Entry Point Control for Path-Safe Knowledge Graph Exploration

Shuya Higuchi[1], Natthawut Kertkeidkachorn[2]

[1]Independent Researcher, Japan
[2]Japan Advanced Institute of Science and Technology, Nomi, Ishikawa, Japan

Abstract

Knowledge graphs support multi-hop traversal in SPARQL, GraphRAG, and enterprise APIs. Existing controls regulate nodes, triples, queries, answers, or subject–resource paths, but often leave authorization of query origins implicit in application logic. Under late filtering, an engine may traverse a forbidden intermediate even when it removes that node from the final result. We propose *Entry Point Control* (EP control), which manages traversal origins as first-class policy objects. Per-hop *Access* checks provide the safety invariant; a separate *Start* decision makes origin scope declarative, auditable, and adjustable before recursion. With check-before-enqueue, per-hop enforcement eliminates traversal-touch exposure (T1a), while EP restriction narrows predicate-input exposure (T1b) and the search space.

We instantiate EP control with a three-level ABAC–ReBAC model and a bounded-depth SPARQL encoding. Physical T1a safety requires an engine or hook that checks candidates before enqueue; our Jena experiments validate final bindings, not internal operator working sets. Evaluation covers synthetic KGs (200–1 M nodes), DBpedia+FOAF, Enron, a retrieval-stage GraphRAG simulation, and Jena ARQ. Reference traversals and the retrieval-stage simulation attain T1a = 0 under EP control, while 75 Jena scenarios return no forbidden final bindings.

Holding 20 readable query seeds fixed and granting *Start* to a nested five reduces visited nodes by 0.3% on DBpedia and 10.8% on Enron, showing topology-dependent saturation. A separate, deliberately maximal unrestricted-origin stress bound yields 37.6–40.4% reductions on DBpedia+FOAF; it is not a typical seeded-query comparison. In a same-seed external-PDP probe, the five-root policy reduces PDP calls by 73.9% and distinct forbidden identifiers persisted in its audit log by 77.3%. Updates to one EP, a 3–5-EP rotation set, or one label alter up to 2 297 reachable nodes without rebuilding a materialized frontier. These results separate path safety, established by per-hop checks, from auditable origin scope, managed by the Start catalog.

Keywords

Knowledge Graph, Data Management, Query Processing, Access Control, SPARQL

CEUR-WS.org/Vol-4322/paper_3.pdf

1. Introduction

Knowledge graphs (KGs) provide graph-structured representations for integrating heterogeneous data [1]. In practice, KG usage is rarely limited to single-hop lookup: systems routinely perform multi-hop expansion through SPARQL Property Paths (+, *) over RDF graphs [2], graph-based RAG pipelines that construct and search graph indexes for LLM context [3], and graph-centric enterprise APIs. This operational shift makes graph exploration itself a *security-relevant process*. Even when final answers are filtered, traversal may evaluate unauthorized intermediates and leave database, cache, log, or orchestration traces; RAG privacy work independently demonstrates risks for private retrieval stores [4]. Unlike relational RLS or authorization-transparent rewriting, which bind predicates to selected rows [5, 6], recursive KG controls may leave the authorized origin set implicit and need not guarantee checking before the next-hop working set.

Many KG access-control approaches operate primarily on visibility: they regulate which nodes, triples, answers, or rewritten queries may be exposed to the user. This includes triple/subgraph-level authorization and policy-based filtering [7, 8], dataset-clause scope constraints and query rewriting [9, 6], and controlled query evaluation (CQE) [10]. Path-aware policy languages and native graph privileges

2nd International Workshop on Data Management for Knowledge Graphs (DMKG 2026), co-located with the 25th International Semantic Web Conference (ISWC 2026), October 25, 2026, Bari, Italy
higuzeroggw@gmail.com (S. Higuchi); natt@jaist.ac.jp (N. Kertkeidkachorn)
© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings ceur-ws.org ISSN 1613-0073
published 2026-11-09

can also constrain which paths or graph elements are traversable (Section 2). Our narrower gap is *origin authorization*: whether an otherwise readable or traversable node may initiate recursive evaluation. Expressive rewriting or action-aware languages can encode this check, but the surveyed KG systems do not expose a distinct, fail-closed *Start* decision maintained independently of candidate access. EP stores the origin policy; its frontier is derived at query time. Without an engine-level check-before-enqueue contract, a SPARQL Property Path such as `?x :knows+ ?y` may still evaluate intermediate bindings that never appear in the final result.

We ask how KG access control can govern not only what may be returned, but also where exploration may begin and how far it may propagate. EP control is not a replacement for per-hop enforcement: per-hop $Access$ checks decide whether a candidate node may be entered; EP control decides whether a node may play the distinct $Start$ role for recursive exploration. An unrestricted-origin per-hop traversal can be T1a/T2-safe yet pass the entire forbidden set through the predicate and visit a much larger authorized region (Section 4).

As an illustrative operating scenario, consider a cross-customer incident KG: an investigator may follow readable evidence from ticket-approved case roots but may not launch a new search from every readable event and fan out into other cases. Denying $Access$ to those events would break valid evidence chains; a security operations team instead rotates $Start$ roots while the data owner maintains node access.

We propose **Entry Point Control**, a policy-layer mechanism that makes traversal origins *first-class policy primitives* and combines them with per-hop access enforcement along every traversal path. We instantiate this idea with a *three-level hybrid security model* combining attribute-based access control (ABAC) for public and internal nodes with a restricted relationship-based user–group–resource rule (ReBAC-style) [11] for group-private nodes, together with a *Shared Node Structural Constraint* that ensures group-private regions interconnect only through shared nodes.

EP control treats recursive-origin authorization as catalog metadata. Administrators can narrow, rotate, audit, or revoke origin assignments without rewriting the graph or rebuilding per-user materialized views; the access predicate remains explicit so that origin governance and per-hop safety are separate contributions.

Our contributions are fourfold:

1. We introduce **Entry Point Control**: authorization to start recursion is a managed policy object, while its reachable frontier is derived at query time.
2. We specify EP-controlled reachability as an **enforcement contract** for recursive KG access, separating $Start$ from per-hop $Access$ in a three-level model. Direct consequences cover role separation, safety, monotone updates, and isolation; C1 remains defense-in-depth.
3. We evaluate synthetic KGs (200–1 M nodes), DBpedia and Enron topologies, fixed-seed and external-PDP trace probes, retrieval-stage GraphRAG exposure, and dynamic frontier maintenance.
4. We implement a **bounded-depth SPARQL realization** in Jena ARQ plus a concrete late-filter counterexample; this does not claim unbounded Property Path equivalence or physical T1a inside ARQ.

2. Related Work

KG Access Control. RBAC [12], ABAC, and ReBAC [11] regulate access to resources; KG-specific models apply such controls to graph entities. Kirrane et al. [7] survey RDF access control; Flouris et al. [8] formalize triple/subgraph-level policies; Valzelli et al. [13] combine discretionary, mandatory, and role-based controls in a fine-grained KG model. Their implementation precomputes allowed identifiers at startup and exposes a per-user virtual subgraph through TinkerPop's SubgraphStrategy, making it the closest precomputed personalized-view pattern in our comparison. EP control instead maintains query-origin sets independently and applies them at evaluation time, without precomputing one authorized

identifier set per user. These authorization models can supply EP control's per-node predicate, but do not themselves make query-origin sets managed catalog objects.

Query Rewriting and CQE. Costabello et al. [9] constrain SPARQL query scope through dataset clauses; Rizvi et al. [6] introduce authorization-transparent querying; Cuenca Grau et al. [10] study controlled query evaluation (CQE); Benedikt et al. [14] analyze inference disclosure. The critical distinction: *query rewriting specifies authorized query semantics*; *EP control additionally manages exploration origins*. Depending on the rewrite and physical plan, filters may be pushed into traversal, but answer equivalence alone does not establish a check-before-enqueue guarantee. An expressive rewrite can inject the same test; our distinction is the represented $Start$ decision and its lifecycle, not exclusive expressibility. CQE targets answer disclosure, while the cited systems do not expose origins as independently updatable policy metadata. Role-sensitive CQE over policy-protected virtual KGs likewise targets answer confidentiality rather than managed recursive origins [15].

Path-Aware and Native Graph Enforcement. XACML4G specifies authorized subject–resource paths and protects vertices and edges through datastore-independent policy enforcement [16]; a recent comparison places it alongside ReBAC and query-rewriting approaches for property graphs [17]. Neo4j's TRAVERSE privilege controls whether nodes and relationships can be found, including label-, type-, and property-pattern rules [18]. Neo4j also distinguishes finding an entity (TRAVERSE) from reading its properties (READ), illustrating that graph actions need not share one permission. EP adds an origin action: a node may be traversable when reached but unable to initiate recursion. An analyst may TRAVERSE all `:Internal` nodes while a weekly review starts only at `{a17, a42}`: restricting TRAVERSE to the anchors would block authorized expansion, so the application must otherwise encode, rotate, and audit them separately. EP control formalizes that encodable pattern as a fail-closed, independently maintained policy object. XACML4G could encode this action if modeled explicitly; EP specifies its lifecycle, seed-resolution contract, and composition with candidate access. An application seed list can implement the same set intersection, but its correctness then depends on every recursive entry path consulting that list; the EP contract centralizes denial-by-default resolution and change audit.

GraphRAG Security. GraphRAG uses graph indexes and summaries for retrieval and synthesis [3]. Because RAG systems can leak private information [4], graph expansion adds an exposure surface that answer redaction does not constrain.

Decentralized KG Querying. Link Traversal Query Processing discovers sources on-the-fly under linked-data semantics [19]; seed URIs determine the explored Web region [20]. Distributed subweb specifications separate source selection from the query for guided traversal [21]. EP addresses the security analogue inside one KG: principal/group-specific origin authorization composed with candidate access.

Positioning. Table 1 compares the policy objects and enforcement scopes. Prior work provides path-aware authorization and engine-level traversal checks, and adjacent LTQP work makes source guidance declarative. To our knowledge, the reviewed KG access-control systems do not expose a principal-specific, separately managed $Start$ action with fail-closed seed resolution. This scoped claim concerns lifecycle, not whether expressive languages can encode the test.

Per-hop enforcement or a native traversal privilege can provide element-level safety, but unless an origin action is modeled separately, the query or application still chooses starting nodes implicitly. EP control makes that choice declarative and auditable, supports a fail-closed origin projection, and lets administrators tune or rotate the frontier without changing the underlying authorization predicate.

Table 1
Comparison by primary policy object and enforcement scope. *Path* denotes per-edge or per-element authorization; *Start* denotes a distinct principal/group-specific origin authorization; *dep.* denotes rewrite/plan dependence; *enc.* denotes an encodable capability without a separately managed object; *Maint.* denotes separate grant/revoke/rotation. "—" means not reported under that object, not unavailable; the table does not rank general access-control capability.

Approach	Policy object	Enforcement	Path	Start	Maint.
RDF/KG access [8, 13]	Entity/triple	Authorization/filter	—	—	—
Query rewriting [6]	Query/result	Rewrite + plan	dep.	enc.	—
CQE [10]	Answer/inference	Censor	—	—	—
XACML4G [16]	Subject–resource path	Path policy	✓	enc.	—
Neo4j TRAVERSE [18]	Label/type/pattern	Engine privilege	✓	—	—
EP Control (Ours)	**Start + candidate**	**Pre-expl. + per-hop**	✓	✓	✓

3. Proposed Method

3.1. Overview: Three-Phase Architecture

The policy lifecycle has three phases:

1. **User & Group Registration**: Users are assigned to groups with an internal/external status flag. Each group may own group-private nodes.
2. **Origin Authorization**: Each group receives $Start$ permission for selected boundary anchors, the only nodes permitted to initiate recursive evaluation.
3. **Secure Traversal**: Starting from entry points, BFS enforces access per hop and prunes forbidden branches.

The model uses one logical KG; private regions are classifications in G, while explorable regions are query-time projections. A node can be accessible when reached yet not startable: $Access$ and $Start$ are independent authorization actions. EP control is a search-space restriction rather than a result filter. Per-hop checks prevent forbidden nodes from entering the working set, although their identifiers may still be enumerated as T1b predicate inputs. EP control comprises three independent layers:

Per-hop enforcement Working-set safety: every neighbor is checked before enqueue (Theorem 1).
Origin authorization Exploration governance: $Start$ designates where traversal may begin, shaping the explorable frontier *before* any path is evaluated.
Structural Constraint Defense-in-depth: no edge connects private nodes of distinct groups, verifiable by an instance-level invariant check.

Figure 1 makes this logical model and division of labour explicit: user–group–EP links are catalog metadata; both configurations attain T1a = 0, while EP restriction narrows T1b.
The core security guarantee depends *only* on entry-point safety and per-hop enforcement—not on the Structural Constraint.

3.2. Threat Model

We distinguish four exposure surfaces of multi-hop traversal: **T1a (traversal-touch)**, forbidden nodes added to the engine's working set (Definition 9); **T1b (predicate-input)**, forbidden identifiers passed to $Access(u, \cdot)$ even when subsequently pruned; **T1c (trace-persistence)**, forbidden identifiers retained in query logs, GraphRAG spans, or audit traces; and **T2 (released-output exposure)**, forbidden identifiers released beyond the method's stated enforcement boundary after its final access filter (Definition 8). Figure 1(b) reports the no-Start-gate per-hop baseline, where T1b reaches 1.000 under correct per-hop enforcement. For SPARQL, T1a is recursive work, T1b access-function or PDP inputs, T1c persisted identifiers, and T2 final bindings. For GraphRAG, the working set is T1a; pre-release candidates are

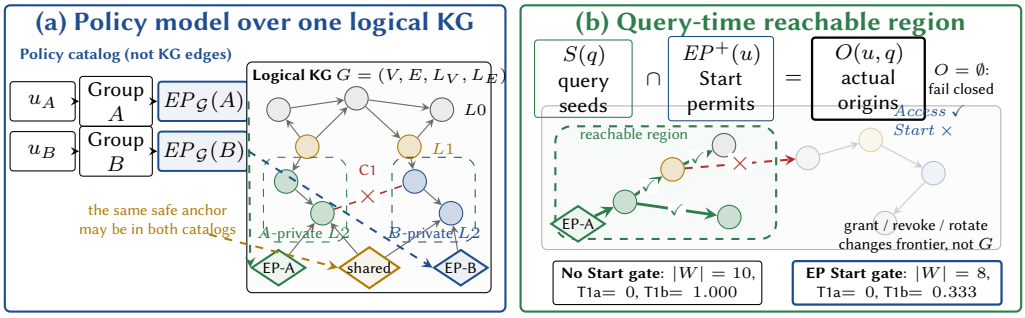


Figure 1: EP control over one logical KG. (a) User–group–EP relations are catalog metadata; private regions coexist in G, and a safe anchor may be shared. C1 is defense-in-depth. (b) $O(u, q)$ and check-before-enqueue derive a query-time region, not a stored per-group graph. A node marked $Access\checkmark / Start\times$ may be used when reached but cannot initiate recursion. Callouts use the fixed 13-node example; scale results are in Table 2.

separate (T1c if persisted), and only post-filter release to a model or application is T2. T1a/T1b become exposure when identifiers cross or persist at a boundary. Check-before-enqueue enforcement eliminates T1a; EP restriction narrows T1b and potential T1c surfaces, while the structural constraint eliminates a specific cross-group class of T1b inputs. Inference-based leakage is out of scope.

3.3. Formal Definitions

Definition 1 (Knowledge Graph). *$G = (V, E, src, dst, L_V, L_E)$ is a directed labeled multigraph with node set V, edge-identity set E, endpoint maps $src, dst : E \to V$, node type $L_V : V \to T$, and edge type $L_E : E \to R$. We write $v \to_G w$ iff some $e \in E$ satisfies $src(e) = v$ and $dst(e) = w$.*

Definition 2 (Security Classification). *A pair (σ, ω) where $\sigma : V \to \{0, 1, 2\}$ assigns a security level and $\omega : V \to \mathcal{G} \cup \{\bot\}$ an owner group ($\omega(v) \neq \bot \iff \sigma(v) = 2$). Level 0 = public (all users), level 1 = internal (authenticated users), level 2 = group-private (owning group only).*

Definition 3 (User Model and Access Predicate). *A user $u \in U$ is characterized by a group membership $group(u) \in \mathcal{G}$ and an internal status $internal(u) \in \{true, false\}$. The access predicate is:*

$$Access(u, v) = \begin{cases} true & \text{if } \sigma(v) = 0 \\ internal(u) & \text{if } \sigma(v) = 1 \\ internal(u) \ \wedge \ group(u) = \omega(v) & \text{if } \sigma(v) = 2 \end{cases} \tag{1}$$

Levels 0–1 follow a threshold pattern (ABAC), while level 2 checks a restricted user–group–resource ownership relation (ReBAC-style), making this a hybrid *access model.*

Definition 4 (Authorized Node Set). *For user u, $R_u = \{v \in V \mid Access(u, v)\}$; the* forbidden set *is $Forbidden(u) = V \setminus R_u$.*

Definition 5 (Origin Authorization and Entry-Point Assignment). *$EP_{\mathcal{G}} : \mathcal{G} \to 2^V$ maps each group to* authorized boundary anchors*: the only nodes from which traversal is permitted to originate. This restriction concerns origins; the same anchor may appear in more than one group's EP catalog. For user u, $EP(u) = EP_{\mathcal{G}}(group(u))$. Define the origin action $Start(u, v) \iff v \in EP(u)$. $Start$ authorizes v to act as a recursive root, whereas $Access$ authorizes v to enter traversal when encountered; neither action implies the other. The desired catalog consistency condition is $\forall u, \ \forall v \in EP(u) : \ v \in R_u$. Because a catalog update or label change may violate this condition, usable anchors are restricted to the* access-safe projection *$EP^+(u) = \{v \in V \mid Start(u, v) \wedge Access(u, v)\}$. Unsafe anchors are rejected or omitted. For a query q with extracted seed set $S(q) \subseteq V$, the actual origin set is $O(u, q) = S(q) \cap EP^+(u)$; seeds outside the authorized EP set are omitted. A query without explicit seeds uses $O(u, q) = EP^+(u)$. If $O(u, q) = \emptyset$, no traversal occurs and the result is empty. The ordinary result intentionally does not reveal no path versus origin rejection or pruning; privileged audit telemetry is separate. Thus stale assignments and unauthorized seeds* fail closed. *Below we suppress q and write O_u.*

Definition 6 (Shared Node Structural Constraint (C1)). *A knowledge graph G satisfies the* shared node structural constraint *if for every edge $e \in E$, writing $v_1 = src(e)$ and $v_2 = dst(e)$:*

$$\sigma(v_1) = 2 \ \wedge \ \sigma(v_2) = 2 \implies \omega(v_1) = \omega(v_2) \tag{2}$$

Equivalently, group-private regions of different groups are connected only *through shared (level 0 or 1) nodes. This constraint ensures that crossing from one group's private region to another's requires passing through a shared boundary node.*

Definition 7 (Proposed Reachability (Path-Level Security)). *The proposed method enforces the access predicate on every node:*

$$Reach_{\text{prop}}(u, d) = \left\{ v \in V \;\middle|\; \begin{array}{l} \exists\, p = (v_0, v_1, \ldots, v_m),\ v_0 \in O_u, \\ v_m = v,\ m \le d,\ \forall i \in \{0, \ldots, m-1\} : v_i \to_G v_{i+1}, \\ \forall i \in \{0, \ldots, m\} :\ Access(u, v_i) \end{array} \right\} \tag{3}$$

Definition 8 (Leakage Ratio (T2)).

$$Leakage(u, d) = \frac{|Out(u, d) \cap Forbidden(u)|}{|Forbidden(u)|} \tag{4}$$

where $Out(u, d)$ contains identifiers released beyond the stated enforcement boundary after its final access filter; internal sets do not count unless released.

Definition 9 (Traversal-Touch Ratio (T1a)).

$$Touch(u, d) = \frac{|Work(u, d) \cap Forbidden(u)|}{|Forbidden(u)|} \tag{5}$$

where $Work(u, d)$ is the set of distinct nodes inserted into the traversal queue, visited set, or expansion working set. Both ratios are defined as zero when $|Forbidden(u)| = 0$.

Definition 10 (Exposure Surfaces). *For a traversal by user u to depth d, let $Enum(u, d)$ be the set of node identifiers enumerated as inputs to $Access(u, \cdot)$; $Work(u, d)$ the set of nodes inserted into the traversal queue, visited set, or expansion working set; $Cand(u, d)$ the pre-release candidate or context-construction set; $Trace(u, d)$ the set of node identifiers retained in logs, spans, caches, or audit traces; and $Out(u, d)$ the set of identifiers released beyond the stated enforcement boundary to an application or model. Absolute exposures are $T_{1a} = |Work(u, d) \cap Forbidden(u)|$, $T_{1b} = |Enum(u, d) \cap Forbidden(u)|$, $T_{1c} = |Trace(u, d) \cap Forbidden(u)|$, and $T_2 = |Out(u, d) \cap Forbidden(u)|$. Definitions 8 and 9 divide the corresponding surfaces by $|Forbidden(u)|$. For the proposed method, $Out = Reach_{\text{prop}}$. Post-filter may populate unsafe $Work$ or $Cand$ but filters before Out; forbidden candidates are reported separately and as T1c if persisted.*

3.4. Security Guarantees

The statements below decompose the enforcement contract and make its assumptions reviewable. Several follow directly from the definitions; they are not presented as deep algorithmic or expressiveness results.

Proposition 1 (Origin-Role Separation). *There are origin policies representable by $(Start, Access)$ that no single context-free node predicate $P(u, v)$, used both to admit roots and reached nodes, can represent.*

Proof sketch. Let $a \to_G x$, where a is startable and both nodes are accessible, but x is not startable. Preserving arrival at x requires $P(u, x) = true$; rejecting it as a root requires $P(u, x) = false$. An action-aware encoding therefore implements the separate origin decision. □

This is a role-separation result, not an expressiveness lower bound for action-aware policy languages.

Theorem 1 (Composed Traversal Safety). *If traversal originates from O_u and exploration enforces path-level security (Definition 7), then $Reach_{\text{prop}}(u, d) \subseteq R_u$ for all u, d.*

Proof sketch. By induction on path length. Base: $v_0 \in O_u \subseteq EP^+(u) \subseteq R_u$. Step: Definition 7 requires $Access(u, v_{m+1})$ for extension, so $v_{m+1} \in R_u$. □

Direct consequences. Theorem 1 gives zero T2 for the proposed method and group isolation because R_u excludes other groups' private nodes. Check-before-enqueue applies the same induction to $Work$, giving T1a = 0 while forbidden predicate inputs remain T1b. For fixed G, u, q, d, and $Access$, enlarging EP^+ cannot shrink reachability. None of these claims requires C1. Without C1, however, a private cross-group edge enumerates its forbidden target before pruning; C1 removes that T1b class and is independently verifiable defense-in-depth.

3.5. Implementation

We implement entry point control as a Python reference prototype (FastAPI + Neo4j 5.14) for mechanism validation; the core definitions transfer to engines supporting action-specific root resolution and per-hop predicates. Its EP manager and security engine implement $Start$ and $Access$; traversal uses check-before-enqueue BFS. Concretely, `can_start` and `resolve_origins` enforce $Start$ before `can_access` enforces candidate $Access$; an accessible non-EP can be reached but cannot start. In Neo4j, nodes carry `security_level` and optional `owner_group` properties; groups connect to EPs via `:HAS_ENTRY_POINT` edges. The API rejects unsafe starts, while the benchmark engine projects catalog sets to EP^+. An in-memory Python engine isolates traversal and maintenance costs (the source of wall-clock values in Tables 4 and 5); Jena ARQ validates the SPARQL realization. The current Neo4j adapter retrieves candidate node records before applying the Python predicate; it therefore demonstrates the policy abstraction and T2-safe application behavior, but does not by itself satisfy the physical T1a check-before-enqueue precondition. A production adapter must push the predicate into the engine or expose only candidate identifiers to a checked traversal hook. In a Jena ARQ 5.3.0 probe, endpoint-filtered `relatedTo+` returns allowed c on the sole route $a \to b \to c$ although b is forbidden; the bounded per-hop query returns nothing. This establishes final-binding behavior, not ARQ working-set T1a; the reference traversal separately enforces check-before-enqueue. The UNION returns endpoints at lengths 1–d; metrics add authorized origins for Definition 7's $m = 0$ case.

4. Evaluation

4.1. Experimental Setup

Evaluation Design. We use synthetic KGs (200–1 M nodes), DBpedia+FOAF (100 k–700 k homepage triples), Enron (19 942 nodes), a deterministic GraphRAG retrieval-stage simulation, and dynamic frontier maintenance. Reported structural metrics use deterministic traversal code and rule- or hash-based overlays (seed = 42); timings are descriptive prototype measurements. Separate LLM-generated graph and sensitivity-label runs are auxiliary and excluded from the quantitative results reported here. The datasets contribute public identifiers and incidence structure. Enron edges derive from communication records; DBpedia homepage pairs are augmented with deterministic synthesized multi-hop connectivity. Access labels and Start catalogs are also overlays, so these are public-data-derived mechanism evaluations, not evidence about deployed organizational policy.

Five-Method Comparison. To isolate individual contributions, we compare five methods: (1) **EP control**: EP restriction + per-hop enforcement on C1-compliant graph (full system); (2) **Post-filter**: unrestricted BFS from EPs, then output filtering—measures traversal-touch exposure (T1a); (3) **Per-hop**: per-hop access check from EPs on a graph *without* C1 (cross-group edges injected)—isolates C1's contribution; (4) **Per-hop-all**: per-hop check from *all authorized nodes* (no EP restriction)—isolates EP restriction's contribution as a deliberately maximal, no-explicit-seed upper-bound baseline rather than a typical application query; (5) **Materialized**: pre-computed authorized subgraph, then BFS from EPs—measures preprocessing cost. All methods except the designated C1 ablation receive identical graphs, users, and EPs; that ablation differs only by injected cross-group private edges. All methods release no forbidden identifiers (T2 = 0) after their stated final filters. Post-filter nevertheless admits forbidden identifiers to its internal working and candidate sets, which is measured separately as T1a. The

Table 2
Ablation on synthetic KGs (depth = 5, seed = 42). T1a counts forbidden nodes added to the working set; T1b is the fraction of the forbidden set passed into the access predicate; *Reduc.* is the fraction of all nodes left unvisited. *Visited* counts distinct working-set nodes before post-filtering. Per-hop-all starts from all authorized nodes (no EP restriction).

Scale	Method	Visited	T1a	T1b	Reduc.
10 k	EP Control	1 529	**0**	0.172	84.7%
	Post-filter	5 768	2 477	—	—
	Per-hop-all	5 500	**0**	1.000	45.0%
100 k	EP Control	2 412	**0**	0.026	97.6%
	Post-filter	12 694	5 269	—	—
	Per-hop-all	55 000	**0**	1.000	45.0%
1 M	EP Control	3 079	**0**	**0.003**	99.7%
	Post-filter	18 701	7 683	—	—
	Per-hop-all	550 000	**0**	1.000	45.0%
	Materialized	3 079	**0**	—	99.7%

BFS experiments treat each stored edge as traversable in both directions, equivalently representing it by two arcs in Definition 1. They use Python 3.10–3.11 with an in-memory graph store. A fixed-condition sensitivity run visits 269 nodes bidirectionally versus 81 following stored directions at 10 k nodes; both keep T1a = T2 = 0. Direction changes the frontier, not check-before-enqueue safety; generator orientations are not claimed as domain semantics.

Code, queries, results, tests, and checked data pointers are public at https://github.com/higuuu/ep-control-dmkg2026-artifact. The reviewed code-and-results snapshot is immutable commit `2562e0aa4a61`; raw datasets are not redistributed.

Research Questions.

RQ1 Do safety guarantees hold across scales and diverse topologies (Section 3.4)?
RQ2 Does EP control keep forbidden identifiers out of retrieval-stage working and candidate-context sets in GraphRAG pipelines?
RQ3 How sensitive are security and efficiency to EP Start authorization, placement, cardinality, and contamination?
RQ4 Can administrators update origin policy and derive new frontiers without rebuilding materialized reachable views?

4.2. RQ1: Traversal Safety, Scalability, and Robustness

Do the safety guarantees hold across scales and diverse topologies?

Table 2 shows that EP control attains $T_1\mathrm{a} = 0$ at every scale, while post-filter touches forbidden nodes during traversal. At 1 M nodes, EP control visits 3 079 nodes versus 550 000 for per-hop-all. The T1b fraction drops from 1.000 under unrestricted-origin per-hop traversal to 0.003 under EP control, a $323\times$ stress-case difference under this maximal baseline, not a typical explicitly seeded estimate; fixed-seed results below isolate $Start$. The scales shown are three representative points from an eight-scale sweep (1 k–1 M nodes), across which EP control leaves 47.1–99.7% of the full graph unvisited and reduces visits by 3.8–99.4% relative to the deliberately maximal per-hop-all baseline. Thus origin policy shrinks permitted work while per-hop enforcement preserves the safety invariant.

Enron Heterogeneous Topology. The Enron builder [22] represents four node types (person, email, thread, folder) from the first 10 000 parsed messages; bodies are excluded. Because the corpus has no access-control labels, a deterministic balanced heuristic overlay propagates group ownership from

Table 3
Ablation on the Enron email corpus (19 942 nodes, 73 270 edges, balanced overlay, depth = 3). Counts are summed across 25 internal users; T1b is the ratio of user-summed forbidden predicate inputs to user-summed forbidden sets.

Method	Visited	T1a	T1b	Prune	Preproc.
EP Control	253 745	**0**	**0.660**	412 k	—
Post-filter	457 805	125 200	—	0	—
Per-hop (no C1)	253 745	**0**	0.672	420 k	—
Per-hop-all	362 410	**0**	0.920	658 k	—
Materialized	253 745	**0**	—	0	yes

Table 4
RDF-encoded evaluation on DBpedia+FOAF (depth = 3). Scale is the number of DBpedia `foaf:homepage` triples; $|V|$ counts distinct RDF subjects and objects (predicate IRIs are excluded); runtime is from the in-memory Python reference engine.

Scale / Partition / Method	$\|V\|$	Visited	T1a	Pruned	Runtime
100 k, hash	277 990				
EP Control		123 611	**0**	64	110 ms
Post-filter		123 675	64	0	61 ms
Per-hop-all		198 033	**0**	79 957	171 ms
700 k, hash	2 024 042				
EP Control		872 884	**0**	96	656 ms
Post-filter		872 980	96	0	624 ms
Per-hop-all		1 463 427	**0**	560 615	1 746 ms

mailbox and message metadata and repairs C1 violations. Exact proportions and generation code are in the artifact. This is not an organizational reconstruction; it tests degree skew, hubs, and cross-group connectivity.

Table 3 confirms the same pattern on heterogeneous real-world topology. Post-filter touches 125 200 forbidden nodes, whereas all per-hop methods attain T1a = 0. Changing only the origin scope reduces T1b from 0.920 under per-hop-all to 0.660 under EP control (125 220 to 89 910 of 136 140 forbidden identifiers, user-summed). Holding EPs fixed, C1 removes a further 1.1 points (0.6715 to 0.6604) of the cross-group T1b class described in Section 3.4.

RDF-Encoded Validation on DBpedia+FOAF. To test EP control in a Semantic Web setting, we construct an RDF graph from real `foaf:homepage` triples in the DBpedia 2022.12 dump [23], plus a synthesized policy-and-connectivity overlay using `foaf:knows`, `schema:worksFor`, and `foaf:mbox`. Entities are partitioned into five organizations by hashing the URI. This deterministic partition is fully reproducible from the dump alone and isolates traversal behavior from any external classifier.

Table 4 shows T1a = 0 for EP control on the 2.02 M-vertex RDF encoding, while post-filter exposes forbidden nodes during traversal. Against the deliberately maximal per-hop-all stress bound, EP restriction reduces the visited set by 37.6–40.4%. This bound demonstrates the possible effect of origin scope when every authorized node can start; it is not a typical fixed-seed application comparison. These rows measure the reference traversal over an RDF-encoded graph, not Jena or triple-store throughput; the separate Jena study below validates the bounded-depth query construction at small scale.

Fixed Seeds on Public-Data Topologies. For DBpedia, the fixed query seeds are the 20 highest-degree entity members of the selected deterministic-overlay organization (URI tie-break); all 20 pass *Access* for that organization's internal user. Granting a nested five of those seeds *Start* changes visits 137 916→137 527 (0.3%). On Enron, the corresponding top 20 readable nodes and nested five change

Table 5
Dynamic frontier maintenance (depth = 3). *Policy* Δ is the changed catalog state; *Frontier* Δ is the change in reachable nodes; rebuild time (ms) is the median of 30 trials. Every row changes the cached view (Stale = yes).

Dataset	Operation	Policy Δ	Frontier Δ	Rebuild	Stale	T_1a
Syn-1M	grant EP	+1 EP	+134	0.552	yes	**0**
	revoke EP	−1 EP	−46	0.312	yes	**0**
	rotate EPs	5-EP set	+397	0.798	yes	**0**
	label update	1 label	+1	0.330	yes	**0**
Enron	grant EP	+1 EP	+332	14.5	yes	**0**
	revoke EP	−1 EP	−2 297	11.8	yes	**0**
	rotate EPs	3-EP set	−2 297	11.3	yes	**0**
	label update	1 label	+1	14.0	yes	**0**

visits 11 787→10 519 (10.8%). T1a and T2 remain zero in both comparisons. A few high-degree roots can saturate one component, so this same-seed ablation, rather than per-hop-all, is the causal comparison.

4.3. RQ2–RQ3: GraphRAG Context and EP Sensitivity

GraphRAG Retrieval-Stage Exposure. When graph expansion supplies candidate LLM context, traversal exposure can become prompt leakage if the candidates are serialized. On the 1 000-node synthetic graph at depths 1–5, EP control, per-hop, and materialized traversal place zero forbidden nodes in the retrieval-stage working set. At depths 4–5, the post-filter working set contains 14 550 forbidden nodes summed across 28 users (519.6 per user; 52.0% of the user-summed pre-filter candidates). Its final filter removes them, so measured T2 remains zero; sending that pre-filter pool to a model would instead constitute T2. This is a deterministic retrieval-stage simulation, not an observed LLM prompt.

External PDP and Trace Persistence. Every candidate decision crosses a loopback HTTP PDP that writes identifier and decision to JSONL. Holding graph, user, 20 seeds, predicate, and depth fixed, a nested five-seed Start set changes visits 906→262, calls 1 427→372, and distinct forbidden logged IDs 481→109 (77.3%); both runs keep T1a/T2 at zero. This measures T1c for that logging configuration, not for every PDP.

EP Sensitivity. All five placement strategies preserve T1a/T2 at zero on 10 k nodes. With 20 readable seeds fixed and a nested five startable, visits fall 906/972→262/269 at 5 k/10 k (71.1–72.3%). Varying $k = 1$ to 20 changes authorized coverage 0.03→0.30; a fully unsafe Start set fails closed with zero visits. Across 75 Jena ARQ scenarios (200/1 k/10 k nodes, five internal groups, depths 1–5), EP-controlled queries produce no forbidden candidate bindings; each unrestricted candidate query does before a final filter. This validates logical binding construction and predicate invocation, not physical T1a in ARQ.

RQ4: Dynamic Frontier Maintenance. We compare grant, revoke, rotation, and label updates with recomputing a materialized reachable-frontier view for the same group and depth. This measures change scope and staleness, not DBMS transaction latency.

Table 5 shows frontier deltas of 1–2 297 nodes; every cached view becomes stale, while EP uses the new catalog at the next query and keeps T1a at zero. Enron revoke and rotation coincide because remaining high-degree seeds saturate the same depth-3 core. Concurrent invalidation is outside this experiment.

5. Discussion

Why Entry Points Matter. At the traversal-algorithm level, EP control operates over a conventional per-hop BFS; algorithmic novelty is not claimed. Its managed $Start(u, v)$ differs from $Access$: a reached

node may be traversable but unable to start (Proposition 1). Allowlists or expressive languages can encode the same decision; the seeded ablation holds $Access$ fixed.

Policy-Aware Query Processing. EP assignments are $O(|\mathcal{G}|k)$ catalog edges or rows for k EPs per group. A security operations team proposes ticket-scoped anchors, the data owner validates accessibility and frontier samples, and a policy administrator grants, revokes, or rotates them. Reviews test empty, broad, stale, and cross-group assignments; audits retain actor, group, origins, depth, policy version, and ticket. Human review cost is unevaluated. The cited rewriting and CQE systems [9, 6, 10] do not expose this lifecycle.

SPARQL Applicability. VALUES carries only small O_u, not the policy; indexed or external sec:access(...) supplies decisions. The UNION implements policy-aware bounded reachability, not complete standard + semantics: depth and pruning deliberately change the query. Ordinary empty results hide no-path versus policy pruning; privileged audit provides diagnostics. A normal FILTER may still bind before testing, so physical T1a needs a checked plan or traversal hook. We do not evaluate the throughput or scaling behavior of an indexed or external sec:access(...) implementation.

Limitations. Enron and DBpedia use reproducible policy overlays, and DBpedia also uses synthesized multi-hop connectivity; they test public-data-derived graphs, not native enterprise policy. The prototype is not production-throughput evidence; GraphRAG is retrieval-stage. Jena supplies logical final-binding evidence only; this scoped physical-T1a limitation does not apply to the reference engine's explicit check-before-enqueue working set. The external-PDP result measures T1c only for its identifier-persisting audit configuration. Per-hop-all is a maximal bound; on fixed high-degree seeds, DBpedia improves by only 0.3%, showing topology-dependent saturation. Concurrent revocation, client caching, policy-authoring studies, indexed/external access-function scalability, standard unbounded Property Paths, and inference leakage remain unevaluated [14].

6. Conclusion

Entry Point Control makes authorization to $Start$ recursion an auditable policy object separate from per-hop $Access$. Reference traversals attain T1a = 0 under check-before-enqueue; Jena validates final bindings, not internal working sets. Restricting the same 20 readable seeds to a nested five reduces visits by 0.3% on DBpedia and 10.8% on Enron; the larger 37.6–40.4% DBpedia+FOAF reduction is a maximal unrestricted-origin stress bound. EP also permits edits affecting up to 2 297 reachable nodes without a materialized frontier, making origin scope independently manageable and auditable at query time.

Declaration on Generative AI

During revision, the authors used Anthropic Claude and OpenAI ChatGPT/Codex for clarity, manuscript–result consistency, copyediting, and formatting. The authors also used them to generate auxiliary graph variants and email-sensitivity labels; these outputs are excluded from the quantitative results, which use deterministic traversal and rule- or hash-based overlays. The authors ran the experiments, verified the claims against code, data, and sources, approved the text, and take responsibility.

References

[1] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. de Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, A.-C. N. Ngomo, A. Polleres, S. M. Rashid, A. Rula, L. Schmelzeisen, J. Sequeda, S. Staab, A. Zimmermann, Knowledge graphs, ACM Computing Surveys 54 (2021).

[2] R. Angles, M. Arenas, P. Barceló, A. Hogan, J. Reutter, D. Vrgoč, Foundations of modern query languages for graph databases, ACM Computing Surveys 50 (2017).
[3] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, D. Metropolitansky, R. O. Ness, J. Larson, From local to global: A graph RAG approach to query-focused summarization, arXiv preprint arXiv:2404.16130 (2024).
[4] S. Zeng, J. Zhang, P. He, Y. Xing, Y. Liu, H. Xu, J. Ren, S. Wang, D. Yin, Y. Chang, J. Tang, The good and the bad: Exploring privacy issues in retrieval-augmented generation (RAG), in: Findings of the Association for Computational Linguistics: ACL 2024, Association for Computational Linguistics, 2024, pp. 4505–4524.
[5] PostgreSQL Global Development Group, Row security policies, PostgreSQL 17 Documentation, 2026. URL: https://www.postgresql.org/docs/17/ddl-rowsecurity.html, accessed 2026-07-18.
[6] S. Rizvi, A. O. Mendelzon, S. Sudarshan, P. Roy, Extending query rewriting techniques for fine-grained access control, in: Proc. ACM SIGMOD, 2004, pp. 551–562.
[7] S. Kirrane, A. Mileo, S. Decker, Access control and the resource description framework: A survey, Semantic Web 8 (2017) 311–352. doi:10.3233/SW-160236.
[8] G. Flouris, I. Fundulaki, M. Michou, G. Antoniou, Controlling access to RDF graphs, in: Proc. Future Internet Symposium (FIS), volume 6369 of *LNCS*, Springer, 2010, pp. 107–117.
[9] L. Costabello, S. Villata, F. Gandon, Context-aware access control for RDF graph stores, in: Proc. European Conference on Artificial Intelligence (ECAI), volume 242 of *FAIA*, 2012, pp. 282–287.
[10] B. Cuenca Grau, E. Kharlamov, E. V. Kostylev, D. Zheleznyakov, Controlled query evaluation for Datalog and OWL 2 profile ontologies, in: Proc. IJCAI, 2015, pp. 2883–2889.
[11] P. W. L. Fong, Relationship-based access control: Protection model and policy language, in: Proc. ACM CODASPY, 2011, pp. 191–202.
[12] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, Role-based access control models, IEEE Computer 29 (1996) 38–47.
[13] M. Valzelli, A. Maurino, M. Palmonari, A fine-grained access control model for knowledge graphs, in: Proc. SECRYPT, 2020, pp. 595–601.
[14] M. Benedikt, B. Cuenca Grau, E. V. Kostylev, Logical foundations of information disclosure in ontology-based data integration, Artificial Intelligence 262 (2018) 52–95.
[15] D. Baura, D. Calvanese, User access control in policy-protected virtual knowledge graphs, in: Proc. 14th International Joint Conference on Knowledge Graphs (IJCKG), volume 16297 of *Lecture Notes in Artificial Intelligence*, Springer, 2026, pp. 204–219. doi:10.1007/978-981-95-5009-8_14.
[16] A. Mohamed, D. Auer, D. Hofer, J. Küng, XACML extension for graphs: Flexible authorization policy specification and datastore-independent enforcement, in: Proc. SECRYPT, 2023, pp. 442–449. doi:10.5220/0012090000003555.
[17] A. Mohamed, D. Auer, D. Hofer, J. Küng, Comparison of access control approaches for graph-structured data, in: Proc. SECRYPT, 2024, pp. 576–583. doi:10.5220/0012861500003767.
[18] Neo4j, Inc., Read privileges: The TRAVERSE privilege, Neo4j Operations Manual, 2026. URL: https://neo4j.com/docs/operations-manual/current/authentication-authorization/privileges-reads/, accessed 2026-07-18.
[19] O. Hartig, SPARQL for a web of linked data: Semantics and computability, in: Proc. ESWC, 2012, pp. 8–23.
[20] R. Taelman, R. Verborgh, Link traversal query processing over decentralized environments with structural assumptions, in: Proc. ISWC, 2023, pp. 3–22.
[21] B. Bogaerts, B. Ketsman, Y. Zeboudj, H. Aamer, R. Taelman, R. Verborgh, Distributed subweb specifications for traversing the web, Theory and Practice of Logic Programming 24 (2024) 394–420.
[22] B. Klimt, Y. Yang, The Enron corpus: A new dataset for email classification research, in: Proc. ECML, 2004, pp. 217–226.
[23] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, C. Bizer, DBpedia: A large-scale, multilingual knowledge base extracted from Wikipedia, Semantic Web 6 (2015) 167–195.