Differential Deletion in Databases

Vishal Chakraborty
University of California (UC), Irvine
vchakrab@uci.edu

Youri Kaminsky
Hasso Plattner Institute,
University of Potsdam
youri.kaminsky@hpi.de

Sharad Mehrotra
UC, Irvine
sharad@ics.uci.edu

Felix Naumann
Hasso Plattner Institute,
University of Potsdam
felix.naumann@hpi.de

Sarvesh Pandey
xyz@uci.edu

Arnav Dhariya
ABC XYZ
xyz@uci.edu

ABSTRACT

Meaningful data deletion from a database, given a set of dependencies, has been interpreted in recent work as: a deleted value should not be reconstructible from the remaining database using the dependencies. We call this property Semantic Erasure Against Leakage (SEAL). Achieving SEAL is challenging as databases contain rich semantic structure. Existing approaches enforce SEAL deterministically (e.g., by deleting all dependent data), often causing substantial over-deletion. We propose non-deterministic deletion with probabilistic guarantees on re-inference and call it *Differential Seal* (DiffSeal). This property bounds what an adversary can learn about a deleted cell's value from the observed auxiliary deletion pattern. We consider a unified dependency model, capturing probabilistic constraints as weighted rules, that induces cell-level inference. To balance post-deletion inferability against the number of auxiliary deletions, we design a bounded utility function. Our deletion mechanisms use this utility function and satisfy ε-DiffSeal while selecting high-utility masks. Experiments on real datasets show that DiffSeal substantially reduces auxiliary deletions relative to deterministic baselines with realistic leakage control.

PVLDB Reference Format:
Vishal Chakraborty, Youri Kaminsky, Sharad Mehrotra, Felix Naumann, Sarvesh Pandey, and Arnav Dhariya. Differential Deletion in Databases . PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at URL_TO_YOUR_ARTIFACTS. (approximate)

1 INTRODUCTION

Meaningful data deletion from a database with a given set of constraints has been interpreted in recent work [2, 8, 9, 11] as: after deletion, the remaining database should not enable re-inference of the deleted value via dependencies. Achieving this goal is challenging as databases contain rich semantic structure.

ID	Zip	Symptom	Result	Diagnosis	Age	BMI	Treatment
t_1	94022	Cough	Pos_Flu	Flu	45	32	3.1
t_2	92617	Cough	Pos_Flu	Flu	42	31	3.1
t_3	94022	Fever	Pos_Flu	—	46	33	3.2
t_4	92617	Wt-loss	A1C_6	Diabetes	65	33	12.4

Constraints:

$\sigma_1 : (\{(\text{Result}, t_{x_1})\}, \emptyset) \Rightarrow \text{Diag}$ $\qquad w_{\sigma_1} = 0.95$

$\sigma_2 : (\{(\text{Age}, t_{x_1}), (\text{BMI}, t_{x_1})\}, \emptyset) \Rightarrow \text{Diag}$ $\qquad w_{\sigma_2} = 0.85$

$\sigma_3 : (\{\text{Sym}, \text{Zip}, \text{Diag}\}, \{t_{x_1}[\text{Zip}] = t_{x_2}[\text{Zip}], t_{x_1}[\text{Sym}] = t_{x_2}[\text{Sym}]\}) \Rightarrow \text{Diag}$ $\qquad w_{\sigma_3} = 0.80$

Figure 1: Database D and dependencies for running example.

Example 1.1. Database D in Figure (Fig.) 1 records medical observations with attributes Symptom, Zip, Age, BMI, Diagnosis, and Result and Treatment. Tuple t_3 records a patient whose diagnosis has been deleted. Constraint σ_1 states that Result predicts Diagnosis with strength 0.95; σ_2 states that Age and BMI together predict Diagnosis with strength 0.85; σ_3 states that patients sharing Symptom and Zip tend to share Diagnosis with strength 0.80. These represent constraints that adversaries may exploit to reconstruct deleted data. Note that σ_3 is cross-tuple: it applies only when its join predicates (e.g., matching Zip and Symptom) hold between two tuples.

Suppose we want to delete Diagnosis of a patient. If the system deletes only this cell, an adversary can still infer the likely diagnosis using the weighted rules. Naïve deletion fails.

SEAL. Prior work on deleting data from databases while preventing re-inference from dependencies — which we call **Semantic Erasure Against Leakage** (SEAL) — requires deleting more than just the requested data. Proposed approaches include deleting *all* dependent data [2, 11] or identifying minimal deletion sets via ILP formulations [29] (often infeasible at scale). A more nuanced formulation, Pre-insertion Post Erasure Equivalence (P2E2) [9], considers adversarial knowledge at insertion time and defines deletion as the gain in knowledge (through constraints) since insertion, significantly reducing required deletions. However, all existing approaches are deterministic, treating inference as either fully enabled or blocked. While this offers strong guarantees, it causes extensive over-deletion; in deeply connected databases, potentially deleting so much critical data that deletion becomes impractical.

We explore an alternative: *non-deterministic deletion with probabilistic guarantees.* Rather than completely blocking all inference paths (requiring deletion of all dependent data), we model data dependencies as probabilistic channels of inference (e.g., approximate functional dependencies [34], differential dependencies [27],

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

or relaxed dependencies [6]) and randomly mask auxiliary data in a coordinated manner that bounds the adversary's inferential advantage while preserving more utility (less additional deletion). Our probabilistic framework complements scoping mechanisms such as P2E2, bringing significant improvements by tolerating bounded inferential leakage rather than requiring perfect inference prevention. Moreover, we quantify how an adversary's prior belief about the deleted value shifts after observing the deletion outcome.

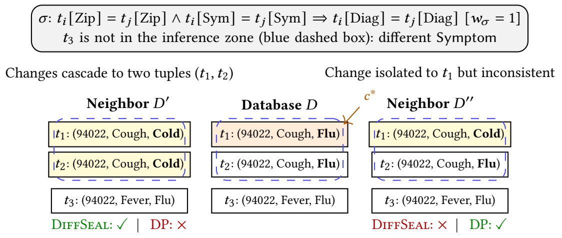
Consider a non-coordinated approach that deletes cells independently using a heuristic rule (e.g., maximum rule strength). This suffers two fundamental limitations. First, it offers no worst-case guarantee: in Example 1.1, deleting each cell with probability equal to its strongest constraint weight (Result at 0.95, Age and BMI at 0.85, Symptom and Zip at 0.80) still retains Result with probability 0.05, allowing inference of Diagnosis with probability 0.95 through σ_1. Second, increasing probabilities to reduce this risk (e.g., deleting cells with probability 0.99) approaches deterministic over-deletion. Independent per-cell decisions cannot simultaneously achieve low expected deletions and bounded worst-case leakage.

Formalizing DiffSeal. We view deletion as a randomized mechanism that outputs a *mask*—a set of auxiliary cells to remove along with the requested cell. The mask reduces residual inferability, but if mask selection depends on the deleted value (in Example 1.1, more deletions for common diagnoses that create many cross-tuple edges, fewer for rare diagnoses), the deletion pattern itself leaks information. Meaningful privacy requires the distribution over masks to be nearly identical regardless of the deleted value. We call this property *Differential Seal* (DiffSeal).

We formalize DiffSeal drawing on differential privacy (DP) [12] and differential obliviousness []. Whilst DP ensures that outputs reveal almost nothing about whether an individual's data is included, DiffSeal ensures that auxiliary deletion patterns reveal bounded information about the deleted cell's *value*. In our example, a patient's Diagnosis could be "Flu" or "Cold"; the mechanism should produce similar mask distributions for both values, preventing adversaries from learning the true value by observing which cells were masked.

Neighbourhood refinements. Standard differential privacy defines neighbouring databases as those differing in a single tuple, implicitly assuming tuples are independent [12]. However, as demonstrated in [22], this assumption breaks down when data exhibits correlations: changing one record may affect others, and privacy guarantees become unclear. The Pufferfish framework [24] addresses correlated data by allowing custom specification of secrets and discriminative pairs, but leaves instantiation for specific constraint types largely open. For databases with integrity constraints, prior work on differentially private data synthesis has assumed that outputs satisfy constraints [15] or enforced consistency via post-processing [18], while work on correlated data has focused on tuple correlations [28, 39] rather than logical constraints.

Our setting presents a unique challenge. We consider *explicit* dependencies (such as functional dependencies and ETL rules) and *implicit* dependencies (discovered from data [1, 32]), expressing both as weighted rules. The weight denotes the probability of correct inference. Therefore, we require a notion of neighborhood that captures alternative values the deleted cell could have taken while maintaining consistency with the constraints. Consequently, neighboring databases should account for cascading changes across tuples linked by cross-tuple constraints. Consider Fig. 2, where rule σ states that tuples with matching Zip and Symptom *tend to* share Diagnosis. Changing t_1[Diagnosis] from "Flu" to "Cold" may then necessitate altering another tuple, i.e., t_2. Unlike DP, in our setting, neighboring databases can differ in more than one tuple.



σ: $t_i[\text{Zip}] = t_j[\text{Zip}] \wedge t_i[\text{Sym}] = t_j[\text{Sym}] \Rightarrow t_i[\text{Diag}] = t_j[\text{Diag}]$ $[w_\sigma = 1]$
t_3 is not in the inference zone (blue dashed box): different Symptom
Changes cascade to two tuples (t_1, t_2)
Change isolated to t_1 but inconsistent
Neighbor D'
Database D
Neighbor D''
c^*
t_1: (94022, Cough, **Cold**)
t_2: (94022, Cough, **Cold**)
t_3: (94022, Fever, Flu)
t_1: (94022, Cough, **Flu**)
t_2: (94022, Cough, **Flu**)
t_3: (94022, Fever, Flu)
t_1: (94022, Cough, **Cold**)
t_2: (94022, Cough, **Flu**)
t_3: (94022, Fever, Flu)
DiffSeal: ✓ | DP: ×
DiffSeal: × | DP: ✓

Figure 2: Neighboring Databases.

We define neighbors *with respect to the the cell c^* being deleted* (target cell). For dependency-aware deletion, two databases are neighbors if they differ in c^*'s value and in changes required to maintain consistency with dependencies. The changes are confined to the *inference zone* (blue dashed box in Fig. 2), comprising cells connected to c^* through dependencies. Cells outside this zone remain identical across neighbors (e.g., t_3 in Fig. 2). Observe that neighboring databases may have *different structures* when join predicates in inter-tuple constraints are satisfied in one database but not in its neighbor, causing tuples to appear or disappear.

DiffSeal Mechanisms. To guarantee DiffSeal, we use the exponential mechanism [30], sampling masks proportional to a bounded utility function, which in our case, balances two competing objectives: minimizing *inferential leakage* and auxiliary deletions to preserve data utility. Our formulation of leakage captures the cumulative threat from multiple inference paths: even if each remaining path has low strength, their combination can yield substantial exposure.

For deployment at scale, we provide two refinements of the exponential mechanism for exact sampling. First, the *greedy Gumbel-max mechanism* which constructs masks progressively without exhaustive enumeration, making it suitable for large inference zones. Second, a *2 phase mask generation* mechanism that pre-computes candidate masks offline and samples from them at deletion time, which is practical for batching frequently arising deleting requests.

Evaluation. We evaluate our mechanisms on four real-world and one artificial dataset against related baselines, measuring post-deletion leakage, auxiliary deletion volume, and deletion time.

Together, our formulation of DiffSeal and mechanisms that guarantee DiffSeal provide a principled foundation for probabilistic meaningful deletion with worst-case guarantees in the presence of explicit and implicit dependencies. Our approaches navigate the inherent tradeoff between inference protection and data preservation, ensuring that the choice of *which* auxiliary cells to delete does not compromise the privacy of *what* was deleted.

Organization. Section (§) 2 introduces our data model and constraints. §3 formalizes deletion through inference using constraints and deletion masks. §4 defines neighboring databases and our deletion guarantee, DiffSeal. §5 analyzes inferential leakage, defines a utility function for masks, and establishes its sensitivity bound for

Table 1: Dependency types with formal notation and instantiation rules.

Type	Scope	Join	Constraint: $(T, J) \Rightarrow h$	w_σ	$\mathcal{H}_{\max}$	$\mathcal{H}_D$
FD [10]	Intra	–	$(\{(\mathsf{SSN}, t_1)\}, \emptyset) \Rightarrow (\mathsf{Name}, t_1)$	1.0	All tuples	All tuples
AFD [34]	Intra	–	$(\{(\mathsf{Age}, t_1), (\mathsf{BMI}, t_1)\}, \emptyset) \Rightarrow (\mathsf{Diag}, t_1)$	$1-g_3$	All tuples	All tuples
CFD [13]	Intra	J_p	$(\{(\mathsf{Country}, t_1), (\mathsf{ZIP}, t_1)\}, \{t_1[C] = \text{'US'}\}) \Rightarrow (\mathsf{State}, t_1)$	1.0	All tuples	Pattern satisfied
DD [27?]	Inter	J_ϕ	$(\{(\mathsf{Date}, t_1)\}, \{\lvert t_1[D] - t_2[D] \rvert \leq 7\}) \Rightarrow (\mathsf{Price}, t_2)$	s	Join satisfiable	Join satisfied
RFD [6]	Inter	J_s	$(\{(\mathsf{Name}, t_1)\}, \{sim(t_1[N], t_2[N]) \geq 0.8\}) \Rightarrow (\mathsf{SSN}, t_2)$	ρ	Join satisfiable	Join satisfied
DC [35]	Inter	J_e	$(\{(\mathsf{SSN}, t_1)\}, \{t_1[S] = t_2[S]\}) \Rightarrow (\mathsf{Name}, t_2)$	$1-\epsilon$	Join satisfiable	Join satisfied
MD [14]	Inter	J_m	$(\{(\mathsf{SKU}, t_1)\}, \{sim(t_1[K], t_2[K]) \geq 0.9\}) \Rightarrow (\mathsf{ProdID}, t_2)$	1.0	All tuples	Join satisfied
IND [7]	Cross	$J_\exists$	$(\{(\mathsf{DeptID}, t_1)\}, \{\exists t_2 \in \mathsf{Dept} : t_1[D] = t_2[D]\}) \Rightarrow (\mathsf{Valid}, t_1)$	1.0	All tuples	FK exists

Notation: $(T, J) \Rightarrow h$ where T=tail (attribute, tuple-variable pairs), J=join predicates, h=head (single attribute, tuple-variable pair). **Scope:** Intra=single tuple (all variables t_1), Inter=tuple pairs (variables t_1, t_2), Cross=cross-table references. **Join predicates:** –=none ($\emptyset$), J_p=pattern (e.g., $t[A] = c$), J_ϕ=distance (e.g., $|t_1[A] - t_2[A]| \leq \delta$), J_s=similarity (e.g., $sim(t_1[A], t_2[A]) \geq \theta$), J_e=equality (e.g., $t_1[A] = t_2[A]$), J_m=matching, $J_\exists$=existence (e.g., $\exists t_2 \in R$). **Weight interpretation (conservative):** g_3=error rate (AFD), s=support (DD), ρ=extent (RFD), ϵ=violation rate (DC). For AFD, we use global error rate. For DD/RFD/DC, we assume $P_{\text{join}} = 1$ (matching tuple always exists).

our mechanisms presented in §6. §7 presents 2-phase mask generation for batched deletions. §8 evaluates our approaches. §9 and §10 discuss related and future work.

2 PRELIMINARIES

We begin by introducing our data model and formalizing the dependency framework.

Data Model. We consider a database instance D consisting of tuples $\{t_1, t_2, \ldots, t_n\}$ over a schema $\mathcal{S} = \{A_1, A_2, \ldots, A_m\}$. For tuple t_i and attribute A_j, we write $t_i[A_j] \in \mathsf{dom}(A_j)$ to denote the value of the cell and $c_{i,j}$ to denote the cell itself. When the context is clear, we may write $t[A]$ for a tuple t and attribute A, and use $t[S]$ to denote the projection of tuple t onto attribute set $S \subseteq \mathcal{S}$. A deletion request targets a specific cell $c^* = t[A_k]$, where t is the tuple containing the user's data and A_k is the attribute.

Dependencies. We capture dependencies among data using weighted dependency constraints. Consider the following example which motivates our formalization in Definition 2.2.

Example 2.1. In Fig. 1, several dependencies enable inference of $t_3[\text{Diagnosis}]$. E.g., Constraint σ_1: test results predict diagnosis 95% of the time, a dependency holding most of the time and Constraint σ_3: patients in the same zip code with identical symptoms share diagnoses 80% of the time, a cross-tuple dependency linking multiple records. These patterns differ in two ways: (1) predictive strength varies (95% vs 80%), requiring *weights*; (2) some operate within single tuples while others span tuples, requiring *join conditions*. Cross-tuple patterns also manifest differently based on data values. If tuples t_2 and t_3 share the same zip and symptoms, an inference link exists between them that wouldn't otherwise. This necessitates distinguishing *schema-level constraints* (input specifications) from *cell-level instantiations* (concrete inference edges over actual data).

Definition 2.2. A dependency constraint $\sigma : (T, J) \Rightarrow h$ has: (1) A *tail* $T = \{(A_{y_1}, t_{x_1}), \ldots, (A_{y_p}, t_{x_p})\}$ of (attribute, tuple-variable) pairs; (2) *Join predicates* J (possibly empty) constraining tuple-variables, e.g., $t_{x_i}[A_{x_i}] = t_{x_j}[A_{y_j}]$ or distance/similarity conditions; (3) A *head* $h = (A_h, t_{x_h})$ consisting of a single (attribute, tuple-variable) pair; (4) A *weight* $w_\sigma \in (0, 1]$ representing inference probability. The constraint states that for tuples satisfying J, knowing all-but-one attributes in the set $T \cup \{h\}$ allows inferring the unknown attribute with probability w_σ.

Constraint Scope. A constraint is *intra-tuple* if all tuple-variables in $T \cup \{h\}$ are identical (e.g., all are t_{x_1}); otherwise it is *inter-tuple*. Intra-tuple constraints enable inference within a single tuple and have $J = \emptyset$ or single-variable predicates (e.g., $t_{x_1}[A] = c$). Inter-tuple constraints enable inference across tuples and have join predicates specifying relationships between tuple-variables (e.g., equality, distance, similarity). These constraints can express various types of approximate constraints explored in the literature (see Table 1).

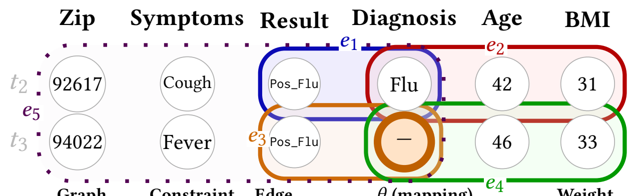
In Fig. 1, the intra-tuple constraint σ_1 has $w = 0.95$ with the tuple variable t_{x_1}; knowing a single tuple's test result allows inferring its diagnosis. The inter-tuple constraint σ_3 has $w = 0.80$, with join predicates $J = \{t_{x_1}[\mathsf{Zip}] = t_{x_2}[\mathsf{Zip}], t_{x_1}[\mathsf{Symptom}] = t_{x_2}[\mathsf{Symptom}]\}$. Tuple-variables differ ($t_{x_1} \neq t_{x_2}$), so knowing t_{x_1}'s attributes and the join conditions allows inferring t_{x_2}'s diagnosis.

Constraint Instantiation. To reason about inference over a given database instance, we instantiate constraints by assigning tuples in D to the tuple-variables appearing in the constraint. Given a constraint σ with tuple-variables $\{t_{x_1}, \ldots, t_{x_p}\}$, an instantiation is a mapping $\theta : \{t_{x_1}, \ldots, t_{x_p}\} \to D$ that satisfies all join predicates in J. This produces a cell-level hyperedge $e[\theta] = \{\theta(t_{x_i})[A] : (A, t_{x_i}) \in T \cup \{h\}\}$ with weight w_σ. When instantiated as a hyperedge, if all but one cell in the hyperedge are known, the remaining cell can be inferred with probability w_σ.

Example 2.3. For intra-tuple σ_1, each tuple yields an instantiation: $\theta_1 : t_{x_1} \mapsto t_3$ produces $e[\theta_1] = \{t_3[\text{Result}], t_3[\text{Diagnosis}]\}$ with weight 0.95. For inter-tuple σ_3, if tuples t_1 and t_3 share Zip="94022" and Symptom="Fever", then $\theta_2 : \{t_{x_1} \mapsto t_1, t_{x_2} \mapsto t_3\}$ produces the hyperedge (with weight 0.80): $e[\theta_2] = \{t_1[\text{Symptom}], t_1[\text{Zip}], t_1[\text{Diagnosis}], t_3[\text{Symptom}], t_3[\text{Zip}], t_3[\text{Diagnosis}]\}$. If t_1 and t_3 do not share these attributes, the join predicates are not satisfied and no hyperedge is created for this tuple pair.

Two Instantiation Modes. We distinguish two instantiation modes corresponding to how constraints are evaluated —(i) *Maximum instantiation:* For intra-tuple constraints, instantiate for *all tuples* in D regardless of current attribute values. For inter-tuple constraints, enumerate tuple pairs where join predicates *could be satisfied* given the database. This provides a conservative, value-agnostic superset of potential data dependencies; (ii) *Actual instantiation:* Include only instantiations where join predicates J are *actually satisfied* by current attribute values in D. This accurately reflects the data dependencies in the current database state.

In our framework, constraints are either *explicit* (weight $w = 1$, e.g., functional dependencies, keys, business rules that must hold) or *implicit* (weight $w < 1$, capturing statistical or heuristic inference rules). We write $\bar{\Sigma}$ for explicit constraints, $\tilde{\Sigma}$ for implicit



Graph	Constraint	Edge	θ (mapping)	Weight
$\mathcal{H}_D$	σ_1	🟦 e_1	$\theta_1 = \{t_{x_1} \mapsto t_2\}$	0.95
$\mathcal{H}_D$	σ_2	🟥 e_2	θ_1	0.85
$\mathcal{H}_D$	σ_1	🟧 e_3	$\theta_2 = \{t_{x_1} \mapsto t_3\}$	0.95
$\mathcal{H}_D$	σ_2	🟩 e_4	θ_2	0.85
Only $\mathcal{H}_{\max}$	σ_3	⁙ e_5	$\theta_3 = \{t_{x_1} \mapsto t_2,\ t_{x_2} \mapsto t_3\}$	0.80

Figure 3: $\mathcal{H}_D$ restricted to $\{t_2, t_3\}$ (with $c^* = t_3$[Diagnosis])

constraints, and $\Sigma = \bar{\Sigma} \cup \tilde{\Sigma}$. We assume that the input instance D satisfies all explicit constraints in $\bar{\Sigma}$. Implicit constraints need not hold universally; instead, they contribute weighted inference edges when their join predicates are satisfied in D.

3 FORMALIZING DELETION MECHANISMS

Given a cell in a database, a deletion mechanism $\mathcal{M}$ determines a *mask*— a additional set of cells to be deleted to prevent re-inference of the deleted cell. In this section, we formalize inference and then formally define deletion masks as a means to prevent inference.

3.1 Inference

In our example, the target cell t_3[Diagnosis] participates in hyperedges such as in Example 2.3. Under our hypergraph semantics, if an adversary knows all-but-one cells in a hyperedge, the remaining cell can be inferred with probability given by the edge's weight w_e.

Definition 3.1 (Dependency Hypergraph). Given constraints Σ over a database D, the dependency hypergraph $\mathcal{H}_D = (V, E)$ has vertices $V = \{c_{i,j} : t_i \in D, A_j \in \mathcal{S}\}$ (all cells) and weighted edges E containing hyperedge e_θ with weight w_σ for each constraint $\sigma \in \Sigma$ and valid instantiation θ of σ over D.

The hypergraph is undirected: each hyperedge represents an inference channel. Given an adversary knowledge set $K \subseteq V$, an inference step via edge e is applicable when $|e \setminus K| = 1$, in which case the unique cell in $e \setminus K$ can be inferred with probability w_e and added to K. A *path* $\pi = (e_1, \ldots, e_\ell)$ to target cell c^* is a sequence of applicable inference steps that ends by inferring c^*. We associate with each path an inference probability $w(\pi) = \prod_{i=1}^{\ell} w_{e_i}$ representing the probability that all inference steps in the path succeed. For a target cell c^*, we write $\mathcal{H}_D^{c^*}$ to denote the subgraph induced by all cells and hyperedges that lie on at least one inference path to c^*. This subgraph captures all possible ways to infer the target cell.

Example 3.2. Fig. 3 shows the hypergraph $\mathcal{H}_D^{c^*}$ for target cell $c^* = t_3$[Diagnosis], with instantiations θ_1, θ_2, and θ_3 restricted to tuples t_2 and t_3. Edge e_1 is constraint σ_1 instantiated on θ_1, edge e_2 is constraint σ_2 instantiated on θ_1. Similarly, edge e_3 (and e_4) is constraint σ_1 (and σ_2) instantiated on θ_2. If tuples t_2 and t_3 shared zip and symptom values, another hyperedge, e_5 (dotted in the figure), constraint σ_3 instantiated on θ_3, would have been present in the hypergraph. However, since the join predicate of σ_3 (t_2[Zip] $\neq$ t_3[Zip]) is not satisfied the edge e_5 does not exist in $\mathcal{H}_D$.

3.2 Deletion Masks

When a cell c^* is deleted from a database D with constraint set Σ, a deletion mechanism must determine which auxiliary cells to remove to prevent inference. This set of cells is called a deletion mask. We now formalize deletion masks and how they prevent inference. Later, in § 5, we quantify the effectiveness of a mask in terms of how much inference it prevents.

Maximum Hypergraph. Since deletion masks must not reveal the value of the target cell, we must account for databases where c^* takes a different value. In such databases, other cells may also change to maintain consistency with Σ. To capture this, we introduce the *maximum hypergraph* of c^*, denoted $\mathcal{H}_{\max}^{c^*}$, a value-agnostic extension of $\mathcal{H}_D^{c^*}$ where constraints are *conservatively instantiated* following Table 1. For intra-tuple constraints, edges are included for all tuples regardless of current attribute values. For inter-tuple constraints, edges are included for all tuple pairs where join predicates could potentially be satisfied. The maximum hypergraph depends only on the schema, constraints, and tuple identifiers, not on actual cell values.

Example 3.3. Consider constraint σ_3 in Fig. 1. In Example 2.3, instantiation θ_3 on σ_3 produces a candidate hyperedge e_5 (dotted int eh figure) with cells t_1[Diag], t_3[Diag], t_1[Zip], t_1[Sym], t_3[Zip], and t_3[Sym] and weight $w = 0.80$. In $\mathcal{H}_{\max}$, such hyperedges exist for all tuple pairs regardless of whether join predicates currently hold. If t_3[Sym] were "Flu" and t_2[Zip] = "94022", both predicates would hold and $e[\theta_2]$ would appear in $\mathcal{H}_D^{c^*}$.

This distinction is essential: the mask space consisting of the vertices in $\mathcal{H}_{\max}^{c^*}$ remains fixed across databases differing only in c^*'s value, enabling our privacy guarantee (§ 4), while leakage computation on $\mathcal{H}_D^{c^*}$ accurately reflects current inference capability.

Inference Zone. Application specifications and regulatory requirements may prohibit certain cells from being masked. The *inference zone* $\mathcal{I}(c^*)$ captures the set of cells that a deletion mechanism may mask when deleting c^*. Formally, given constraint set Σ over database D and target cell $c^* \in D$, let $\mathcal{R}(c^*) \subseteq D$ denote the cells reachable from c^* in $\mathcal{H}_{\max}^{c^*}$: a cell $c \in \mathcal{R}(c^*)$ if there exists a path $(e_1, \ldots, e_\ell)$ with $c^* \in e_1$, $c \in e_\ell$, and $e_i \cap e_{i+1} \neq \emptyset$ for consecutive edges. Let $\mathcal{F} \subseteq D$ denote the set of immutable cells that cannot be masked due to application constraints. The inference zone is then $\mathcal{I}(c^*) = \mathcal{R}(c^*) \setminus \mathcal{F}$. Unless otherwise stated, we assume $\mathcal{F} = \emptyset$, so $\mathcal{I}(c^*) = \mathcal{R}(c^*)$.

Definition 3.4 (Deletion Mask). Given a target cell c^* with value $t_\ell[A_k]$ in database D, a deletion mechanism $\mathcal{M} : (D, c^*) \rightarrow M$ produces a mask $M \subseteq \mathcal{I}(c^*)$ with $c^* \in M$. The mask transforms each cell $c \in D$ as:

$$t_\ell[A_j] \mapsto \begin{cases} \perp & \text{if } c \in M \\ t_\ell[A_j] & \text{otherwise} \end{cases} \tag{1}$$

Example 3.5 (continued). For target cell $c^* = t_3$[Diagnosis] in our running example, different masks provide different protection levels. The minimal mask $M = \{c^*\}$ blocks no inference paths, leaving all paths active and resulting in maximum post-deletion inference. At the other extreme, a mask hitting all hyperedges containing c^* blocks all inference paths, achieving zero post-deletion inference.

We design mechanisms that, given a target cell in a database, produce masks balancing two competing objectives: minimizing post-deletion inference (quantified via inferential leakage in §5) and minimizing auxiliary cell deletions (measured by mask utility).

4 DIFFERENTIAL SEAL

This section formalizes our *privacy* guarantee for deletion: a mechanism should not reveal the deleted value through the *deletion pattern* (i.e., which auxiliary cells are masked). We capture this goal by constraining the distinguishability of the *mask distribution* across plausible alternative databases, i.e., neighboring databases.

4.1 Neighboring Databases

Standard differential privacy defines neighboring datasets by modifying a single tuple [12]. In cell-level deletion under dependencies, however, changing a target cell may necessitate corresponding changes to other cells in order to remain consistent with constraints [1]. We therefore define adjacency relative to a target cell and its inference zone.

Definition 4.1 (Neighboring Databases). Given a database instance D with tuples $t_1, \ldots, t_n$ over schema $\mathcal{S}$ and a target cell c^* in D, we say that database D' is a neighbor of D (written $D \sim_{c^*} D'$) if:

1. **Target cell differs:** The value of c^* in D differs from its corresponding cell in D'.
2. **Cells outside the inference zone are identical:** For all cells $c \notin \mathcal{I}(c^*)$, the value of c in D equals its value in D'.
3. **Explicit consistency:** Both $D \models \bar{\Sigma}$ and $D' \models \bar{\Sigma}$, where $\bar{\Sigma}$ denotes the set of explicit (must-hold) constraints.

In Definition 4.1, Condition (2) restricts all changes to the inference zone $\mathcal{I}(c^*)$, while allowing arbitrary (but consistent) modifications within $\mathcal{I}(c^*)$. This yields a robust adjacency notion: the privacy guarantee holds even under structural variation and non-minimal cascades inside the zone. Although implicit (weighted) dependencies $\tilde{\Sigma}$ may/may not be treated as integrity constraints, these constraints induce inference edges when their join predicates are satisfied in the realized instance.

Example 4.2. Let $c^* = t_3[\text{Diagnosis}]$ have value "Flu" in D. Suppose an explicit constraint states that Treatment is functionally determined by Diagnosis, in our notation, $(\{(\text{Diagnosis}, t_1)\}, \emptyset) \Rightarrow (\text{Treatment}, t_1)$ with weight 1. In a neighboring database D' where $t_3[\text{Diagnosis}]$ = "Cold", the cell $t_3[\text{Treatment}]$ must also change to satisfy the explicit constraint. Moreover, if tuples t_1 and t_3 share the same Zip and Symptom values, then a cross-tuple dependency can create an inference edge linking $t_1[\text{Diagnosis}]$ and $t_3[\text{Diagnosis}]$. In that case, $t_1[\text{Diagnosis}] \in \mathcal{I}(c^*)$, and neighboring databases may also differ in $t_1[\text{Diagnosis}]$ while remaining identical outside $\mathcal{I}(c^*)$.

Structural Variation Across Neighbors. The actual hypergraph $\mathcal{H}_D$ depends on which join predicates and patterns are satisfied in the current database. Neighboring databases may have different actual hypergraphs $\mathcal{H}_D$ and $\mathcal{H}_{D'}$: when values within $\mathcal{I}(c^*)$ change, join predicates may become satisfied or unsatisfied, causing hyperedges (and thus inference paths) to appear or disappear.

Example 4.3 (Structural Variation). Consider Fig. 2. It demonstrates a cross-tuple dependency with join predicate $t_1[\text{Symptom}] = t_3[\text{Symptom}]$, where $t_1[\text{Symptom}]$ = "Fever" (from Example 4.2). Suppose that in D' (to remain explicitly consistent with the changed diagnosis) we also changed $t_3[\text{Symptom}]$ to "Cough". Then:

- In D: the join predicate holds and the corresponding cross-tuple edge may exist in $\mathcal{H}_D^{c^*}$.
- In D': the join predicate does not hold, and the cross-tuple edge is absent in $\mathcal{H}_{D'}^{c^*}$.

Thus, $\mathcal{H}_D^{c^*}$ and $\mathcal{H}_{D'}^{c^*}$ can differ, leading to different inference paths (and potentially different leakage values) for the same mask.

Despite this structural variation, our privacy guarantee remains well-defined because any realized hypergraph $\mathcal{H}_D$ is a sub-hypergraph of the value-agnostic maximum hypergraph $\mathcal{H}_{\max}$. Consequently, the mechanism's output space is fixed across neighbors: masks are subsets of the same inference zone $\mathcal{I}(c^*)$ derived from $\mathcal{H}_{\max}$.

4.2 Privacy Guarantee

With neighboring databases defined, we now formalize *Differential Seal* (DiffSeal).

Definition 4.4 (ε-Differential Seal (DiffSeal)). For a randomized deletion mechanism $\mathcal{M} : (D, c^*) \to M$ that outputs a deletion mask $M \subseteq \mathcal{I}(c^*)$, we say the mechanism $\mathcal{M}$ satisfies *ε-Differential Seal* if for all pairs of neighboring databases $D \sim_{c^*} D'$ and all measurable sets of masks $\mathcal{R} \subseteq 2^{\mathcal{I}(c^*)}$,

$$\Pr[\mathcal{M}(D, c^*) \in \mathcal{R}] \leq e^{\varepsilon} \cdot \Pr[\mathcal{M}(D', c^*) \in \mathcal{R}] . \tag{2}$$

This definition ensures that observing the mask does not allow an adversary to reliably distinguish whether the deleted cell had one value or another among neighboring worlds. Equivalently, the mask can change an adversary's posterior odds between any two neighboring hypotheses by at most a multiplicative factor e^{ε}.

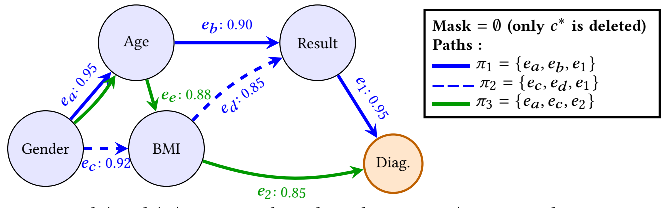
Example 4.5. In our running example, suppose the mechanism has to delete $c^* = t_3[\text{Diagnosis}]$ and outputs mask $M = \{t_3[\text{Result}]\}$ with probability p on D (where the diagnosis is "Flu"). In a neighboring database D' where $t_3[\text{Diagnosis}]$ = "Cold", the same mask must be output with probability within a factor of $e^{\pm\varepsilon}$ of p.

The parameter ε controls this tradeoff between privacy and utility: smaller ε forces the mechanism to behave more similarly across neighbors, which can restrict how aggressively it can optimize for inference suppression and deletion cost. In the next section, we formalize inference suppression via a leakage function that computes the probability an adversary can correctly infer c^* from the unmasked cells, and a utility function that balances inference prevention with deletion cost.

5 LEAKAGE ANALYSIS OF MASKS

To determine the effectiveness of a mask, we formalize residual post-deletion inference as inferential leakage (§ 5.1), design a utility function balancing inference risk against deletion costs (§ 5.2), and establish sensitivity bounds for the exponential mechanism.

[1] This can be limited to only explicit constraints ($\bar{\Sigma}$: keys, FDs, hard business rules with weight 1) or extended to include implicit constraints ($\tilde{\Sigma}$: statistical dependencies with weight < 1). Usually, explicit constraints must be maintained for database consistency.


Age
Result
Gender
BMI
Diag.
e_b: 0.90
e_a: 0.95
e_e: 0.88
e_d: 0.85
e_1: 0.95
e_c: 0.92
e_2: 0.85
Mask = $\emptyset$ (only c^* is deleted)
Paths :
$\pi_1 = \{e_a, e_b, e_1\}$
$\pi_2 = \{e_c, e_d, e_1\}$
$\pi_3 = \{e_a, e_c, e_2\}$

Path (weight)	Channel Weight	Leakage
$\pi_1\,(0.814)$ $\pi_2\,(0.743)$	$w^*_{e_1}$=1−(1−0.814)(1−0.743)=0.952	$\mathcal{L}$ = 1−(1−0.952) × (1−0.710) =0.986
$\pi_3\,(0.710)$	$w^*_{e_2}$=0.710	

Leakage for other masks:
$M = \{Res\}$: Blocks $\pi_1, \pi_2 \rightarrow \mathcal{L} = 0.710$; $M = \{Res, BMI\}$: Blocks all paths $\rightarrow \mathcal{L} = 0$

Figure 4: Leakage computation in hypergraph $\mathcal{H}_{ex}$ (nodes are depicted at the attribute level for presentational ease).

5.1 Inferential Leakage

To evaluate whether a mask provides adequate privacy protection, we need to quantify how much inference capability an adversary retains after deletion. We measure this through inferential leakage: the probability that at least one active inference channel successfully reveals the deleted cell's value. We will refer to the hypergraph $\mathcal{H}_{ex}$ depicted in Fig. 4 to illustrate concepts presented in this section.

A path $\pi = (e_1, \ldots, e_\ell)$ to target c^* is *blocked* by mask M if the adversary cannot complete the sequence of inferences. This occurs when some hyperedge e_i in the path requires a masked cell to infer its output. Formally, at step i, edge e_i infers a unique cell $x_i \in e_i$ given that all other cells in $e_i \setminus \{x_i\}$ are known. The path is blocked if for some i, $(e_i \setminus \{x_i\}) \cap M \neq \emptyset$.

Let $E^* = \{e \in E : c^* \in e\}$ denote the set of hyperedges containing the target cell. Each hyperedge $e \in E^*$ represents a distinct *inference channel*—an inference channel is the final step through which c^* can be inferred. In Fig. 4, paths π_1 and π_2 both end at edge e_1 which is an inference channel. For each inference channel $e \in E^*$, we define its effective weight under mask M by considering all unblocked paths ending at e. Since multiple paths may end at the same edge (representing alternative ways to establish the cells needed for that final inference step), we aggregate them as follows:

$$w^*_e(\mathcal{H}, M) = 1 - \prod_{\substack{\pi \in \mathcal{H}:\text{final}(\pi)=e \\ \pi \text{ not blocked by } M}} (1 - w(\pi)) \tag{3}$$

where $w(\pi) = \prod_{e' \in \pi} w(e')$ is the path inference probability, computed as the product of edge weights under the assumption that inference steps are conditionally independent. This represents the probability that all inference steps in the path succeed sequentially. The aggregation in Equation 3 computes the probability that *at least one* unblocked path ending at e succeeds. If all paths ending at e are blocked, then $w^*_e(M) = 0$. In Fig. 4, for the empty mask $M = \emptyset$, we have $w^*_{e_1}(\mathcal{H}_{ex}, \emptyset) = 0.952$ and $w^*_{e_2}(\mathcal{H}_{ex}, \emptyset) = 0.710$.

Definition 5.1 (Inferential Leakage). Given database D, target cell c^*, and mask M, the inferential leakage is:

$$\mathcal{L}(M, c^*, D) = 1 - \prod_{e \in E^*} \left(1 - w^*_e(\mathcal{H}, M)\right). \tag{4}$$

The leakage formula in Equation 4 represents the probability that *at least one* inference channel successfully reveals c^*. The formula computes the leakage $\mathcal{L}$ = Pr(at least one channel succeeds) =1 − Pr(all channels fail) =1−$\prod_{e \in E^*}$ Pr(channel e fails) where the probability Pr(channel e fails) = 1 − $w^*_e(M)$. In Fig. 4, we show the leakages for various masks for the hypergraph $\mathcal{H}_{ex}$.

Our formulation groups inference paths by their final hyperedge because paths ending at the same edge $e \in E^*$ are *dependent*. They represent alternative ways to establish the prerequisite cells for the same final inference step. Once any path successfully establishes the cells in $e \setminus \{c^*\}$, the inference of c^* via edge e succeeds with the same probability w_e, regardless of which path was used. Therefore, for paths ending at the same edge, we compute the probability that at least one path succeeds (Equation 3).

In contrast, paths ending at *different* edges use fundamentally different inference mechanisms: different constraints operating on different combinations of attributes. We treat these as conditionally independent inference attempts. This grouping structure ensures we don't overcount dependent paths (paths to the same edge) while correctly combining independent inference channels (different edges). When multiple paths to the same final edge share intermediate cells (e.g., both paths π_1 and π_2 start by inferring Age from Gender in Fig. 4), they are not fully independent. In such cases, our formula provides a conservative (upper bound) estimate of the true inference probability, as treating positively correlated paths as independent overestimates the "at least one succeeds" probability. This conservatism is desirable for privacy analysis, ensuring we do not underestimate inference risk.

The leakage satisfies $\mathcal{L} \in [0, 1]$: when all paths are blocked, $w^*_e(M) = 0$ for all $e \in E^*$ and $\mathcal{L} = 0$ (no inference possible); when at least one unblocked path has weight 1 (deterministic inference, e.g., via FD), $w^*_e(M) = 1$ for some $e \in E^*$ and $\mathcal{L} = 1$ (certain inference); when strong probabilistic paths remain active (e.g., weights 0.85, 0.95), $\mathcal{L}$ approaches 1.

Example 5.2. For $c^* = t_3$[Diagnosis] in our running example, the actual hypergraph $\mathcal{H}^{c^*}_D$ contains two instantiated channels: e_1 (from σ_1: Result $\Rightarrow$ Diagnosis) with $w(e_1) = 0.95$ and e_2 (from σ_2: Age, BMI $\Rightarrow$ Diagnosis) with $w(e_2) = 0.85$. The cross-tuple constraint σ_3 (with $w = 0.80$) is a candidate edge in the maximum hypergraph $\mathcal{H}^{c^*}_{\max}$ but is not instantiated in $\mathcal{H}^{c^*}_D$ because the join predicates are not satisfied (Figure 3, dotted edge e_5), hence excluded from leakage computation. For the empty mask $M = \emptyset$, both channels are active: $\mathcal{L}(\emptyset, c^*, D) = 1 - (1 - 0.95)(1 - 0.85) = 0.9925$ (probability adversary successfully infers c^*). For mask $M = \{t_3[\text{Result}]\}$ which blocks e_1: $\mathcal{L}(M, c^*, D) = 1 - (1 - 0)(1 - 0.85) = 0.85$ (probability through remaining channel). For mask $M = \{t_3[\text{Result}], t_3[\text{Age}]\}$ (or $\{t_3[\text{Result}], t_3[\text{BMI}]\}$) which blocks both channels: $\mathcal{L}(M, c^*, D) = 0$.

Leakage Across Neighbors. As established in Section 4.1, neighboring databases may have different actual hypergraphs $\mathcal{H}^{c^*}_D$ and $\mathcal{H}^{c^*}_{D'}$ due to value-dependent join predicates and patterns. This affects which inference channels and paths exist. A channel $e \in E^*$ may be present in $\mathcal{H}^{c^*}_D$ but absent from $\mathcal{H}^{c^*}_{D'}$ (or vice versa). While all weights w_e remain constant (determined by constraint specifications), which edges are present changes. Consequently, inference paths may exist in one database but not the other.

Thus, for a mask M, the leakage values $\mathcal{L}(M, c^*, D)$ and $\mathcal{L}(M, c^*, D')$ may differ, representing the actual inference probability in each respective database state. For any target cell c^* in a database D and mask M, we have $0 \leq \mathcal{L}(M, c^*, D) \leq 1$ since in Equation 4, each factor $(1 - w_e^*(M)) \in [0, 1]$, making their product also in $[0, 1]$.

Lemma 5.3. *For any database D, target cell c^*, and mask M, $0 \leq \mathcal{L}(M, c^*, D) \leq 1$.*

This bound holds regardless of how the hypergraph structures differ across neighbors. Along with this and the fixed output space (masks are subsets of $\mathcal{I}(c^*)$) enables the sensitivity analysis below.

5.2 Utility Function and Sensitivity

To select masks balancing privacy protection against information loss, we penalize both residual leakage and deletion cost.

Definition 5.4 (Mask Utility Function). For a mask $M \subseteq \mathcal{I}(c^*)$, target cell c^* in a database D, we define the utility function as $u(M, c^*, D) = -\alpha \cdot \mathcal{L}(M, c^*, D) - \beta \cdot |M|$ where $\alpha > 0$ is the leakage penalty and $\beta > 0$ is the deletion cost.

Higher utility (less negative) indicates better masks, i.e., low leakage and few deletions. The ratio α/β controls the tradeoff. With a high α/β, our mechanism prefers low-leakage masks, accepting more deletions while with a low α/β, our mechanism prefers fewer deletions, accepting higher leakage.

Example 5.5. For $c^* = t_3$[Diagnosis] with $\alpha = 10$, $\beta = 1$, using channels from Example 5.2: Empty mask $M = \emptyset$ has utility $u = -10(0.9925) - 1(0) = -9.925$. Mask $M = \{t_3[\text{Result}]\}$ (blocks e_1, leakage 0.85) has utility $u = -10(0.85) - 1(1) = -9.50$. Mask $M = \{t_3[\text{Result}], t_3[\text{Age}]\}$ (blocks both channels, leakage 0) has utility $u = -10(0) - 1(2) = -2.0$. With high $\alpha = 10$, the mask blocking both channels achieves best utility despite deleting two cells.

To illustrate the tradeoff, if $\beta = 5$: Empty mask has $u = -9.925$, one-cell mask has $u = -10(0.85) - 5(1) = -13.50$, two-cell mask has $u = -10(0) - 5(2) = -10.0$. Now the empty mask is preferable to deleting just one cell (though two cells is still best), showing how α/β controls the privacy-utility tradeoff.

The exponential mechanism requires bounding utility sensitivity across neighbors [12]. With our definition of utility, the key insight in Lemma 5.3 provides the sensitivity bound below that is independent of the number of paths or constraints.

Lemma 5.6. *For neighboring databases $D \sim_{c^*} D'$, the following holds: $\Delta u = \max_{M \subseteq \mathcal{I}(c^*)} |u(M, c^*, D) - u(M, c^*, D')| \leq \alpha$.*

Utility Analysis. We now analyze how the privacy parameter ε affects the expected utility, formalizing the privacy-utility tradeoff.

Theorem 5.7. *Let $\mathcal{R} \subseteq 2^{\mathcal{I}(c^*)}$ be the candidate mask set considered by the mechanism, and let $u(M, c^*, D)$ be as in Definition 5.4 with sensitivity Δu. If $\mathcal{M}$ samples a mask from $\mathcal{R}$ using the exponential mechanism with privacy parameter ε, then for any $\delta \in (0, 1)$, with probability at least $1 - \delta$, the following holds: $u(\mathcal{M}(D, c^*), c^*, D) \geq u^*(D) - \frac{2\Delta u}{\varepsilon}\left(\ln |\mathcal{R}| + \ln \frac{1}{\delta}\right)$, where $u^*(D) = \max_{M \in \mathcal{R}} u(M, c^*, D)$.*

Theorem 5.7 shows that the expected utility approaches optimal as ε increases, with a gap that scales as $O(1/\varepsilon)$. As $\varepsilon \to \infty$, the mechanism selects the optimal mask $M^* = \arg\max_{M \in \mathcal{R}} u(M, c^*, D)$ with probability approaching 1, achieving optimal balance between leakage and deletions. As $\varepsilon \to 0$, the distribution over masks approaches uniform, and both expected deletions and expected leakage approach their average values over all possible masks.

6 DIFFSEAL MECHANISMS

Having formalized the deletion problem and privacy guarantee, we now present mechanisms that achieve ε-DiffSeal. The key challenge is to design a randomized mask selection procedure that satisfies the indistinguishability requirement while balancing two competing objectives: minimizing inferential leakage and minimizing the number of auxiliary cell deletions. We present two approaches to accomplish this in § 6.2 and § 6.3. Both of our approaches, on being given a target cell c^* in a database D, and a set Σ of dependency constraints, begin with constructing the corresponding graphs $\mathcal{H}^{c^*}_{\max}$ and $\mathcal{H}^{c^*}_{D}$, and the inference zone $\mathcal{I}(c^*)$. We present these algorithms in § 6.1 before describing our mechanisms.

6.1 Graph Construction and Path Extraction

Given a target cell $c^* = (t^*, A^*)$ to be deleted, we construct two hypergraphs: the maximum hypergraph $\mathcal{H}^{c^*}_{\max}$ containing all possible constraint instantiations involving c^*, and the actual hypergraph $\mathcal{H}^{c^*}_{D}$ containing only instantiations whose join predicates and patterns are satisfied in the current database state. Both constructions use frontier-based exploration starting from c^*.

Algorithm 1 Construct Hypergraph

```
Input: Database $D$, constraints $\Sigma$, target cell $c^*$, mode $\in$ {Max, Actual}
Output: Hypergraph $\mathcal{H} = (V, E)$
1:  $V \leftarrow \{c^*\}, E \leftarrow \emptyset$, frontier $\leftarrow \{c^*\}$, visited $\leftarrow \emptyset$
2:  while frontier $\neq \emptyset$ do
3:      next $\leftarrow \emptyset$
4:      for each $c \in$ frontier where $c \notin$ visited do
5:          visited $\leftarrow$ visited $\cup \{c\}$
6:          for each $\sigma : (T, J) \Rightarrow h$ with weight $w_\sigma$ in $\Sigma$ do
7:              if $c$ is in $\sigma$ then  ▷ $c$ matches some $(A, t_i) \in T \cup \{h\}$
8:                  edges $\leftarrow$ Instantiate($\sigma, c, D$, mode)
9:                  for each hyperedge $e$ in edges do
10:                     $E \leftarrow E \cup \{(e, w_\sigma)\}, V \leftarrow V \cup e$
11:                     next $\leftarrow$ next $\cup$ ($e \setminus$ visited)
12:     frontier $\leftarrow$ next
13: return $(V, E)$
```

The construction begins with $V = \{c^*\}$ and incrementally discovers connected cells through constraint instantiations. For each cell c in the frontier, we check which constraints $\sigma \in \Sigma$ involve c, instantiate those constraints to find hyperedges, add newly discovered cells to V, and update the frontier. This continues until no new cells are discovered, yielding the inference zone $\mathcal{I}(c^*) = V \setminus \{c^*\}$.

The algorithm is called with mode = Max to construct $\mathcal{H}^{c^*}_{\max}$ (for computing inference zone and mask space) or mode = Actual to construct $\mathcal{H}^{c^*}_{D}$ (for computing leakage). The mode parameter controls how constraints are instantiated in the Instantiate subroutine, which follows the instantiation rules specified in Table 1.

SQL-based constraint instantiation. We use SQL queries to instantiate constraints involving the target cell. The instantiation differs based on whether we construct $\mathcal{H}^{c^*}_{\max}$ or $\mathcal{H}^{c^*}_{D}$:

For **intra-tuple constraints**: in $\mathcal{H}^{c^*}_{\max}$, instantiate for *all tuples* by omitting pattern predicates. In $\mathcal{H}^{c^*}_{D}$, include pattern predicates to retain only tuples where patterns hold. Example for CFD σ with pattern $t[\text{Country}] = \text{'US'}$:

```
-- For H_max: omit pattern check
SELECT t.id FROM Patients t WHERE t.id = @target_tuple_id

-- For H_D: include pattern check
SELECT t.id FROM Patients t
WHERE t.id = @target_tuple_id AND t.Country = 'US'
```

Each returned tuple t_{id} yields a hyperedge containing the cells specified by the constraint's schema. For example, σ_1 (Result $\Rightarrow$ Diagnosis) produces $\{(t_{\text{id}}, \text{Result}), (t_{\text{id}}, \text{Diagnosis})\}$ with weight w_{σ_1}.

For **inter-tuple constraints**: in $\mathcal{H}^{c^*}_{\max}$, omit join predicates to capture all tuple pairs where predicates could be satisfied. In $\mathcal{H}^{c^*}_{D}$, include join predicates to retain only pairs where predicates are currently satisfied. Example for σ_3 where tuples sharing Zip and Symptom share Diagnosis values:

```
-- For H_max: omit join predicates
SELECT t1.id, t2.id FROM Patients t1, Patients t2
WHERE t1.id != t2.id
  AND (t1.id = @target_tuple_id OR t2.id = @target_tuple_id)

-- For H_D: include join predicates
SELECT t1.id, t2.id FROM Patients t1, Patients t2
WHERE t1.id != t2.id
  AND (t1.id = @target_tuple_id OR t2.id = @target_tuple_id)
  AND t1.Zip = t2.Zip AND t1.Symptom = t2.Symptom
```

Each row $(t_{\text{id1}}, t_{\text{id2}})$ defines a hyperedge containing the cells specified by σ_3's schema: $\{(t_{\text{id1}}, \text{Symptom}), (t_{\text{id1}}, \text{Zip}), (t_{\text{id1}}, \text{Diagnosis}), (t_{\text{id2}}, \text{Diagnosis})\}$ with weight w_{σ_3}.

This distinction ensures that $\mathcal{H}^{c^*}_{\max}$ provides a conservative structure that remains fixed across neighboring databases (enabling our differential privacy guarantee), while $\mathcal{H}^{c^*}_{D}$ accurately represents which inference paths are currently active in the database.

Path extraction. Algorithm 2 enumerates inference paths to the target cell c^* by recursively chaining hyperedges in the actual dependency hypergraph $\mathcal{H}^{c^*}_{D}$. It starts from the initially observed cells $S \subseteq V$ (typically all cells outside the masked set M, excluding c^*) and repeatedly searches for *applicable* hyperedges: an edge e is applicable for inferring a cell $x \in e$ when all other cells in e are already known, i.e., $e \setminus K = \{x\}$ for the current known set K. When such an edge is found, the algorithm appends it to the current partial path and either records a completed path if $x = c^*$, or continues recursively by treating x as the next intermediate cell to be inferred.

Each discovered path $\pi = (e_1, \ldots, e_\ell)$ is assigned inference probability $w(\pi) = \prod_{i=1}^{\ell} w(e_i)$ computed under the assumption that inference steps are conditionally independent.

The IsBlocked function checks whether a path is blocked by mask M: a path is blocked if any edge in the path requires a masked cell (i.e., a cell in M) to perform its inference step.

Leakage computation. Once inference paths are extracted, we compute leakage according to Definition 5.1 by aggregating unblocked paths per final edge.

Algorithm 2 Extract Inference Paths to a Target Cell

```
Input: Target cell $c^*$, hypergraph $\mathcal{H} = (V, E)$ where each $e \in E$
       has weight $w(e)$, initial known set $S \subseteq V$
Output: Set of inference paths $\Pi$ (each path is a sequence of hyperedges)
1: $\Pi \leftarrow \emptyset$
2: DFS($c^*, S, [\,], \Pi$)
3: return $\Pi$

4: function DFS($x$, $K$, path, $\Pi$)
5:     for all $e \in E$ such that $x \in e$ do    ▷ Edges that can infer $x$
6:         $U \leftarrow e \setminus K$    ▷ Unknown cells in $e$ under current knowledge
7:         if $U = \{x\}$ then    ▷ All other cells in $e$ are known, so $x$ is inferable
8:             $\Pi \leftarrow \Pi \cup \{ path \mathbin{+\!\!+} [e] \}$
9:         else if $|U| = 1$ then
10:            $y \leftarrow$ the unique element in $U$    ▷ $y \neq x$; first infer $y$
11:            if $y \notin$ VisitedCells($path$) then    ▷ Prevents cycles
12:                DFS($y$, $K \cup \{y\}$, path $\mathbin{+\!\!+} [e]$, $\Pi$)

13: function VisitedCells(path)
14:     return $\bigcup_{e \in path} e$    ▷ All cells appearing in path

15: function IsBlocked(path, mask $M$)
16:     for all $e$ in path do
17:         $x \leftarrow$ the cell inferred by edge $e$ in this path
18:         required $\leftarrow e \setminus \{x\}$    ▷ Cells needed to infer $x$
19:         if required $\cap M \neq \emptyset$ then    ▷ A masked cell is required
20:             return true
21:     return false
```

Algorithm 3 Compute Leakage

```
Input: Mask $M$, target cell $c^*$, hypergraph $\mathcal{H}_D = (V, E)$, paths $\Pi$
Output: Leakage $\mathcal{L}(M, c^*, D)$
1: $E^* \leftarrow \{e \in E : c^* \in e\}$    ▷ Final edges containing target
2: for each $e \in E^*$ do
3:     $\Pi_e \leftarrow \{\pi \in \Pi : \text{final}(\pi) = e \text{ and } \neg\text{IsBlocked}(\pi, M)\}$
4:     if $\Pi_e = \emptyset$ then
5:         $w^*_e \leftarrow 0$    ▷ All paths to $e$ blocked
6:     else
7:         $w^*_e \leftarrow 1 - \prod_{\pi \in \Pi_e} (1 - w(\pi))$    ▷ Aggregate paths to $e$
8: $\mathcal{L} \leftarrow 1 - \prod_{e \in E^*} (1 - w^*_e)$
9: return $\mathcal{L}$
```

In our running example with $c^* = t_3[\text{Diagnosis}]$, if the adversary observes all unmasked cells and two channels are active (via σ_1 with weight 0.95 and σ_2 with weight 0.85), Algorithm 3 computes $\mathcal{L}(\emptyset, c^*, D) = 1 - (1 - 0.95)(1 - 0.85) = 0.9925$. If mask $M = \{t_3[\text{Result}]\}$ blocks the first channel, leakage reduces to $\mathcal{L}(M, c^*, D) = 0.85$. In practice, database indexes on join attributes (e.g., Zip, Symptom) support efficient enumeration of valid constraint instantiations.

6.2 Exponential Mechanism for Exact Sampling

Our first approach provides an exact implementation of DiffSeal using the exponential mechanism [30]. The exponential mechanism samples outputs from a discrete set with probability proportional to the exponential of their utility, scaled by the privacy parameter and inversely scaled by the sensitivity.

Algorithm 4 Exponential Deletion Mechanism

```
Input: Database $D$, target cell $c^*$, constraints $\Sigma$, parameters $\alpha, \beta, \varepsilon$
Output: Deletion mask $M$
1:  Construct $\mathcal{H}_{\max}$ from $\Sigma$ (all possible instantiations)
2:  Compute inference zone $\mathcal{I}(c^*) \leftarrow$ Reachable($c^*, \mathcal{H}_{\max}$)
3:  Construct $\mathcal{H}_D$ from $\Sigma$ and $D$ (valid instantiations only)
4:  Extract inference paths $\Pi \leftarrow$ FindPaths($c^*, \mathcal{H}_D$)
5:  Enumerate candidate masks $\mathcal{R} \subseteq 2^{\mathcal{I}(c^*)}$
6:  for each $M \in \mathcal{R}$ do
7:      $E^* \leftarrow \{e \in E : c^* \in e\}$    ▷ Final edges containing target
8:      for each $e \in E^*$ do
9:          $\Pi_e \leftarrow \{\pi \in \Pi : \text{final}(\pi) = e \text{ and } \neg\text{IsBlocked}(\pi, M)\}$
10:         if $\Pi_e = \emptyset$ then
11:             $w^*_e \leftarrow 0$
12:         else
13:             $w^*_e \leftarrow 1 - \prod_{\pi \in \Pi_e}(1 - w(\pi))$ ▷ Agg. unblocked paths
14:     $\mathcal{L}(M) \leftarrow 1 - \prod_{e \in E^*}(1 - w^*_e)$
15:     $u(M) \leftarrow -\alpha \cdot \mathcal{L}(M) - \beta \cdot |M|$
16: Sample $M$ with $\Pr[M] \propto \exp(\varepsilon \cdot u(M)/(2\Delta u))$ where $\Delta u = \alpha$
17: return $M$
```

Algorithm 4 presents the exponential deletion mechanism. The algorithm begins by constructing two hypergraphs. The maximum hypergraph $\mathcal{H}_{\max}$ (Line 1) includes all possible constraint instantiations regardless of whether join predicates are satisfied, and determines the inference zone $\mathcal{I}(c^*)$ (Line 2). The actual hypergraph $\mathcal{H}_D$ (Line 3) includes only valid instantiations where join predicates and patterns hold under the current database values, and determines which inference paths exist.

The algorithm extracts all inference paths from visible cells to the target c^* in $\mathcal{H}_D$ (Line 4) using depth-first search as described in Algorithm 2. For each candidate mask $M \in \mathcal{R}$ (Line 5), the algorithm groups paths by their final edge $e \in E^*$—edges that directly contain the target cell. For each final edge, it aggregates all unblocked paths ending at that edge (Lines 8-12). The leakage $\mathcal{L}(M)$ (Line 15) combines these edge-level probabilities, computing the probability that at least one inference channel successfully reveals c^*. Utility $u(M)$ (Line 16) balances leakage reduction against deletion cost.

The exponential mechanism samples a mask with probability proportional to $\exp(\varepsilon \cdot u(M)/2\Delta u)$ where $\Delta u = \alpha$ is the utility sensitivity from Lemma 5.6 (Line 17). Thus, we have: $\Pr[\mathcal{M}(D, c^*) = M] = \frac{\exp(\varepsilon \cdot u(M)/2\alpha)}{\sum_{M' \in \mathcal{R}} \exp(\varepsilon \cdot u(M')/2\alpha)}$. The privacy parameter ε controls how strongly the mechanism favors high-utility masks. Large ε yields near-deterministic selection of optimal masks; small ε spreads probability across candidates, providing stronger privacy at the cost of selecting suboptimal masks.

Theorem 6.1. *Algorithm 4 satisfies ε-DiffSeal.*

Asymptotics. Hypergraph construction is $O(|\Sigma| \cdot n^k)$ where k is the maximum number of tuple variables per constraint and n is the number of tuples. Inference zone computation is $O(|E_{\max}| \cdot \max_e |e|)$ where $E_{\max}$ is the edge set of $\mathcal{H}_{\max}$. Path extraction is $O(|\Pi| \cdot |\mathcal{I}(c^*)|)$ where $|\Pi|$ is the number of paths discovered. Leakage computation aggregates paths per final edge: $O(|\Pi| + |E^*|)$. Candidate evaluation is $O(|\mathcal{R}| \cdot (|\Pi| + |E^*|))$. As we will see in Section 8, the algorithm is practical for moderate inference zones, typically $|\mathcal{I}(c^*)| \leq 15$.

Our mechanisms provide *pattern privacy* via ε-DiffSeal, and suppress residual inference by optimizing the utility $u(M) = -\alpha\mathcal{L}(M) - \beta|M|$ defined in Section 5. By Theorem 5.7, the exponential mechanism yields near-optimal utility within the candidate space $\mathcal{R}$.

6.3 Greedy Gumbel-Max Mechanism

For databases with large inference zones where exhaustive enumeration becomes intractable, we develop a greedy approximation using the Gumbel-max trick [23]. Rather than enumerating all possible masks upfront, this approach constructs a mask iteratively by selecting one cell at a time, choosing the cell that provides the best marginal utility gain subject to randomization for privacy.

The Gumbel-max trick provides an efficient way to sample from an exponential distribution without computing normalization constants. For a set of candidates with utilities $\{u_i\}$, sampling Gumbel noise $g_i \sim \text{Gumbel}(0, b)$ for each candidate and selecting $\arg\max_i(u_i + g_i)$ is equivalent to sampling with probability proportional to $\exp(u_i/b)$. The Gumbel distribution can be sampled efficiently using $g = -b \cdot \ln(-\ln(u))$ where $u \sim \text{Uniform}(0, 1)$.

Algorithm 5 Greedy Gumbel-Max Deletion

```
Input: Database $D$, target cell $c^*$, constraints $\Sigma$, parameters $\alpha, \beta, \varepsilon$,
       iterations $K$
Output: Deletion mask $M$
1:  Construct $\mathcal{H}_{\max}$ from $\Sigma$ and compute $\mathcal{I}(c^*)$
2:  Construct $\mathcal{H}_D$ and extract paths $\Pi \leftarrow$ FindPaths($c^*, \mathcal{H}_D$)
3:  $M \leftarrow \emptyset$,   $\varepsilon' \leftarrow \varepsilon/K$
4:  for $k = 1$ to $K$ do
5:      for each $c \in \mathcal{I}(c^*) \setminus M$ do
6:          $\mathcal{L}_{\text{curr}} \leftarrow$ ComputeLeakage($M, c^*, \mathcal{H}_D, \Pi$) ▷ Current leakage
7:          $\mathcal{L}_{\text{new}} \leftarrow$ ComputeLeakage($M \cup \{c\}, c^*, \mathcal{H}_D, \Pi$) ▷ After adding c
8:          $\Delta u(c) \leftarrow \alpha \cdot (\mathcal{L}_{\text{curr}} - \mathcal{L}_{\text{new}}) - \beta$
9:          $g_c \sim \text{Gumbel}(0, 2\alpha/\varepsilon')$
10:         $s_c \leftarrow \Delta u(c) + g_c$
11:     $g_{\text{stop}} \sim \text{Gumbel}(0, 2\alpha/\varepsilon')$    ▷ Option to terminate
12:     $s_{\text{stop}} \leftarrow 0 + g_{\text{stop}}$    ▷ Stop has zero marginal utility
13:     if $s_{\text{stop}} > \max_c s_c$ then
14:         break
15:     $M \leftarrow M \cup \{\arg\max_c s_c\}$
16: return $M$
```

Algorithm 5 presents the greedy Gumbel-max deletion mechanism. After constructing the hypergraphs and extracting inference paths as in Algorithm 4, we initialize an empty mask and allocate privacy budget $\varepsilon' = \varepsilon/K$ to each of K iterations (Lines 1-3).

Marginal utility. In each iteration, we consider adding one cell from $\mathcal{I}(c^*) \setminus M$ to the current mask. For each candidate cell c, we compute the current leakage $\mathcal{L}(M)$ and the leakage after adding c to the mask: $\mathcal{L}(M \cup \{c\})$ (Lines 6-7). Both computations follow Definition 5.1: group paths by final edge, aggregate unblocked paths per edge using $w_e^*(M) = 1 - \prod_\pi(1 - w(\pi))$, then combine across edges. The marginal utility gain is:

$$\begin{aligned}\Delta u(c \mid M) &= u(M \cup \{c\}) - u(M)\\ &= -\alpha \cdot \mathcal{L}(M \cup \{c\}) - \beta \cdot |M \cup \{c\}| - (-\alpha \cdot \mathcal{L}(M) - \beta \cdot |M|)\\ &= \alpha \cdot (\mathcal{L}(M) - \mathcal{L}(M \cup \{c\})) - \beta\end{aligned}$$

The first term captures leakage reduction from blocking additional paths; the second is the deletion cost. Positive marginal utility indicates the leakage reduction exceeds the cost of deleting cell c.

For each candidate cell c, we sample independent Gumbel noise with scale $2\alpha/\varepsilon'$ and compute a noisy score $s_c = \Delta u(c) + g_c$ (Lines 9-10). The algorithm selects the cell with maximum noisy score. By properties of the Gumbel distribution, this is equivalent to sampling from the exponential mechanism distribution $\Pr[c] \propto \exp(\varepsilon' \cdot \Delta u(c)/2\alpha)$, but without explicit probability computation or normalization. We include a stop option with zero marginal utility (Lines 11-14), allowing for early termination when further deletions provide insufficient benefit. The stop option receives its own Gumbel noise and competes with candidate cells through the same Gumbel-max procedure: if its noisy score exceeds all candidates, the algorithm terminates. This prevents over-deletion when leakage is already low.

THEOREM 6.2. *Algorithm 5 satisfies ε-DIFFSEAL.*

Asymptotics. Each iteration evaluates $O(|\mathcal{I}(c^*)|)$ candidates. Computing leakage for each candidate requires aggregating paths by final edge: $O(|\Pi| + |E^*|)$ per candidate. Therefore, total complexity is $O(K \cdot |\mathcal{I}(c^*)| \cdot (|\Pi| + |E^*|))$, polynomial in all parameters.

The greedy algorithm makes locally optimal decisions without considering future impacts. It may mask a cell providing moderate leakage reduction at a given step, missing an alternative that would enable blocking multiple paths later with fewer total deletions. However, the privacy guarantee holds regardless of mask quality: suboptimal masks satisfy ε-DIFFSEAL, ensuring the masking pattern does not reveal the deleted cell's value. The tradeoff is between efficiency and mask optimality, with the greedy approach favoring scalability.

7 TWO-PHASE MASK GENERATION

Data erasure requests have been broadly divided into two types [9]—(i) demand-driven (user-requested and erratic) and (ii) retention-driven (periodic and fixed time-to-live assigned to data when it is collected). For repeated deletions targeting the same attribute type, as in retention-driven erasure, we amortize computation by separating schema-level pre-processing from instance-specific evaluation. Algorithm 6 and 7 present the offline and online phases respectively.

Intra-tuple vs. cross-tuple decomposition. The key insight is that intra-tuple inference zones (from FD, AFD, CFD, MD, IND) depend only on the schema and constraint structure—they are identical for all tuples and all database states. In contrast, cross-tuple inference zones (from DD, RFD, DC) depend on which tuples satisfy

Algorithm 6 Offline: Attribute-Level Template Pre-computation

```
Input: Schema $\mathcal{S}$, constraints $\Sigma$
Output: Template index $\mathcal{T}$ for all attributes
1:  $\mathcal{T} \leftarrow \emptyset$
2:  for each attribute $A \in \mathcal{S}$ do
3:      ▷ Identify intra-tuple constraints for attribute A
4:      $\Sigma_{\text{intra}}(A) \leftarrow \{\sigma \in \Sigma : A$ appears in $\sigma \wedge J(\sigma) = \emptyset\}$
5:      ▷ Build template hypergraph for attribute A
6:      $\mathcal{H}_{\text{template}}(A) \leftarrow$ BUILDTEMPLATEGRAPH$(A, \Sigma_{\text{intra}}(A))$
7:      $\mathcal{I}_{\text{intra}}(A) \leftarrow$ vertices of $\mathcal{H}_{\text{template}}(A) \setminus \{A\}$
8:      $\Pi_{\text{intra}}(A) \leftarrow$ EXTRACTPATHTEMPLATES$(A, \mathcal{H}_{\text{template}}(A))$
9:      ▷ Enumerate all possible intra-tuple masks
10:     $\mathcal{R}_{\text{intra}}(A) \leftarrow 2^{\mathcal{I}_{\text{intra}}(A)}$
11:     ▷ Pre-compute path blocking for each template
12:     for each template $T \in \mathcal{R}_{\text{intra}}(A)$ do
13:         BLOCKED$[T] \leftarrow \{\pi \in \Pi_{\text{intra}}(A) : T \cap \pi \neq \emptyset\}$
14:         UNBLOCKED$[T] \leftarrow \Pi_{\text{intra}}(A) \setminus$ BLOCKED$[T]$
15:     ▷ Index cross-tuple constraints for online phase
16:     $\Sigma_{\text{cross}}(A) \leftarrow \{\sigma \in \Sigma : A$ appears in $\sigma \wedge J(\sigma) \neq \emptyset\}$
17:     $\mathcal{T}[A] \leftarrow (\mathcal{I}_{\text{intra}}(A), \Pi_{\text{intra}}(A), \mathcal{R}_{\text{intra}}(A),$
18:             BLOCKED, UNBLOCKED, $\Sigma_{\text{cross}}(A))$
19: return $\mathcal{T}$
```

join predicates in the current database, varying per deletion request. By pre-computing the intra-tuple component offline, we amortize this cost across many deletion requests for the same attribute.

Offline Phase: Algorithm 6 performs one-time pre-computation for each attribute A in a given schema. For attribute A, it constructs a template hypergraph using only the schema and constraints; no database values are accessed. This involves identifying the intra-tuple inference zone $\mathcal{I}_{\text{intra}}(A)$ containing attributes reachable through single-tuple constraints, and extracting path templates that represent potential inference routes. The algorithm then enumerates all candidate mask templates (subsets of the intra-tuple zone) and pre-computes which path templates each mask would block. Additionally, it indexes cross-tuple constraints by their head attribute for efficient lookup during online processing. These templates are attribute-specific but value-independent, enabling reuse across all deletion requests for the same attribute. Since path weights are fixed (constraint weights don't change) and path blocking depends only on mask structure (not tuple values), this pre-computation is valid across all instances.

Path templates vs. instantiated paths. Path templates in the offline phase are schema-level structures: sequences of constraint types without specific tuple identifiers. For example, a template might specify "infer via σ_1 (FD), then via σ_2 (AFD)" without instantiating it for a specific tuples. During the online phase, these templates are instantiated to specific cells by binding tuple variables to the target tuple t^*. For instance, template path "$\sigma_1 \longrightarrow \sigma_2$" becomes instantiated path "$(t^*[\text{Age}], t^*[\text{BMI}]) \longrightarrow (t^*[\text{Result}], t^*[\text{Diagnosis}])$" for a specific tuple t^*. This separation enables offline pre-computation of path blocking: a mask template blocks a path template if they share an attribute, independent of which tuple is targeted.

Online Phase: Algorithm 7 handles deletion requests for specific cells using the pre-computed attribute templates from Algorithm 6.

Algorithm 7 Online: Cell-Specific Deletion with Templates

```
Input: Database $D$, target cell $c^* = (t^*, A^*)$, template index $\mathcal{T}$,
       parameters $\alpha, \beta, \varepsilon$, size threshold $k_{\max}$
Output: Deletion mask $M$
1:                ▷ Retrieve pre-computed templates for attribute $A^*$
2:  $(\mathcal{I}_{\text{intra}}(A^*), \Pi_{\text{intra}}(A^*), \mathcal{R}_{\text{intra}}(A^*),$
3:      $\textsc{Blocked}, \textsc{Unblocked}, \Sigma_{\text{cross}}(A^*)) \leftarrow \mathcal{T}[A^*]$
4:                ▷ Instantiate intra-tuple templates to specific tuple $t^*$
5:  $\mathcal{I}_{\text{intra}}(c^*) \leftarrow \{(t^*, A') : A' \in \mathcal{I}_{\text{intra}}(A^*)\}$
6:  $\Pi_{\text{intra}}(c^*) \leftarrow \textsc{InstantiatePaths}(\Pi_{\text{intra}}(A^*), t^*)$
7:                ▷ Compute cross-tuple zone by evaluating join predicates
8:  $\mathcal{I}_{\text{cross}}(c^*) \leftarrow \emptyset, \Pi_{\text{cross}}(c^*) \leftarrow \emptyset$
9:  for each $\sigma \in \Sigma_{\text{cross}}(A^*)$ do
10:     linked_tuples $\leftarrow \textsc{ExecuteJoinQuery}(\sigma, t^*, D)$
11:     for each tuple $t' \in$ linked_tuples do
12:         Extract cells from $t'$ participating in $\sigma$ and add to
            $\mathcal{I}_{\text{cross}}(c^*)$
13:         Instantiate constraint $\sigma$ with $(t^*, t')$ and add path to
            $\Pi_{\text{cross}}(c^*)$
14: $\mathcal{I}(c^*) \leftarrow \mathcal{I}_{\text{intra}}(c^*) \cup \mathcal{I}_{\text{cross}}(c^*)$
15: $\Pi(c^*) \leftarrow \Pi_{\text{intra}}(c^*) \cup \Pi_{\text{cross}}(c^*)$
16:               ▷ Choose mechanism based on cross-tuple zone size
17: if $|\mathcal{I}_{\text{cross}}(c^*)| \leq k_{\max}$ then        ▷ Exact enumeration feasible
18:     $\mathcal{R} \leftarrow \emptyset$
19:     for each intra-tuple template $T_{\text{intra}} \in \mathcal{R}_{\text{intra}}(A^*)$ do
20:         for each cross-tuple subset $T_{\text{cross}} \subseteq \mathcal{I}_{\text{cross}}(c^*)$ do
21:             $M \leftarrow \{(t^*, A') : A' \in T_{\text{intra}}\} \cup T_{\text{cross}}$
22:             $\mathcal{R} \leftarrow \mathcal{R} \cup \{M\}$
23:     ▷ Compute utilities using pre-computed blocking and aggre-
        gation
24:     for each $M \in \mathcal{R}$ do
25:         $E^* \leftarrow \{e \in E : c^* \in e\}$      ▷ Final edges containing target
26:         for each $e \in E^*$ do
27:             $\Pi_e \leftarrow \{\pi \in \Pi(c^*) : \text{final}(\pi) = e \wedge$
                $\neg\textsc{IsBlocked}(\pi, M)\}$
28:             if $\Pi_e = \emptyset$ then
29:                 $w_e^* \leftarrow 0$
30:             else
31:                 $w_e^* \leftarrow 1 - \prod_{\pi \in \Pi_e}(1 - w(\pi))$ ▷ Aggregate paths
32:         $\mathcal{L}(M) \leftarrow 1 - \prod_{e \in E^*}(1 - w_e^*)$
33:         $u(M) \leftarrow -\alpha \cdot \mathcal{L}(M) - \beta|M|$
34:     Sample $M$ with $\Pr[M] \propto \exp(\varepsilon \cdot u(M)/2\alpha)$
35: else                   ▷ Large cross-tuple zone, use greedy approach
36:     $M \leftarrow \textsc{GumbelDeletion}(D, c^*, \mathcal{I}(c^*), \Pi(c^*), \alpha, \beta, \varepsilon)$
37: return $M$
```

Given a target cell $c^* = t^*[A^*]$, it retrieves the templates for attribute A^* and instantiates the intra-tuple zone to cells within tuple t^* (Lines 1–6). It then evaluates cross-tuple constraints by executing join queries to find tuples linked to t^*, extending the inference zone with cells from these linked tuples (Lines 7–14). The key optimization is that intra-tuple path blocking has been pre-computed offline, so the online processing focuses only on the variable cross-tuple component.

If the cross-tuple zone is small ($|\mathcal{I}_{\text{cross}}(c^*)| \leq k_{\max}$), the algorithm enumerates combinations of pre-computed intra-tuple templates with cross-tuple cell subsets (Lines 17–22). For each candidate mask, it computes leakage by grouping paths by final edge and aggregating unblocked paths per edge using the probability formula from Definition 5.1 (Lines 24–33). The pre-computed blocking information accelerates this evaluation: for intra-tuple paths, we directly look up which templates block which paths; for cross-tuple paths, we check blocking dynamically. The exponential mechanism then samples a mask (Line 34). For large cross-tuple zones, the algorithm falls back to the greedy Gumbel-max mechanism (Line 36), which constructs masks iteratively without exhaustive enumeration.
Asymptotics. The offline phase for attribute A runs once: $O(2^{|\mathcal{I}_{\text{intra}}(A)|} \cdot |\Pi_{\text{intra}}(A)|)$ to enumerate mask templates and compute blocking. The online phase for cell c^* executes join queries ($O(|\Sigma_{\text{cross}}| \cdot n)$ where n is database size), instantiates paths ($O(|\Pi_{\text{cross}}|)$), and evaluates utilities. When $|\mathcal{I}_{\text{cross}}(c^*)| \leq k_{\max}$, enumeration is $O(2^{|\mathcal{I}_{\text{intra}}(A^*)|} \cdot 2^{k_{\max}} \cdot (|\Pi| + |E^*|))$. The two-phase approach is advantageous when processing many deletions for the same attribute, amortizing the offline cost across multiple requests.

8 EVALUATION

This section evaluates the effectiveness and efficiency of our tow mechanisms that guarantee *Differential Inference Deletion (Diff-Seal)*, namely, (i) **DelExp**: the Exponential Mechanism (presented in §6.2) and (ii) **DelGum**: the Greedy Gumbel-Max mechanism (presented in §6.3). We also evaluate the effectiveness of our two phase mask generation, **2PMask** (presented in §7).

In our experiments, we focus on (i) the extent to which Diff-Seal suppresses inferential leakage by blocking active inference paths; (ii) the auxiliary deletion overhead required to achieve this suppression relative to deterministic baselines; (iii) the privacy–utility tradeoff induced by varying the parameter ε and its effect on leakage, deletion volume, and output variability; (iv) the accuracy, efficiency, and stability of **DelExp** and **DelGum**; (v) the overhead reduction achieved by our two-phase mask generation algorithm.

8.1 Experimental Setup

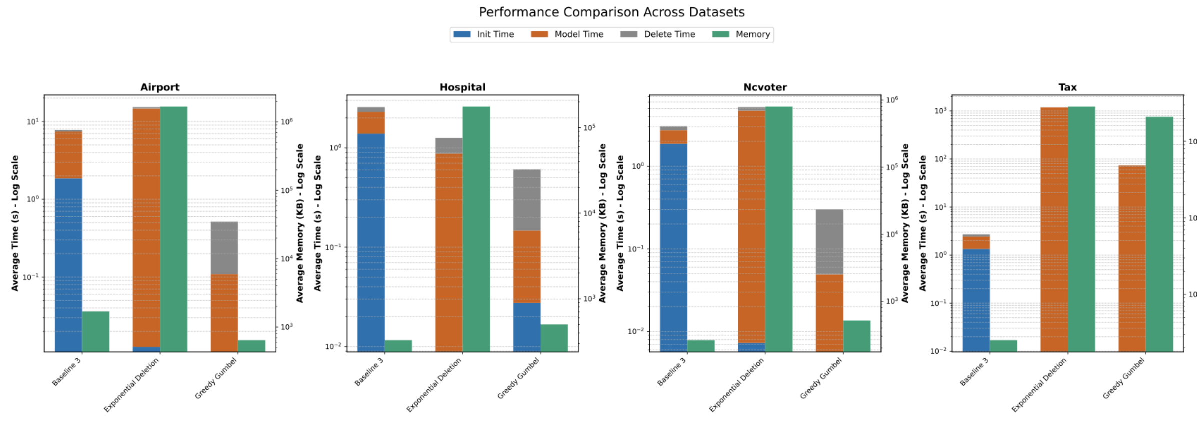
All experiments were run on an Ubuntu-based (20.04 LTS) server (Intel Xeon E5-2650; RAM: 256 GB). All algorithms are single-threaded, running on Java 11 and the datasets are stored in a PostgreSQL (v12.20) database. The ILP approach uses the Gurobi (v11.03) solver.

Datasets. We use four real-world datasets. Table 5a presents some statistics and properties of these datasets. In the following, we briefly describe each of them.
(a) *Medical* [],
(b) *Adult* [],
(c) *COMPAS* [], and
(d) *Flights* []
(e) *TPC-H* []

These datasets exhibit diverse dependency characteristics: the Medical dataset contains dense cross-tuple inference; Adult and COMPAS encode rich attribute correlations; Flights contains many weak, approximate dependencies. Implicit dependencies are mined using standard discovery tools (AFDs, differential dependencies) [],

Dataset	#cells	#Constraints	Source	Instantiated (as logged) DelAll	DelOne	DelMin	DelExp	Deleted auxiliary cells DelAll	DelOne	DelMin	DelExp
Airport[†]			P	21	22	40	24	21	14	14	5.90
Hospital[‡]			H	10	13	40	13	10	11	9	5.16
NCVoter[§]			F	13	13	40	16	13	13	13	5.81
Tax[*]			M	7	7	40	18 450	7	5	5	4.45

(a) Summary of datasets and average number of instantiated & deleted cells. Dependency weights extracted from: [†]Pyro AFD discovery (g_3 error) [?], [‡]HoloClean probabilistic inference [?], [§]FastDD (DD support/confidence) [?], [*]Manual specification.



(b) Average runtime and space overhead

Figure 5: Evaluation of demand-driven erasures

and rule strengths are derived from confidence and selectivity. Explicit business rules and FDs are included where available.

Metrics. We report: (i) **Deletion time**: end-to-end runtime including hypergraph construction, zone extraction, and sampling; (ii) **Mask size** ($|M|$): number of additional cells deleted beyond the target; (iii) **Leakage** ($\mathcal{L}$ as defined in Equation 4); (iv) **Total paths** ($|\Pi|$): number of inference paths to the target cell; (v) **Paths blocked**: number of inference paths eliminated by a mask.

Baselines. We compare our two mechanisms guaranteeing DiffSealto three classes of methods detailed below. The masks produced by the mechanisms below do not reason about probabilistic dependencies nor deletion-pattern leakage — they treat inference as either being present or blocked. Thus, all the mechanisms below assigns weight one to each dependency: (i) **DelAll**: The deterministic inference-blocking mask that removes *all* cells (as in [2, 11]) required to block every extracted inference path. This serves as a leakage-minimizing reference point, but often deletes excessively; (ii) **DelOne**: The deterministic inference-blocking mask that removes one cell for each dependency. This strategy is often used in the context of inference-control in settings related to access control [33]. (iii) **DelMin**: We adapt the deterministic rule-blocking strategy used in meaningful erasure that minimizes the number of deletions needed using ILP [9].

Workload. Given the lack of suitable deletion benchmarks [37], we evaluate demand- and retention-driven erasures separately, as well as varying combinations of each.

8.2 Deletion Overheads

8.3 Leakage Analysis

8.4 Ablation Studies

9 RELATED WORK

The problem of privacy-preserving non-deterministic data deletion with meaningful guarantees on the shift of the prior of an adversary sits at the inter-section of several research areas, each addressing complementary but insufficient aspects of the challenge.

Differential privacy. Classical differential privacy (DP) [12] defines indistinguishability under a tuple-level neighboring relation whose semantics become subtle when data exhibits correlations or constraints. This has motivated semantic generalizations such as Pufferfish [24] and Blowfish [19], which specify secrets and correlations through explicit world models or policy graphs, and Dependent Differential Privacy (DDP) [28], which quantifies privacy loss under correlated data via dependence coefficients. Our framework differs by incorporating correlations directly into a *cell-level* adjacency derived from a dependency hypergraph, and by applying DP to the *choice of deletion mask.* This contrasts with prior DP systems for SQL engines [21, 26], federated querying [3], streaming [38], update-pattern hiding [20], and inconsistency estimation [31], which add noise to query outputs or access patterns. We instead ensure that the deletion strategy itself is insensitive to the true value across all constraint-consistent worlds.

Deletion propagation and meaningful erasure. Classical view-update and deletion propagation frameworks [4, 25] study how deletions must cascade under logical dependencies. ILP-based propagation [29] seeks to minimize side effects on source data and derived views, but

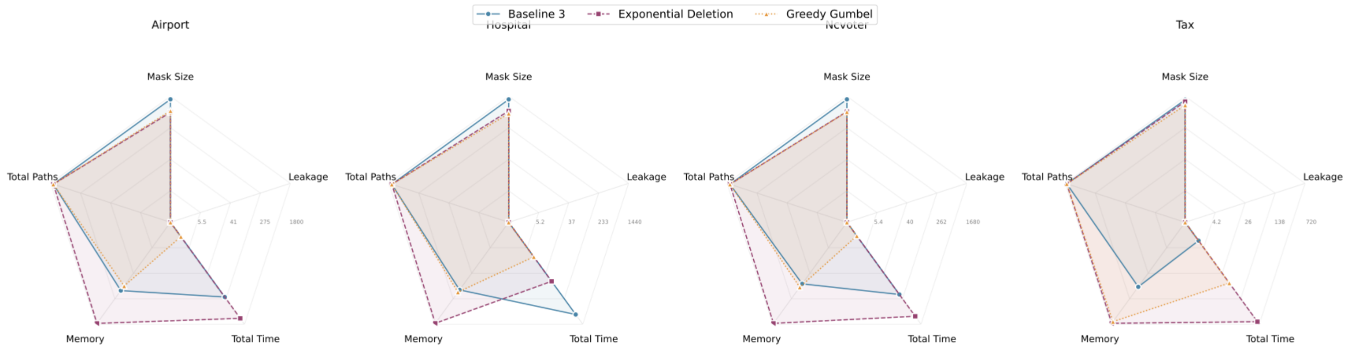


Figure 6: Effect of inference complexity. Star plots compare methods on five axes: mask size $|M|$, leakage L, deletion time, memory, and number of paths $|\Pi|$. Lower is better for time, $|M|$, and L; higher is better for paths blocked.

does not address inference prevention and are not practically feasible due to long convergence time or high deletion volume . Graph-based deletion methods [2, 11] ensure correctness via ownership-based cascades but typically over-delete due to coarse dependency abstractions. P2E2 [9] formalizes meaningful erasure as preserving insertion-time inferability and uses deterministic mechanisms over Relational Dependency Rules (RDRs) where inference is either enabled or blocked, two extremes. These works do not incorporate probabilistic dependencies, adversarial priors, or guarantees on how deletion patterns may shift an adversary's posterior.

Privacy compliance systems. Practical compliance systems [8] automate erasure operations but offer limited semantic privacy guarantees. K9db [2] enforces declarative ownership-based deletion, and adaptations to query languages [36] specify retention and erasure policies. These systems lack formal protections against inference by adversaries who observe deletion outcomes or combine auxiliary information with residual data.

Machine unlearning. Machine unlearning aims to remove the influence of training data from learned models through retraining, gradient adjustments, or data removal [5, 16, 17]. These guarantees apply to model parameters. Our approach operates at the data layer, ensuring differential indistinguishability of deletion mechanisms *before* model training occurs. The two approaches are complementary: DP-style deletion prevents leakage through the visible database state, while unlearning prevents leakage through downstream models.

10 FUTURE WORK AND CONCLUSION

Under the assumption that inference attempts through different final edges are conditionally independent, $\mathcal{L}$ represents the exact probability that at least one channel succeeds. When channels are positively correlated (e.g., they rely on overlapping knowledge), the formula provides a conservative upper bound on the actual inference probability, ensuring our privacy analysis errs on the side of caution.

REFERENCES

[1] Ziawasch Abedjan, Xu Chu, et al. 2015. Detecting data errors: Where are we and what needs to be done?. In *VLDB*.

[2] Kinan Dak Albab, Ishan Sharma, Justus Adam, Benjamin Kilimnik, Aaron Jeyaraj, Raj Paul, Artem Agvanian, Leonhard Spiegelberg, and Malte Schwarzkopf. 2023. K9db: Privacy-Compliant Storage For Web Applications By Construction. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. USENIX Association, Boston, MA, 99–116.

[3] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: Efficient SQL Query Processing in Differentially Private Data Federations. In *Proceedings of the VLDB Endowment*, Vol. 12. 307–320.

[4] Peter Buneman, Sanjeev Khanna, and Wang-Chiew Tan. 2002. On Propagation of Deletions and Annotations Through Views. In *Proc. of PODS*. 150–158. https://doi.org/10.1145/543613.543633

[5] Yinzhi Cao and Junfeng Yang. 2015. Towards Making Systems Forget with Machine Unlearning. In *Proc. of IEEE S&P*. 463–480. https://doi.org/10.1109/SP.2015.35

[6] Loredana Caruccio, Vincenzo Deufemia, Felix Naumann, and Giuseppe Polese. 2021. Discovering Relaxed Functional Dependencies Based on Multi-Attribute Dominance. *IEEE Transactions on Knowledge and Data Engineering* 33, 9 (2021), 3212–3228.

[7] M. A. Casanova, R. Fagin, and C. H. Papadimitriou. 1984. Inclusion Dependencies and Their Interaction with Functional Dependencies. *J. Comput. System Sci.* 28, 1 (1984), 29–59.

[8] Vishal Chakraborty, Stacy Ann-Elvy, Sharad Mehrotra, Faisal Nawab, Mohammad Sadoghi, Shantanu Sharma, Nalini Venkatasubramanian, and Farhan Saeed. 2024. Data-CASE: Grounding Data Regulations for Compliant Data Processing Systems. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. Paestum, Italy, 108–115.

[9] Vishal Chakraborty, Youri Kaminsky, Sharad Mehrotra, Felix Naumann, Faisal Nawab, Primal Pappachan, Mohammad Sadoghi, and Nalini Venkatasubramanian. 2025. Meaningful Data Erasure in the Presence of Dependencies. *Proc. VLDB Endow.* 18, 10 (2025), 3435 – 3448. https://doi.org/10.14778/3748191.3748206

[10] E. F. Codd. 1970. A Relational Model of Data for Large Shared Data Banks. *Commun. ACM* 13, 6 (1970), 377–387.

[11] Katriel Cohn-Gordon, Georgios Damaskinos, Divino Neto, Joshi Cordova, Benoît Reitz, Benjamin Strahs, Daniel Obenshain, Paul Pearce, and Ioannis Papagiannis. 2020. DELF: Safeguarding Deletion Correctness in Online Social Networks. In *Proceedings of the 29th USENIX Conference on Security Symposium (SEC)*. USENIX, USA, Article 60, 18 pages.

[12] Cynthia Dwork and Aaron Roth. 2014. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science* 9, 3–4 (2014), 211–407. https://doi.org/10.1561/0400000042

[13] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional Functional Dependencies for Capturing Data Inconsistencies. In *ACM Transactions on Database Systems*, Vol. 33.

[14] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2010. Towards Certain Fixes with Editing Rules and Master Data. In *Proceedings of the VLDB Endowment*, Vol. 3. 173–184.

[15] Chang Ge, Shubhankar Mohapatra, Xi He, and Ihab F. Ilyas. 2021. Kamino: Constraint-Aware Differentially Private Data Synthesis. *Proc. VLDB Endow.* 14, 10 (2021), 1886–1899.

[16] Antonio Ginart, Melody Guan, Gregory Valiant, and James Zou. 2019. Making AI Forget You: Data Deletion in Machine Learning. In *Advances in Neural Information Processing Systems (NeurIPS)*.

[17] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens van der Maaten. 2020. Certified Data Removal from Machine Learning Models. In *Proc. of ICML*. 3832–3842.

[18] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms Through Consistency. *Proc. VLDB Endow.* 3, 1-2 (2010), 1021–1032.

[19] Xi He, Ashwin Machanavajjhala, and Bolin Ding. 2014. Blowfish privacy: tuning privacy-utility trade-offs using policies. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (Snowbird, Utah, USA) *(SIGMOD '14)*. Association for Computing Machinery, New York, NY, USA, 1447–1458. https://doi.org/10.1145/2588555.2588581

[20] Botong Huang, Yuchao Tao, Zhifeng Bao, Hui (Wendy) Wang, and Xuchen Zhou. 2021. DP-Sync: Hiding Update Patterns in Secure Outsourced Databases with Differential Privacy. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*. 691–703.

[21] Noah Johnson, Joseph P. Near, and Dawn Song. 2018. Towards Practical Differential Privacy for SQL Queries. In *Proceedings of the VLDB Endowment*, Vol. 11. 526–539.

[22] Daniel Kifer and Ashwin Machanavajjhala. 2011. No Free Lunch in Data Privacy. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*. 193–204. https://doi.org/10.1145/1989323.1989345

[23] Daniel Kifer and Ashwin Machanavajjhala. 2012. No Free Lunch in Data Privacy. In *Proc. of SIGMOD*. 193–204. https://doi.org/10.1145/2213836.2213863

[24] Daniel Kifer and Ashwin Machanavajjhala. 2014. Pufferfish: A framework for mathematical privacy definitions. *ACM Trans. Database Syst.* 39, 1, Article 3 (Jan. 2014), 36 pages. https://doi.org/10.1145/2514689

[25] Benny Kimelfeld. 2012. A Dichotomy in the Complexity of Deletion Propagation with Functional Dependencies. In *Proc. of PODS*. 191–202. https://doi.org/10.1145/2213556.2213587

[26] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: A Differentially Private SQL Query Engine. In *Proceedings of the VLDB Endowment*, Vol. 12. 1371–1384.

[27] Shulei Kuang, Honghui Yang, Zijing Tan, and Shuai Ma. 2024. Efficient Differential Dependency Discovery. *Proceedings of the VLDB Endowment* 17, 7 (2024), 1552–1564.

[28] Changchang Liu, Supriyo Chakraborty, and Prateek Mittal. 2016. Dependence Makes You Vulnerable: Differential Privacy Under Dependent Tuples. In *NDSS*.

[29] Neha Makhija and Wolfgang Gatterbauer. 2025. Is Integer Linear Programming All You Need for Deletion Propagation? A Unified and Practical Approach. *Proc. VLDB Endow.* 18, 10 (2025), 2667–2680. https://doi.org/10.14778/3742728.3742756

[30] Frank McSherry and Kunal Talwar. 2007. Mechanism Design via Differential Privacy. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 94–103. https://doi.org/10.1109/FOCS.2007.41

[31] Prasang Mohapatra, Arun Kumar, and Benny Kimelfeld. 2025. Differentially Private Estimation of Inconsistency Measures. In *Proceedings of the VLDB Endowment*, Vol. 18. To appear.

[32] Thorsten Papenbrock and Felix Naumann. 2016. Functional dependency discovery: An experimental evaluation of seven algorithms. In *VLDB*.

[33] Primal Pappachan, Shufan Zhang, Xi He, and Sharad Mehrotra. 2022. Don't Be a Tattle-Tale: Preventing Leakages through Data Dependencies on Access Control Protected Data. *Proceedings of the VLDB Endowment (PVLDB)* 15, 11 (2022), 2437–2449.

[34] Marcel Parciak, Stef Weytjens, Niel Hens, Frank Neven, Lieve M. Peeters, and Stijn Vansummeren. 2025. Measuring Approximate Functional Dependencies: A Comparative Study. *The VLDB Journal* 34, 4 (2025).

[35] Eduardo H. M. Pena, Eduardo C. de Almeida, and Felix Naumann. 2019. Discovery of Approximate (and Exact) Denial Constraints. *Proceedings of the VLDB Endowment* 13, 3 (2019), 266–278.

[36] Subhadeep Sarkar and Manos Athanassoulis. 2022. Query Language Support for Timely Data Deletion. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 418–429.

[37] Subhadeep Sarkar, Tarikul Islam Papon, Dimitris Staratzis, and Manos Athanassoulis. 2020. Lethe: A Tunable Delete-Aware LSM Engine. ACM, New York, NY, USA, 893–908. https://doi.org/10.1145/3318464.3389757

[38] Tianhao Wang, Joann Qiongna Chen, Zhikun Zhang, Dong Su, Yueqiang Cheng, Zhou Li, Ninghui Li, and Somesh Jha. 2022. LDP-IDS: Local Differential Privacy for Infinite Data Streams. In *Proceedings of the 2022 International Conference on Management of Data (SIGMOD)*. 2327–2340.

[39] Junjie Zhao, Junshan Zhang, and H. Vincent Poor. 2017. Dependent Differential Privacy for Correlated Data. In *IEEE GLOBECOM*.