Don't Be a Tattle-Tale: Preventing Leakages through Data Dependencies on Access Control Protected Data

Primal Pappachan†‡, Shufan Zhang*, Xi He*, Sharad Mehrotra†

† University of California Irvine ‡ Pennsylvania State University * University of Waterloo

1. Inference Problem

Zip→ State

Role → SalPerHr

	Eid	EName	Zip	State	Role	WorkHrs	SalPerHr
t_1	34	Tina	45678	WA	Student	20	40
t_2	56	Bobby	54321	CA	Faculty	40	200 🔒
t_3	78	Dale	53567	CA	Faculty	40	180
t_4	12	Khan	54321	CA	Staff	30	70

Wid	Eid	DeptName	Salary	
1	34	CS	800 🔒	t_5
2	56	EE	8000	t_6
3	78	CS	7200	t_7
4	12	BIO	2100	t_8

RQ1. How to design an appropriate security notion to capture such leakage model?

RQ2. How to identify "minimal" number of cells to hide to prevent such leakages?

RQ3. How to design an algorithm or system to "efficiently" prevent such leakages?

2. Background

Tuple Representation

$\forall A_i\ Dom(A_i) = \{1,2,3\}$

	A_1	A_2	A_3
t_1	c_1 1	c_2 2	c_3 3
t_2	c_4 1	c_5 2	c_6 3

Database as a collection of cells

Cell Representation

c_1	c_2	c_3	c_4	c_5	c_6
1	2	2	1	2	3
A_1	A_2	A_3	A_1	A_2	A_3

Access Control Policies:

- Policy maps a Cell c_i to either *sensitive or non-sensitive.*
- When a cell is sensitive, its value is replaced with * *(null)*
- Base view where all cells are nulls

Querier Views:

	c_1	c_2	c_3	c_4	c_5	c_6
$V(C)$	1	*	2	1	2	*
$V_0(C)$	*	*	*	*	*	*
	A_1	A_2	A_3	A_1	A_2	A_3

2. Related Work

- Design time prevention → Different inference channels, poor data availability [TKDE'96, CSF'98]
- Query time prevention → Weak security (fully reconstruct the sensitive cell) [TKDE'00]
- Perfect secrecy or randomized algorithm → Poor data utility [SIGMOD'04, ICDE'20]

3. Security Model

Inference channel:

- **Denial Constraints (DCs)**
- Function-based Constraints

$\tilde{\delta}_1 : \neg(Pred_1 \wedge Pred_2 \wedge \cdots \wedge Pred_n)$

$Pred_i = (c_i\ op\ constant)$ or $(c_i\ op\ c_j)$

where op: $\{=, \neq, \leq, \geq, >, <\}$

DC instantiation:

c_1	c_2	c_3	c_4	c_5	c_6
1	2	2	1	2	3
A_1	A_2	A_3	A_1	A_2	A_3

$A_1 \rightarrow A_3$:

$\delta_1 : \neg((c_1 = c_4) \wedge (c_3 \neq c_6))$

Inference function:

$I(c_i|V, \delta_1)$

	c_1	c_2	c_3	c_4	c_5	c_6
$V(C)$	1	*	2	1	2	*
	A_1	A_2	A_3	A_1	A_2	A_3

Full deniability:

For $S_\Delta = \{\delta_i\}, \forall c_i \in C^S$,

$I(c_i|\ V, S_\Delta) = I(c_i|\ V_0, S_\Delta)$

Adversary can learn nothing about the sensitive cells beyond what is given in the base view.

4. Tattle-Tale Condition

$$TTC(\delta, V, c_i) = \begin{cases} \text{True, when all the other predicates (except } Pred(c^*)) \text{ evaluate as True} \\ \text{False, otherwise} \end{cases}$$

Full deniability *is achieved for a sensitive cell in a view if for all relevant dependency instantiations,* ***Tattle-Tale Condition*** *evaluates False.*

Preventing Leakages:

Hide one other cell in δ to hide the *truth value* of another predicate

c_1	c_2	c_3	c_4	c_5	c_6	c_{12}
*	2	2	1	2	*	.....

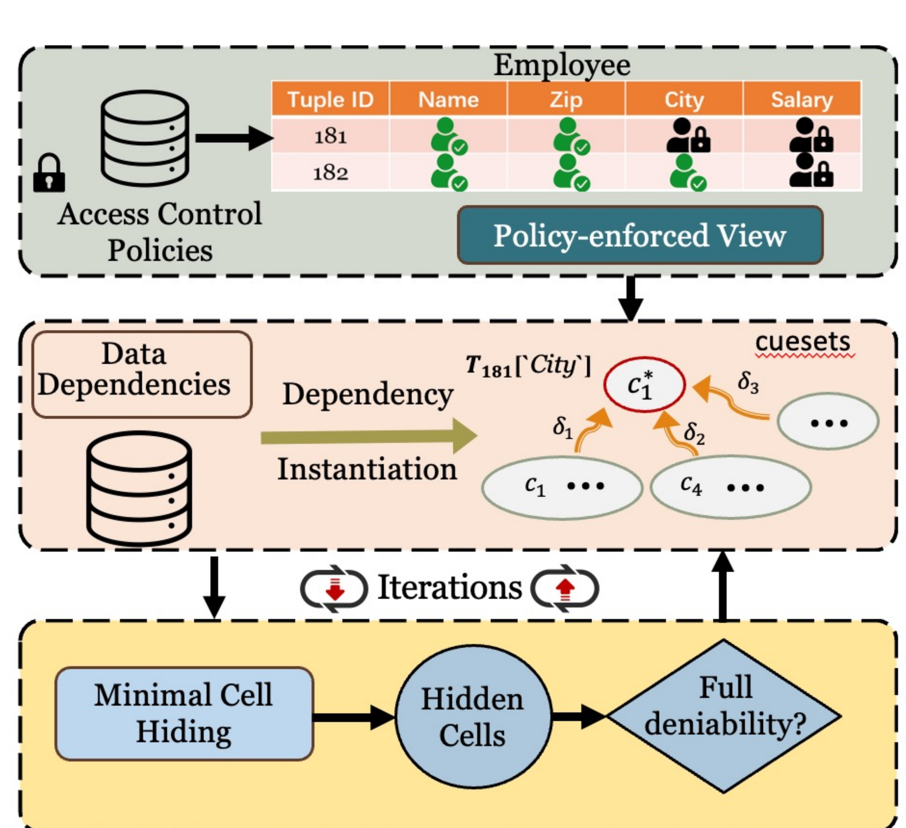
δ_1: $\neg((c_1 = c_4) \wedge (c_2 = c_5) \wedge (c_3 < c_6)$

$\neg((* = 1) \wedge (2 = 2) \wedge (3 < *)$

Unknown True Unknown

Cueset *for* c_1: $\{c_2, c_5, c_3, c_6\}$

5. System Overview



Determine Sensitivity

Inference Detection

Inference Protection

Inference Detection:

- Done naively could result in quadratic blow up
- Optimization: Instantiate only the set of cells the lead to $Pred(c^*)$ evaluating as False

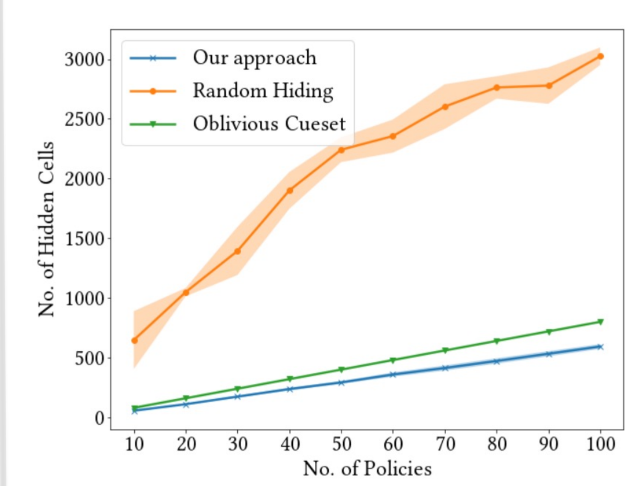
Inference Protection:

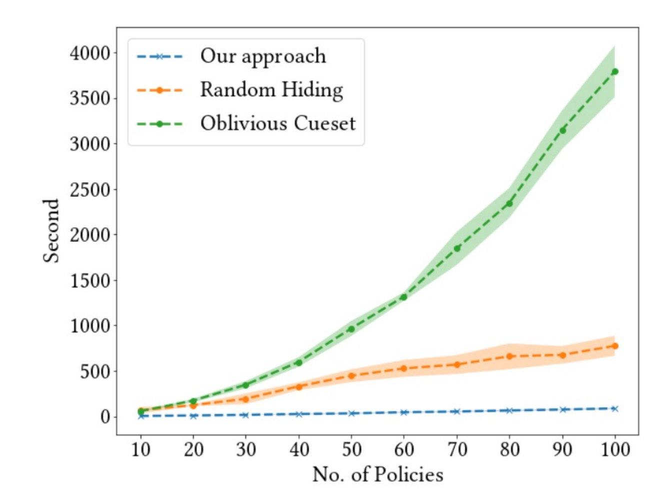
- Perform Minimum Subset Cover (MSC) on the cuesets to find a minimum cover of the subsets
- Repeat until all cuesets corresponding to sensitive cells and hiddenCells are protected

6. Evaluation

Dataset: Tax [ICDE'07]

- 10k tuples, 14 attributes, with 11 dependencies



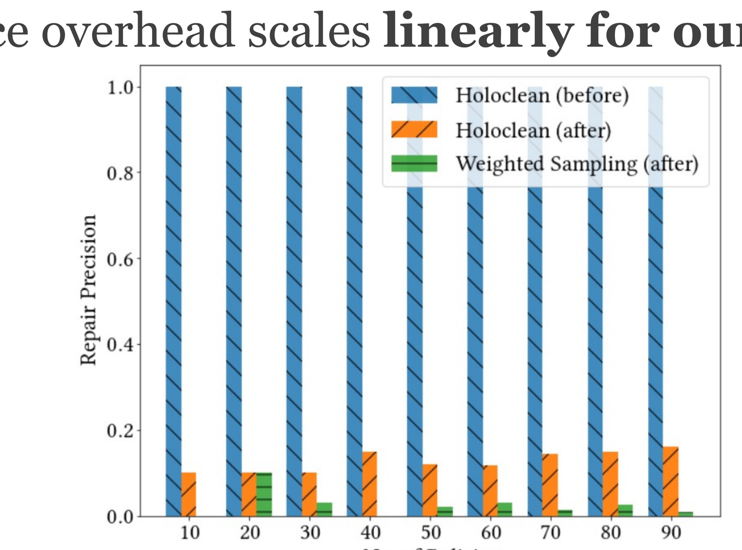


Data Utility:

- With increasing number of sensitive cells, the number of hidden cells increases **linearly**
- Our approach always hides less cells than the baselines

Performance:

- The overhead of our approach is small, compared to the baselines
- When increasing the number of sensitive cells, the performance overhead scales **linearly for our approach**



Holoclean [VLDB'17] as an Adversary:

- Holoclean can reconstruct 100% protected cells before applying our approach
- After applying our approach, it only recovers 10%-15% cells, with marginal improvement over random guess

7. Takeaways

Leakage attack based on two types of data dependencies

- **Denial Constraints**
- Function-based Constraints

Strong security model

- **Tattle-Tale Condition**
- **Full Deniability**
- Relaxing assumptions in the model

End-to-end system

- **Utility, efficiency,** scalability and convergence
- Optimizations to improve performance

Check out the project repo!

VLDB 2022, Sydney, Australia