Database-Level End User Authorization (DB-EUA)

Saurav Bhattacharya, Gaurav Deshmukh, Ankush Rastogi, Dewank Pant, Durga Krishnamoorthy, Manan Wason, Madhavi Najana, Uttam Kumar

Abstract Application servers are traditionally the policy enforcement point for databases. In that model, the database cannot verify the end user's identity or intent for each operation; it can only trust whatever context the application supplies. This creates systemic exposure to server compromise, confused-deputy problems, and weak provenance. DB-EUA moves verifiable authorization into the data path: every create/read/update/delete (CRUD) is accompanied by a user-authenticated, cryptographically verifiable token that the database (or a hardened database proxy) validates and binds to the session executing the SQL. The result is a tamper-evident, user-attributable audit trail and strong least-privilege enforcement at the DB layer—aligned with Zero Trust principles and regulatory accountability requirements. We present: A precise threat model and trust assumptions. A two-token architecture (server channel token + per-user operation token). A reference implementation blueprint for PostgreSQL Row-Level Security (RLS) with a wire-protocol proxy. Hardening guidance (key management, mTLS, channel binding, log hygiene). Compliance mappings (HIPAA, SOX, GDPR, CCPA) and a summary matrix. A practical vendor roadmap (DB engines, cloud DBs, gateway/proxy vendors, backend platforms).

***Index Terms*— Zero Trust; Row-Level Security (RLS); Cryptographically Verifiable Authorization; Verifiable Credentials (VC);**

I. INTRODUCTION

Most stacks authenticate users at the edge (OIDC/OAuth 2.0) and authorize requests in the application tier. The app then connects to the database with a pooled application identity (single database user/role) and performs CRUD "on behalf of" the end user. The database has no independent proof of the end user's identity or intended scope for each operation [1]. When the application is compromised (RCE, key theft, RBAC misconfig), it can impersonate users and write arbitrary data while the DB can only see "the app" [7][8][9].

Zero Trust's core message—continuously verify, least privilege, assume breach—applies at the data layer as much as at the network perimeter [13]. Provenance and accountability regulations (HIPAA, SOX) and data subject rights (GDPR/CCPA) increasingly require per-operation auditability and demonstrable consent/authority, which is infeasible if the DB never receives authenticated user context tied to the specific SQL statement [14][15][16][17].

To overcome these challenges we present the DB-EUA model. We validate that the DB-EUA model: Accurately verifies end-user identity and intent at the database layer, provides tamper-evident and user-attributable audit trails, enforces least-privilege and Zero Trust principles, resists confused-deputy attacks and application server compromises.

[1] The manuscript was submitted for review on 8/25/2025. It was supported by The New World Foundation. All authors are affiliated to The New World Foundation.

II. RELATED WORK

- OAuth 2.0 / OIDC. Excellent at user auth and delegation at the application boundary, but do not propagate signed per-operation claims to the database [1].
- PostgreSQL RLS. Expressive row-level policies driven by session variables; relies on the app to set truthful context [4].
- DIDs / Verifiable Credentials (VCs). Portable, cryptographic identity and attestations suitable for capability delegation; not commonly enforced at the SQL boundary today [2][3].
- UCAN / ZCAP-LD. User-controlled, delegable capabilities; promising for least-privilege chaining to resources (tables/rows) [5][6].
- Cloud DB IAM integrations. Azure SQL + AAD, AWS IAM DB Auth, GCP IAM for Cloud SQL improve who can connect, not who may change this row now [7][8][9].

III. THREAT MODEL

Goals. Prevent the application server (or an attacker within it) from executing arbitrary CRUD that appears as legitimate user actions; make all writes verifiably attributable to a specific user and scope.

Out of scope. Insider with DB superuser privileges; physical attacks on the DB host; cryptographic primitive failures (e.g., if JOSE/JWT are broken) [18][19].

Adversaries.

- Compromised app server: RCE/malware issuing SQL with forged user context.
- Token thief: Attempts replay of an intercepted token.
- Curious operator: Observes SQL logs to exfiltrate

user tokens/claims.

- Confused deputy: Microservice A misuses capabilities delegated to B.

Assumptions.

- User tokens are signed by the user's key or by a capability issuer under the user's control (e.g., VC/UCAN), and validated independently by the proxy/DB using a trusted key distribution (JWKS) [3][5][18][20][21].
- The proxy/DB has authenticated transport to clients/servers (mTLS) and maintains secure key material (HSM/KMS) [13].

IV. Architecture: Two Token, Verifiable CRUD

4.1 Token types

- Server Channel Token (SCT): Long-lived credential binding the proxy to the DB (e.g., mTLS client cert + DB role).
- User Operation Token (UOT): Short-lived, scoped token accompanying each CRUD (or batch) with claims describing who, what, where, until when [18][19].

UOT minimal claims (JWT/VC/UCAN):

sub	subject = end user identifier [18].
act or scope	capability: create/read/update/delete + resource selector, e.g., invoice:row_id=456) [5][6].
aud	intended verifier: DB/proxy [18]
nbf/exp	freshness window) [18
jti	nonce for replay defense) [18].
cnf	detached JWS w/ channel binding material to tie token use to the specific TLS session (prevents token replay on a different channel) [19][23]

Formats. UOT can be JOSE/JWT (JWS signed) [18][19], a VC carrying capabilities [3], or a UCAN capability proof [5]. Keys are distributed via JWKS [20] and optionally discovered with OIDC Discovery [21].

4.2 Execution flow

- Authenticate user (OIDC) and mint UOT from the user's key or a user-controlled capability issuer [1][3][5]. The identity provider (IdP) can also mint the token. The application proxy requests a UOT from the IdP after user login, and this UOT is then passed to the client to be attached to requests.
- Submit query → Proxy. Client forwards SQL without embedding the token in SQL. Instead, send UOT in a protected side-channel (protocol extension/header) to avoid log leakage.
- Verify UOT (sig, aud, window, jti uniqueness, chain of capabilities) and derive stable session claims (e.g., app.user_id, app.tenant_id, app.scopes).
- Bind claims to DB session (e.g., PostgreSQL SET LOCAL app.user_id = 'u123') and do not accept client-supplied GUCs.
- Enforce RLS on tables with policies referencing session claims; execute SQL [4].
- Audit: Persist a compact, immutable action record: (ts, user, op, resource, hash(sql), jti, token_hash); optionally hash-chain records per table for tamper-evidence.

4.3 Why a proxy?

Many engines cannot yet validate JOSE/VC artifacts natively. The proxy is the Policy Enforcement Point (PEP) and token firewall. Vendors can later push verification into the engine (native JWT/VC support) [4][7][8][9].

V. Reference Implementation (PostgreSQL)

5.1 Set immutable, proxy-owned GUCs:

```
-- Set by proxy, not by client
SET LOCAL app.user_id = 'user-123';
SET LOCAL app.tenant_id = 't-42';
SET LOCAL app.scopes = 'invoice:w,profile:r';
```

5.2 RLS policy examples

```
ALTER TABLE invoices ENABLE ROW LEVEL SECURITY;

CREATE POLICY tenant_isolation ON invoices
USING (tenant_id = current_setting('app.tenant_id',
true));

CREATE POLICY owner_can_write ON invoices
FOR UPDATE USING (
 tenant_id = current_setting('app.tenant_id', true) AND
 owner_id  = current_setting('app.user_id',   true)
);
```

RLS evaluates on every statement; policies compose with AND/OR as needed [4].

5.3 Auditing (append-only)

```
CREATE TABLE audit_log (
 ts           timestamptz NOT NULL DEFAULT now(),
 user_id      text        NOT NULL,
 op           text        NOT NULL,
 resource     text        NOT NULL,
 sql_hash     bytea       NOT NULL,
 jti          text        NOT NULL,
```

```
 token_hash    bytea       NOT NULL,
 prev_hash     bytea,
 this_hash     bytea       NOT NULL
);
```

VI. Hardening & Operational Guidance

- Key management. Store signing/verifying keys in HSM/KMS; rotate via JWKS kid and automated key rollover [20].
- mTLS everywhere. Client→Proxy and Proxy→DB channels with mutual TLS; prefer short-lived SCTs [13].
- Replay defense. Enforce jti uniqueness per audience and short exp; optionally require channel binding (e.g., tls-unique) in the token proof [19] [23].
- Scope minimalism. Map CRUD ops to least-privilege capability strings; deny queries that attempt resources not covered by scope [5][6].
- Statement hashing. Include canonicalized SQL hash in the audit record; optionally bind token proof to that hash to detect TOCTOU between authorization and execution.
- Multi-tenant isolation. Require tenant_id in every UOT; all policies AND with tenant constraint; validate cross-tenant queries.
- Observability. Emit structured events (user, op, resource, jti) to SIEM; correlate with app traces.

VII. Performance Considerations

Note. Actual latency/throughput impact depends on hardware, crypto algorithms, policy complexity, and dataset distribution; vendors should publish benchmarks alongside reference implementations.

- Token verification cost. JWS verification is dominated by public-key ops; amortize by caching valid kid keys (JWKS) and using short but not per-row tokens (one UOT per statement or per transactional unit) [18][19][20].
- RLS overhead. Policy predicates add per-row checks; design indexes aligned to predicate columns (tenant_id, owner_id) [4].
- Connection pooling. Ensure the proxy resets/overrides session GUCs between requests to avoid claim bleed across pooled connections.
- Batching. Encourage set-based operations with a single UOT covering a bounded resource set (e.g., row IDs list) to limit per-statement overhead while retaining auditability.

VIII. Compliance Alignment

This section translates regulatory obligations into concrete controls, mechanisms, and evidence artifacts that DB-EUA can produce.

9.1 HIPAA (Privacy & Security Rule)

Objectives. Ensure confidentiality, integrity, and availability of ePHI; restrict access to authorized individuals; maintain an accounting of disclosures [14].
Relevant provisions. Privacy Rule – Accounting of disclosures (45 CFR §164.528); Security Rule – Technical safeguards (45 CFR §164.312); Integrity (45 CFR §164.312(c)(1)).
DB-EUA Controls. Per-operation user attribution; JWS-backed intent + hash-chained audit; RLS for clinician-patient or role scoping [4][18][19].
Auditor Evidence. Audit extracts; RLS catalog & change history; JWKS rotation SOPs [20].
Residual Risks. DB superuser bypass; token replay within window—mitigated with dual-control, short exp, jti uniqueness, and channel binding [19][23].

9.2 SOX (Sections 302 & 404)

Objectives. Ensure accuracy of financial reporting and effectiveness of internal controls [15].
Controls. UOT-scoped writes; immutable provenance; least-privilege RLS [4][18][19].
Evidence. Traceability packs linking change→user→UOT→ticket; access reviews (effective scopes per role/team).
Residual Risks. Scope misconfiguration—address with policy linting/tests and four-eyes review.

9.3 GDPR

Objectives. Accountability (Art. 5), consent (Art. 7), rights (Arts. 15–20) [16].
Controls. Purpose-bound tokens; per-user logs; erasure/portability flows [16][18].
Evidence. Consent/provenance ledger; DSR runbooks.
Residual Risks. Over-collection of claims—mitigate via selective disclosure (SD-JWT) [10][25].

9.4 CCPA

Objectives. Rights to know, delete, opt-out, and non-discrimination [17].
Controls. Preference-aware policies; scoped delete tokens; user-attributed access logs.
Evidence. Fulfillment reports; allow/deny policy tests.

IX. Vendor Implementation Roadmap

10.1 Database Engine Vendors

- Phase 1 — Proxy/Extension. Ship an official wire-protocol proxy (or extension hook) that validates UOTs, binds trusted claims to sessions, and blocks client-set GUCs. Provide RLS helper library and migration recipes; publish log hygiene defaults (e.g., restrict SQL text exposure in statement stats) [4][24].
- Phase 2 — Native Verification. Add verification functions (verify_jws(), vc_verify()), trusted server-only session variables, and a policy DSL. Introduce capability-aligned indexes.
- Phase 3 — Privacy-Preserving Reads. Support SD-JWT and ZK proofs for read authorization [10] [25].

10.2 Cloud DB Providers

Managed JWKS & KMS integration [7][8][9][20]; attested verification proxies; reference architectures and throughput baselines with/without RLS.

10.3 Backend/Platform Vendors
Translate API scopes to UOTs; bind to SQL sessions; ship starter RLS policies; add policy testing harnesses in CI/CD; provide SIEM exports.

10.4 Security/Compliance Tooling Vendors
Policy linters and capability analyzers (UCAN/VC); audit compilers that assemble evidence packs for HIPAA/SOX/GDPR/CCPA.

KPIs. Coverage of write paths under DB-EUA; % tables with RLS; latency deltas; audit completeness (valid jti and token digest rate).

X. COMPLIANCE SUMMARY MATRIX

Regulation	Key Obligations	DB-EUA Mechanisms	Auditor Evidence	Residual Risks
HIPAA	Privacy Rule (access/accounting), Security Rule (technical safeguards, integrity)	UOT-attributed CRUD; RLS isolation; hash-chained audit	Audit extracts; RLS catalog; KMS/JWKS SOPs [14] [19][20]	DB superuser bypass; token replay window
SOX	§302 (certification), §404 (internal controls)	Capability-scoped writes; immutable provenance	Traceability packs; access reviews; policy test results [15] [4]	Scope misconfig; insider override
GDPR	Art. 5 (accountability), Art. 7 (consent), Arts. 15–20 (rights)	Purpose-bound tokens; per-user logs; erasure/portability flows	Consent/ provenance ledger; DSR runbooks [16][18]	Over-collection; linkage risk
CCPA	§1798.100(b) (notice), §1798.105 (delete), §1798.110 (know), §1798.120 (opt-out), §1798.125 (non-discrimination)	Preference-aware policies; scoped delete tokens	Fulfillment reports; allow/deny test logs [17]	Ambiguity in sale/share definitions

Additionally, To align with regulatory requirements such as NIST SP 800-207 (Zero Trust Architecture), ISO/IEC 27001:2022 controls, and GDPR/GLBA audit principles, the DB-EUA model must ensure:

- Strong Identity Binding – every CRUD operation is tied to a cryptographically verifiable, user-authenticated token.
- Least-Privilege Enforcement – tokens must specify permitted operations and data scope, preventing unauthorized escalation.
- Tamper-Evident Audit Trails – immutable logs must capture who accessed what data, when, and why, supporting accountability.
- Replay & Forgery Protection – tokens must expire and include cryptographic signatures to prevent misuse.
- Traceability for Audits – systems should generate user-attributable reports demonstrating compliance with regulations (e.g., SOX, GDPR, HIPAA).

By embedding these controls into DB-EUA, organizations achieve regulatory accountability, Zero Trust enforcement, and verifiable provenance of all database operations [26] [27]

XI. CONCLUSION

Database-Level End User Authorization (DB-EUA) re-centers trust at the data layer by binding every database operation to a cryptographically verifiable user identity and intent. Unlike traditional models that rely on the application tier to enforce access controls, DB-EUA provides tamper-evident auditability, least-privilege enforcement, and resilience against confused-deputy and server-compromise attacks. Our two-token architecture, reference PostgreSQL blueprint, and compliance mappings demonstrate that DB-EUA is both technically feasible and operationally valuable. While challenges remain around performance trade-offs, proxy trust, and vendor adoption, the model aligns with Zero Trust principles and modern regulatory demands. By embedding verifiable provenance into the SQL path, DB-EUA lays the foundation for accountable, privacy-preserving, and regulation-ready data systems—turning the database from a passive storage layer into an active enforcer of user-centric security.

REFERENCES

[1] Hardt, D. (2012). The OAuth 2.0 Authorization Framework. IETF RFC 6749. DOI: 10.17487/RFC6749.
[2] Sporny, M., Longley, D., Chadwick, D., et al. (2022). Decentralized Identifiers (DIDs) v1.0. W3C Recommendation.
[3] Sporny, M., Longley, D., et al. (2019). Verifiable Credentials Data Model 1.0. W3C Recommendation.
[4] PostgreSQL Global Development Group. Row Security Policies. PostgreSQL Documentation.
[5] UCAN Specification. User-Controlled Authorization Networks. ucan.xyz.
[6] W3C CCG. Authorization Capabilities for Linked Data (ZCAP-LD) (Draft).
[7] Microsoft. Azure SQL Database and Azure Active Directory Integration.
[8] Google Cloud. IAM Authentication for Cloud SQL.
[9] Amazon Web Services. IAM Database Authentication for MySQL/PostgreSQL.
[10] W3C. (2023). Secure Data Storage (Draft).
[11] Cameron, K. (2005). The Laws of Identity.
[12] Decentralized Identity Foundation. (2020). Decentralized Identity Architecture.
[13] Rose, S., Borchert, O., Mitchell, S., Connelly, S. (2020). NIST SP 800-207: Zero Trust Architecture. DOI: 10.6028/NIST.SP.800-207.
[14] U.S. HHS. HIPAA Security/Privacy Rule Resources.
[15] U.S. Congress. (2002). Sarbanes–Oxley Act of 2002 (SOX).
[16] European Union. (2016). General Data Protection Regulation (GDPR), Regulation (EU) 2016/679.

[17] State of California. (2018). California Consumer Privacy Act (CCPA), Cal. Civ. Code §1798.100–1798.199.
[18] Jones, M., Bradley, J., Sakimura, N. (2015). RFC 7519: JSON Web Token (JWT). DOI: 10.17487/RFC7519.
[19] Jones, M., Bradley, J., Sakimura, N. (2015). RFC 7515: JSON Web Signature (JWS). DOI: 10.17487/RFC7515.
[20] Jones, M. (2015). RFC 7517: JSON Web Key (JWK). DOI: 10.17487/RFC7517.
[21] OpenID Foundation. OpenID Connect Discovery 1.0.
[22] Jones, M. (2012). RFC 6750: OAuth 2.0 Bearer Token Usage. DOI: 10.17487/RFC6750.
[23] Altman, N., et al. (2010). RFC 5929: Channel Bindings for TLS. DOI: 10.17487/RFC5929.
[24] PostgreSQL Docs. pg_stat_statements.
[25] Yasuda, K., Lodderstedt, T., et al. Selective Disclosure JWT (SD-JWT) (IETF Draft).
[26] Rose, S., Borchert, O., Mitchell, S., Connelly, S. (2020). Zero Trust Architecture. DOI: 10.6028/NIST.SP.800-207.
[27] ISO/IEC. (2022). ISO/IEC 27001:2022 Information security, cybersecurity and privacy protection — Information security management systems — Requirements.