bytes. Its quorum-signed, encrypted ledger event averaged 1,530 bytes, and the off-chain lineage projection averaged 1,412 bytes. The ABAC token and plain audit record averaged 381 and 133 bytes, respectively. Thus LEXCAP adds approximately 1.67 KiB to the online capability-plus-ledger record and 1.38 KiB to the protected lineage archive per grant. Production batching or Merkle aggregation could reduce endorsement overhead but was not implemented.

8.5 Fingerprint Ablation

Table 5 isolates three design choices. Hashing the entire source record invalidated every capability after an administrative-only edit. Omitting authorization versions failed to detect every value-rollback sequence in which a record changed and returned to its original value. Omitting the policy digest failed to detect every policy-version change. The material projection with authorization versions and policy digest avoided all three failures.

Table 5. Fingerprint ablation results (%).

Variant	Admin. false invalidation	Rollback detected	Policy change detected
LEXCAP projection	0	100	100
Full source record	100	100	100
Value only	0	0	100
No policy digest	0	100	0

9 Discussion

The results expose a three-way design trade-off. A bearer token preserves availability but cannot notice state changes. A matter-wide epoch notices changes but over-invalidates because its dependency scope is coarse. A record-scoped fingerprint can provide both safety and precision if materiality and version semantics are governed correctly.

LEXCAP does not require every policy rule to execute on-chain. The ledger is an accountability substrate: it binds the capability hash, fingerprint, nonce, lineage commitment, and event order. This separation limits public metadata and avoids smart-contract duplication of a complex legal policy engine. A real consortium could replace the emulator with a permissioned ledger, a transparency log, or a replicated append-only database, provided it preserves authenticated ordering and checkpoints. The authorization-relevant version is operationally critical. If an adapter fails to classify a legally material field, a stale capability may remain valid. Conversely, an overly broad projection recreates epoch-style false invalidation. Deployments therefore need schema review, signed materiality profiles, migration tests, and dual-control approval for policy-version changes. NIST access-control policy testing techniques provide a useful foundation for such validation [22].

Governance of materiality definitions. Authorization-relevant fields, material projections, dependency definitions, canonicalization rules, and policy-input schemas should be controlled as signed, versioned security-policy artifacts. Only designated policy/schema maintainers should be permitted to change them, with dual-control review for material changes. Each approved change should receive a new schema/materiality version and be recorded in the audit lineage with the approving principal(s), effective time, and prior-version reference. An under-inclusive mapping can leave a capability valid after a legally material change; an over-inclusive mapping can recreate broad epoch-style invalidation. Migration tests should therefore verify both false-valid and false-invalidations cases before a new mapping becomes effective.

10 Limitations, Privacy, and Legal Governance

The corpus and policy are synthetic. They represent plausible legal-document conditions but not the rules, terminology, or procedural requirements of every jurisdiction. The prototype uses one process, memory-resident state, and a local ledger emulator; it excludes network faults, disk durability, validator

equivocation, cross-organization key management, ledger finality, disaster recovery, and sustained multi-node throughput. The RBAC baseline is intentionally coarse and should not be interpreted as a general result about RBAC systems.

The controlled generator also means that perfect rejection or detection percentages validate exercised code paths rather than estimate operational attack probability. In addition, the study does not experimentally validate crash recovery across separate durable stores or distributed validators; partial-failure handling is tested only implicitly through the single-process commit design.

The security argument assumes a trusted issuer, monotonic authorization versions, correct historical archives, collision-resistant hashing, uncompromised keys, and at most one faulty validator. Tail truncation requires clients or external auditors to retain signed height checkpoints. AES-GCM encrypted envelopes still reveal event timing, frequency, stable pseudonyms, and approximate size. Production deployments should rotate or context-scope pseudonym keys, apply retention limits, separate duties, protect keys in HSMs, and assess whether commitment retention conflicts with sealing, expungement, or deletion obligations.

Automated denial can affect participation and due process. LEXCAP should record the reason and divergence class, support authorized override and appeal workflows, and avoid embedding substantive legal judgments in opaque code. The architecture supplies technical provenance; it does not determine privilege, waive confidentiality, establish admissibility, or provide legal advice.

11 Conclusion

Legal-document access sits at an awkward intersection: authorization decisions must be fast enough for interactive workflows, yet the state that justifies them—participant status, privilege determinations, protective-order restrictions—can change at any moment after a token is issued. This paper presented LEXCAP, an architecture that addresses this tension by binding each capability to a deterministic fingerprint of the exact material state, authorization-relevant versions, policy semantics, request context, and one-time issuance nonce that produced the grant, and by anchoring that decision to a quorum-signed, encrypted audit chain.

The controlled evaluation showed, within the controlled prototype and selected baselines, that this record-scoped approach combines stale-state rejection with finer-grained invalidation than the tested bearer-token and matter-epoch mechanisms. Across five seeded runs, LEXCAP rejected all 1,500 capabilities affected by relevant post-issuance revocations while preserving every grant untouched by administrative edits or unrelated same-matter changes, detected all 1,250 issuance races and duplicate-nonce attempts, and reconstructed and localized every audited decision. The fingerprint ablation confirmed that material projections, authorization versions, and the policy digest are each necessary: removing any one reintroduced false invalidation, undetected rollbacks, or undetected policy changes. In this single-host prototype, these observed properties added a median 0.873 ms over the signed ABAC bearer and roughly 1.67 KiB of online record per grant. Whether that overhead is acceptable in operational legal workflows requires multi-node testing that includes database, network, ordering, durability, and recovery costs.

These results are bounded by the evaluation's scope. They reflect a single-host prototype with a synthetic corpus, an in-memory state store, and an emulated ledger, and they validate implementation behavior rather than field-scale security or legal sufficiency. The natural next steps are a multi-organization deployment over a production database and permissioned ledger, adapters for real case-management schemas with governed materiality profiles, fault and equivocation injection, a formal privacy review of the audit plane, and evaluation against representative jurisdictional workflows. More broadly, the work suggests that transaction-consistent state binding is a practical middle path between the availability of bearer credentials and the safety of aggressive revocation, wherever authorization depends on volatile, versioned state.

DECLARATIONS

Funding. This research received no external funding.

Conflicts of interest. The author declares that they have no competing interests.

Ethics approval Not applicable. This study did not involve human participants, animals, or any activities requiring ethical approval.

Informed consent. Not applicable. This study did not involve human participants or identifiable personal data.

Data availability. No datasets or source code are publicly available in connection with this study.

Author contributions. The sole author was responsible for the conceptualization, methodology, software development, investigation, analysis, validation, and writing of the manuscript. The author reviewed and approved the final version of the manuscript.

Use of artificial intelligence. Artificial intelligence tools were used solely to improve the language, grammar, readability, and formatting of the manuscript. All scientific content, methodology, analysis, interpretation of results, and conclusions were conceived, verified, and approved by the author, who takes full responsibility for the content of this manuscript.

REFERENCES

- [1] D. F. Ferraiolo and D. R. Kuhn, "Role-based access controls," in Proc. 15th National Computer Security Conf., 1992, pp. 554–563.
- [2] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-based access control models," Computer, vol. 29, no. 2, pp. 38–47, 1996.
- [3] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, Guide to Attribute Based Access Control (ABAC) Definition and Considerations, NIST SP 800-162, updated 2019.
- [4] J. Park and R. Sandhu, "The UCONABC usage control model," ACM Trans. Inf. Syst. Secur., vol. 7, no. 1, pp. 128–174, 2004.
- [5] A. Birgisson, J. G. Politz, U. Erlingsson, A. Taly, M. Vrable, and M. Lenczner, "Macaroons: Cookies with contextual caveats for decentralized authorization in the cloud," in Proc. NDSS, 2014.
- [6] M. Jones, J. Bradley, and N. Sakimura, "JSON Web Token (JWT)," IETF RFC 7519, May 2015.
- [7] D. Fett, B. Campbell, J. Bradley, T. Lodderstedt, M. Jones, and D. Waite, "OAuth 2.0 Demonstrating Proof of Possession (DPoP)," IETF RFC 9449, Sep. 2023.
- [8] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, Zero Trust Architecture, NIST SP 800-207, 2020.
- [9] W3C, "Decentralized Identifiers (DIDs) v1.0," W3C Recommendation, 2022.
- [10] W3C, "Verifiable Credentials Data Model v2.0," W3C Recommendation, 15 May 2025.
- [11] R. Pang et al., "Zanzibar: Google's consistent, global authorization system," in Proc. USENIX ATC, 2019, pp. 33–46.
- [12] H. Berenson, P. Bernstein, J. Gray, J. Melton, E. O'Neil, and P. O'Neil, "A critique of ANSI SQL isolation levels," in Proc. ACM SIGMOD, 1995, pp. 1–10.
- [13] M. J. Cahill, U. Röhm, and A. D. Fekete, "Serializable isolation for snapshot databases," in Proc. ACM SIGMOD, 2008, pp. 729–738.
- [14] P. Buneman, S. Khanna, and W.-C. Tan, "Why and where: A characterization of data provenance," in Proc. ICDT, 2001, pp. 316–330.
- [15] B. Schneier and J. Kelsey, "Secure audit logs to support computer forensics," ACM Trans. Inf. Syst. Secur., vol. 2, no. 2, pp. 159–176, 1999.
- [16] S. A. Crosby and D. S. Wallach, "Efficient data structures for tamper-evident logging," in Proc. USENIX Security, 2009, pp. 317–334.
- [17] E. Androulaki et al., "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in Proc. EuroSys, 2018, Art. no. 30, pp. 1–15.
- [18] A. Rundgren, B. Jordan, and S. Erdtman, "JSON Canonicalization Scheme (JCS)," IETF RFC 8785, June 2020.
- [19] National Institute of Standards and Technology, Secure Hash Standard (SHS), FIPS PUB 180-4, updated 2015.
- [20] M. Dworkin, Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC, NIST SP 800-38D, 2007.
- [21] S. Josefsson and I. Liusvaara, "Edwards-Curve Digital Signature Algorithm (EdDSA)," IETF RFC 8032, Jan. 2017.
- [22] V. C. Hu et al., Verification and Test Methods for Access Control Policies/Models, NIST SP 800-192, 2017.