Don't be a tattle tale: Preventing leakages through data dependencies on access control protected data

Primal Pappachan, Shufan Zhang, Xi He, Sharad Mehrotra
Pennsylvania State University, University of Waterloo, University of California, Irvine

PennState UNIVERSITY OF WATERLOO UCI VLDB 2022

What is a Tattle-Tale?

THE NYT PARENTING NEWSLETTER

Why Your Kid Is Such a Tattletale

There's a developmental reason behind children's obsession with rules.

Give this article 21

Melissa Mathieson

By Jessica Grose

April 28, 2021

wikiHow Mom to do anything...

How to Handle a Tattletale Child

Co-authored by Paul Chernyak, LPC

Last Updated: January 21, 2022 References

To stop a child from tattling, you must understand why they are tattling in the first place. If they need attention, reassess how much attention you've been giving them and demonstrate an interest in them. If your child is struggling with social skills, help them

Tattle-Tale in Databases

	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	200
t_4	12	Khan	54321	CA	Staff	30	70

[State = CA, Role] → [SalPerHr]

Tattle-Tale in Databases

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

Using **[State = CA, Role] → [SalPerHr] and Dale's SalPerHr**
Bobby's SalPerHr can be inferred

Our Goal

Detect and prevent leakages due to data dependencies by hiding "minimal" number of cells

Hide Bobby's State for protecting his SalPerHr

[State = CA, 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	200
t_4	12	Khan	54321	CA	Staff	30	70

Our Goal

Detect and prevent leakages due to data dependencies by hiding "minimal" number of cells

Additionally hide Bobby's Zip for protecting his State

Zip→ State

	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

Extent of Leakage

- Tested on Tax dataset which contains address and tax information of individuals
- 14 attributes and 10 associated dependencies
  - E.g., if two persons live in the same state, the one earning a lower salary has a lower tax rate
- Salary attribute marked as sensitive and tested against a real-world adversary
  - **Holoclean [VLDB2017]** which is a state-of-the-art tool for inferring missing data.

Able to reconstruct the actual values of sensitive cells 100% of the time highlighting the importance of preventing leakages through dependencies

Prior Work

Design time prevention

[Delugach&Hinke, TKDE'96], [Yip&Levitt, CSF'98], etc.

Poor data availability.

Weak security model.

[Brodsky et al, TKDE'00]

Query time prevention

Randomized algorithms (DP/OSDP)

[Kotsogiannis et al, ICDE'20]

Suppresses too many cells.

Not practical for query answering

[Miklau & Suciu, SIGMOD'04]

Perfect secrecy on views

None of the prior works have studied leakage on sensitive data due to data dependencies with strong security guarantees and practical utility.

Main Contributions

***Covered in this presentation**

**Refer to the full paper

Formalizing leakage attack based on two types of data dependencies

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

Defining a security model

- **Tattle-Tale Condition for Leakage Detection**
- **Full Deniability**
- Relaxation of the assumptions in the model

Developing algorithmic solutions to implement security model

- With focus on **Utility, Efficiency**, and Convergence
- Optimizations to improve performance
- **Evaluated on 2 different datasets**
- End-to-end System implementation in MySQL

Formalizing Leakage Attacks

Access Control Policies mark cells in the database as sensitive

Formalizing Leakage Attacks

Data Dependencies causes the leakage

Expressed in the form of Denial Constraints (DCs)

$$\delta_1^{\sim}: \forall t_i, t_j \neg\big((t_i[A] = t_j[A]\big) \wedge (t_i[B] \neq t_j[B]))$$

Formalizing Leakage Attack

t_1			t_2		
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

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

V_0

c_1	c_2	c_3	c_4	c_5	c_6
*	*	*	*	*	*

Base view

Adversary Infers

$c_6 = \{1, 2, 3\}$

V_1

c_1	c_2	c_3	c_4	c_5	c_6
1	2	3	1	2	*

Adversary Infers

$c_6 = \{1, 2, 3\}$

Formalizing Leakage Attack

c1	c2	c3	c4	c5	c6
1	2	2	1	2	3
A_1	A_2	A_3	A_1	A_2	A_3

V_0

c1	c2	c3	c4	c5	c6
*	*	*	*	*	*

Base view

V_1

c1	c2	c3	c4	c5	c6
1	2	3	1	2	*

$$\delta^{\sim}: A_1 \rightarrow A_3$$

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

Adversary Infers

$$c_6 = \{1, 2, 3\}$$

Adversary Infers

$$c_6 = \{3\}$$

Formalizing Leakage Attack

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

V_0

V_2

View V_2 achieves **Full Deniability**
i.e., adversary is unable to infer
nothing more than the base view V_0

$$\mathrm{I}(c_i \mid V`, \delta) = \mathrm{I}(c_i \mid V_0, \delta)$$
$$\forall c_i \in C^S, \forall \delta \in S_\Delta$$

δ_1: $\mathrm{I}(c_1 = c_4 \wedge c_3 \neq c_6)$

Adversary Infers

$c_6 = \{1, 2, 3\}$

Adversary Infers

$c_6 = \{1, 2, 3\}$

What caused leakage?

Shared View

V_1

1	2	3	1	2	*

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

Tattle-Tale is True when all the other predicates, except the one with the sensitive cell, evaluate as True

$$\neg((1 = 1) \wedge (3 \neq *))$$

True ?????

Truth value of the last predicate must be False in a clean database

$$\therefore c_6 = \{3\}$$

Remember that, in the base view we had $c_6 = \{1, 2, 3\}$

What prevented leakage?

Shared View

V_2

*	2	3	1	2	*

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

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

????? ?????

Either of the predicates could be False

$$c_6 = \{1, 2, 3\}$$

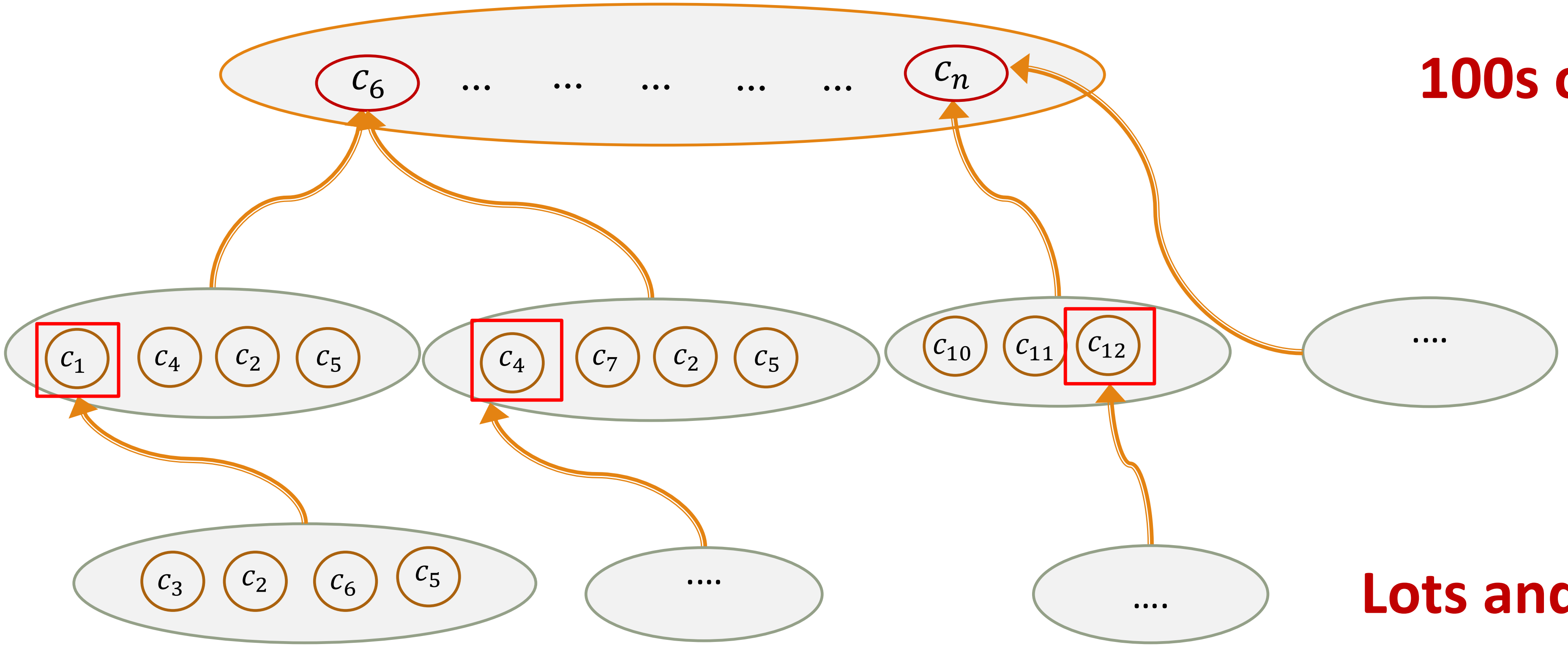
Same as in the base view

Tattle-Tale is False when at least 1 other predicate evaluate as False or Unknown.

Security model

Full deniability *is achieved for a shared view if for all the hidden cells in that view and their dependency instantiations,* ***Tattle-Tale Condition*** *is False.*

The Tail of Tattle-Tales!



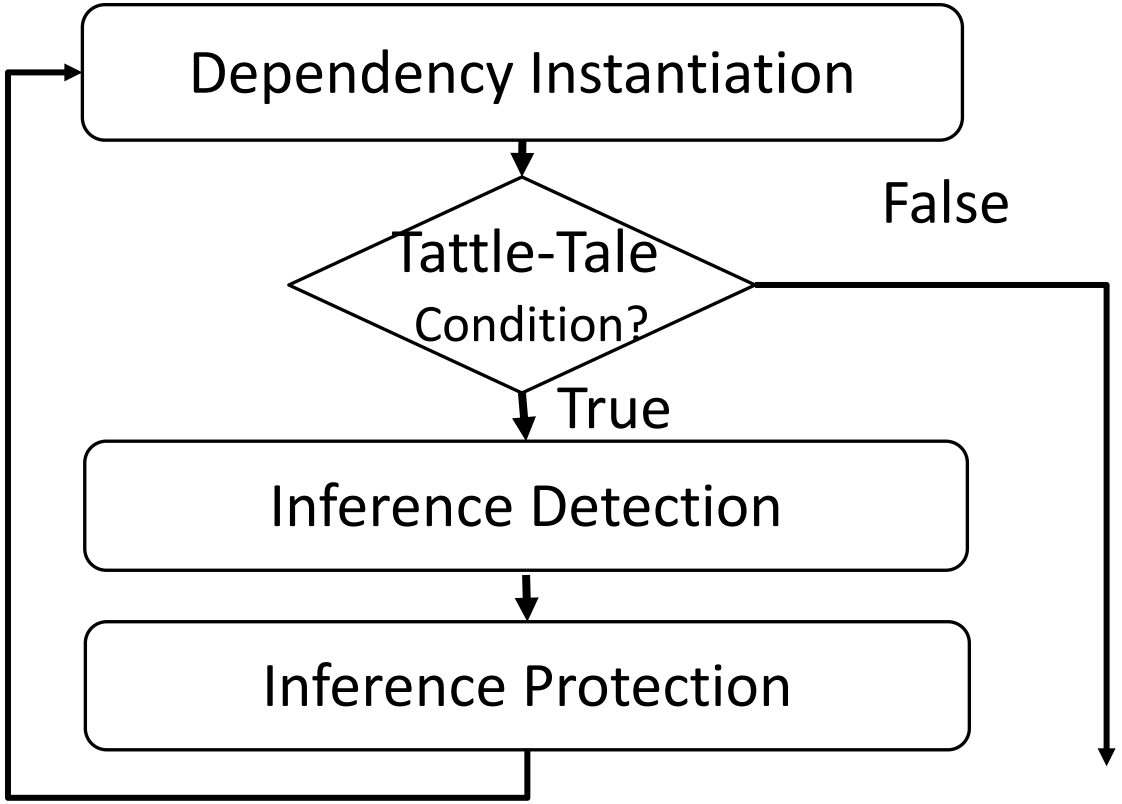
100s of sensitive cells

Millions of cuesets

Lots and lots of hidden cells

Our approach

Input: Database D, Set of Sensitive/Hidden Cells, Set of data dependencies,



Output: View V that achieves full deniability

Step 1: Instantiations…!

Dependency Instantiation

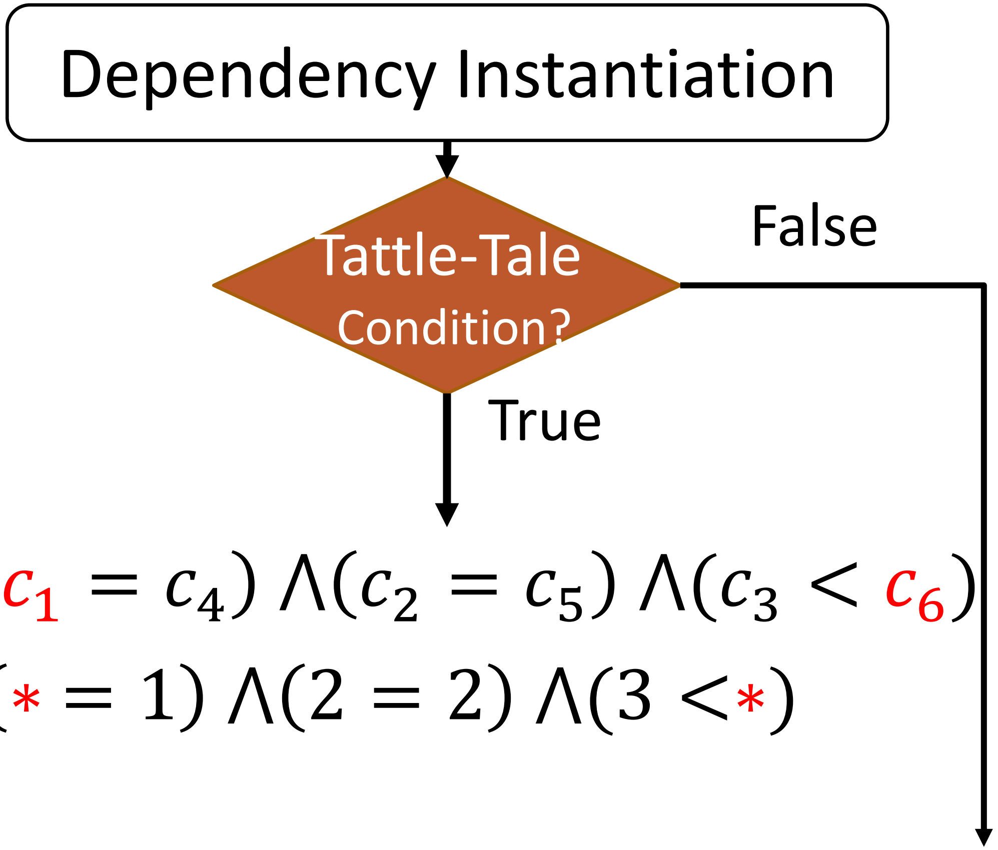
$\delta_1 : \neg((c_1 = c_4) \wedge (c_2 = c_5) \wedge (c_3 < c_6))$
$\delta_2 : \neg((c_7 = c_4) \wedge (c_8 = c_5) \wedge (c_9 < c_6))$

.
.
.
.

$\delta_{|D^2|} : \neg((\ldots \ldots \ldots.) \wedge (\ldots \ldots \ldots..) \wedge (.. \ldots < c_6))$

- For each hidden cell, instantiate all the dependencies
- **Challenge**: In the worst, there are $|D|^2$ instantiations for each sensitive cell
- **Solution**: We converted dependency instantiation operation into an efficient join query to reduce the complexity.

Step 2: Who are the Tattle-Tales?

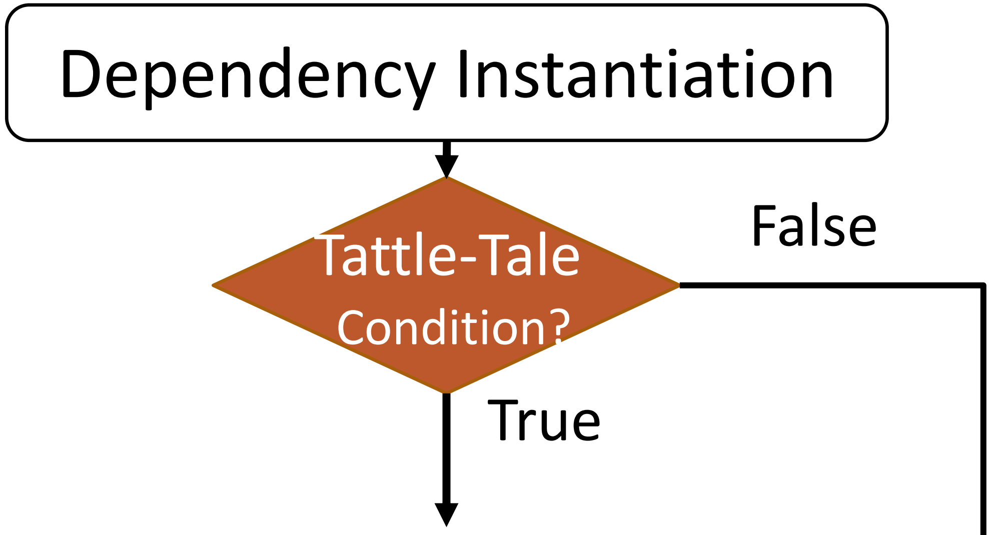


$$\delta_1: \neg((c_1 = c_4) \wedge (c_2 = c_5) \wedge (c_3 < c_6))$$

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

- Check for each hidden cell and their dependency instantiations
- **Termination Condition**: If it returns *False* for all hidden cells and their dependency instantiations, then the view has achieved Full Deniability.

Step 2: Who are the Tattle-Tales?

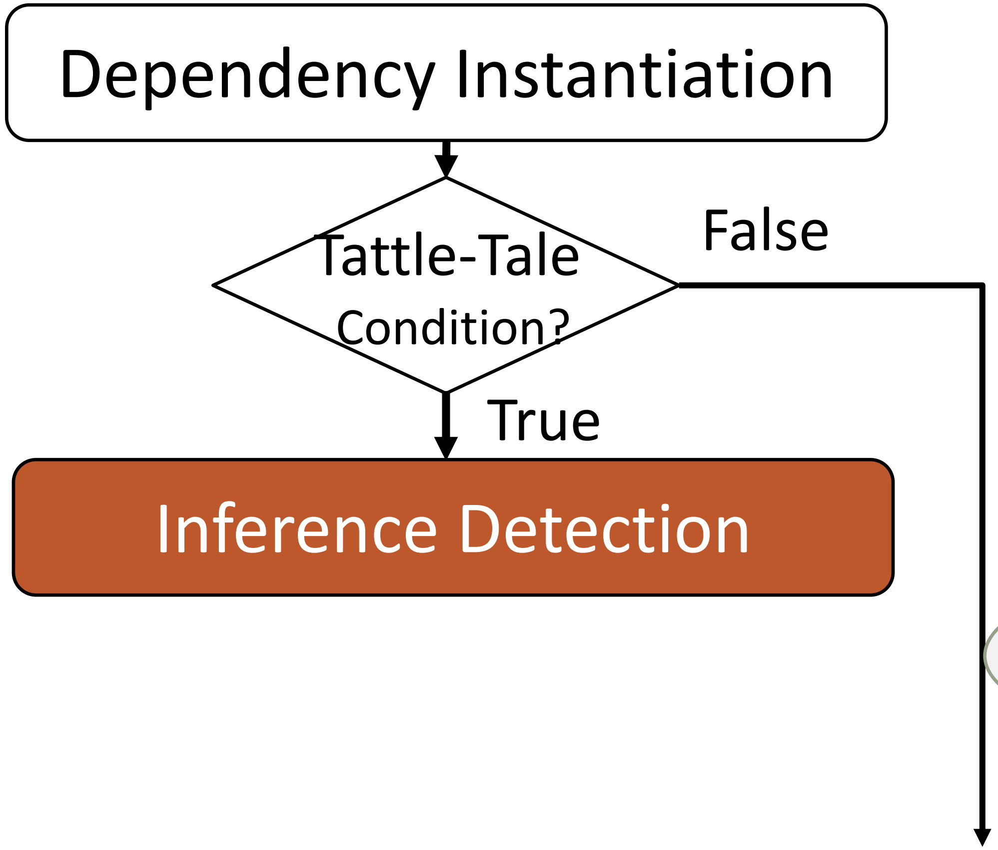


$$\delta_1: \neg((c_1 = c_4) \wedge (c_2 = c_5) \wedge (c_3 < c_6))$$

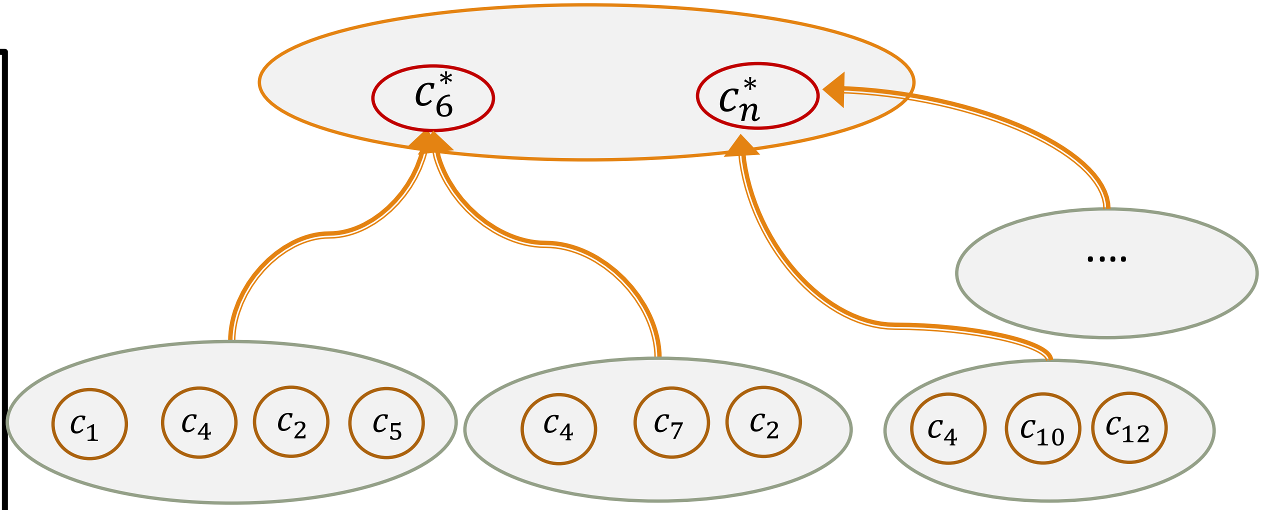
$$\neg((1 = 1) \wedge (2 = 2) \wedge (3 < *))$$

- Check for each hidden cell and their dependency instantiations
- **Termination Condition**: If it returns *False* for all hidden cells and their dependency instantiations, then the view has achieved Full Deniability.
- If it returns *True* for at least 1 of them, then there is leakage

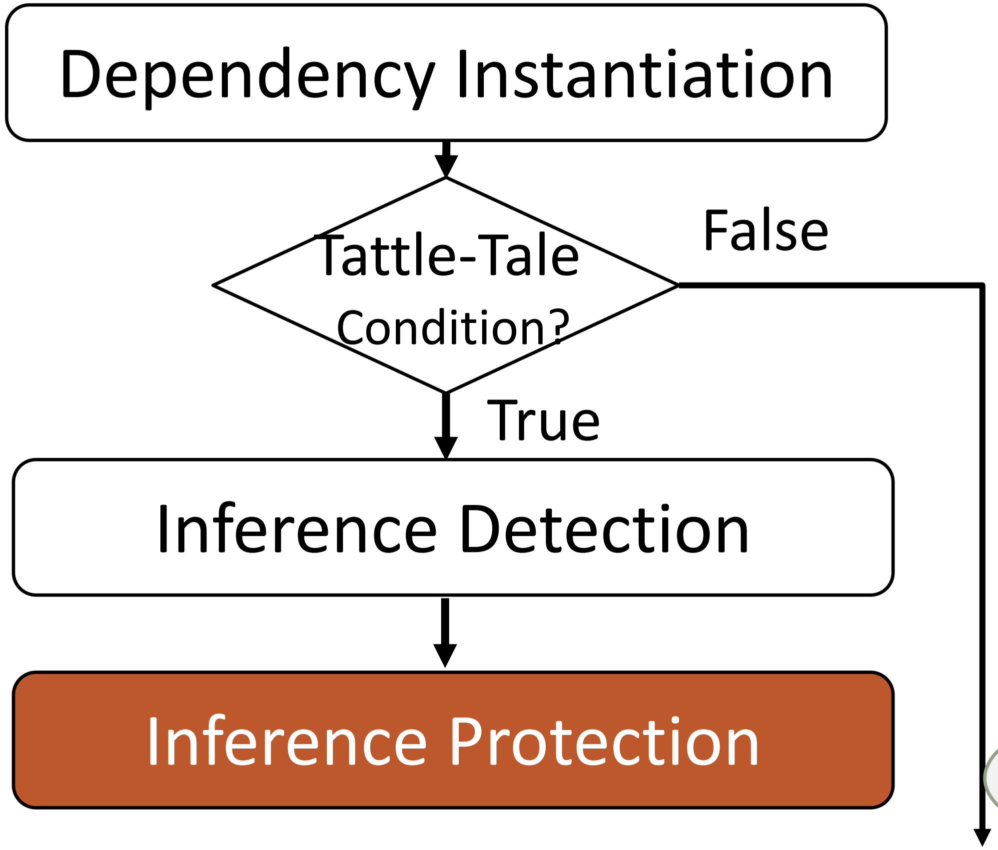
Step 3: Cue them up!



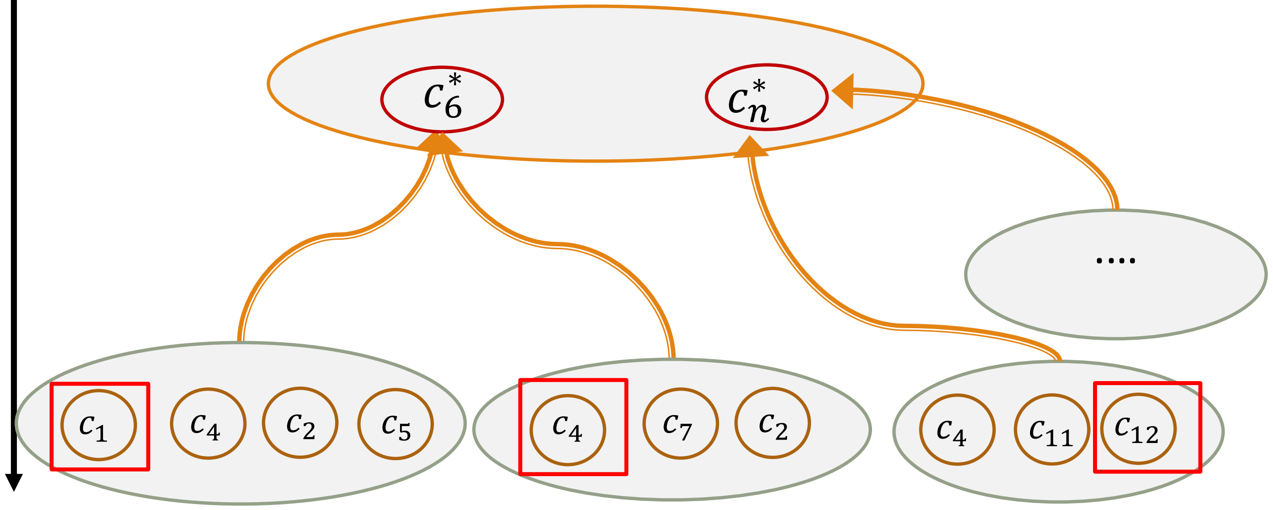
- Outputs cuesets for sensitive cells which satisfy the Tattle-Tale



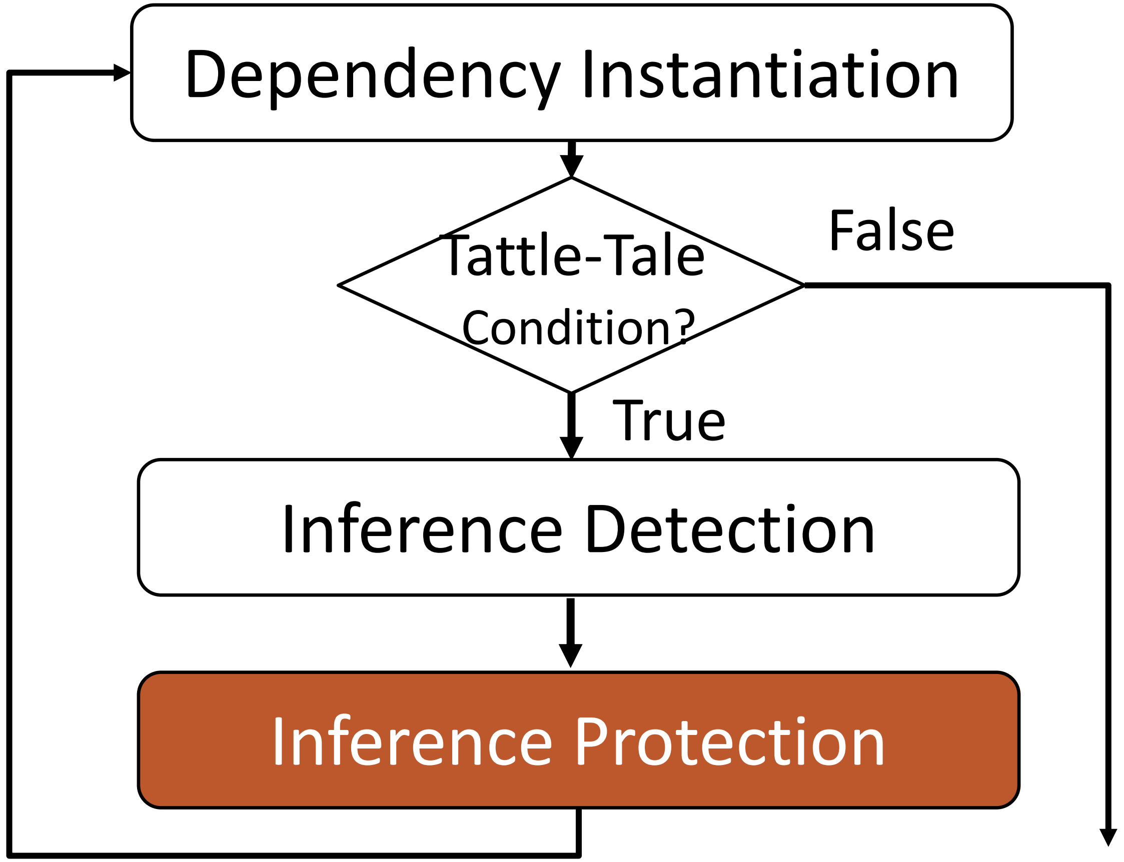
Step 4: Hide yo cells!



- Choose cells to hide from the cuesets
- Random Hiding leads to poor utility (Baseline)



Step 4: Hide yo cells, hide yo cells!



- Choose cells to hide from the cuesets
- **Challenge:** Selecting minimal cells to hide is NP-Hard.
- Use a greedy heuristic based on Minimum Subset Cover
- Run the approach again for newly hidden cells

Experimental Setup

Scan me for source code!

Datasets: Tax dataset [1], (larger) Hospital dataset [2]

Dependencies: Using a data profiling tool [2]. 11 dependencies on Tax dataset and 14 dependencies on *Hospital dataset*

Baselines:

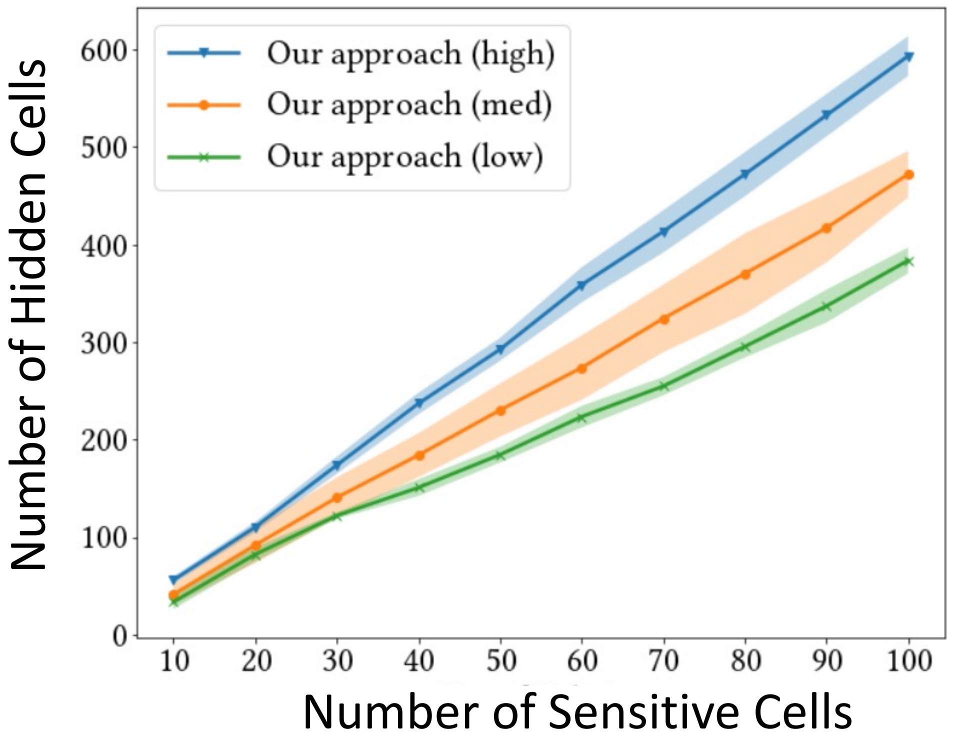
- Random Hiding for Inference Protection
- Oblivious of Tattle-Tale for Inference Detection

End-to-end implementation of the system with steps done at pre-processing

Source code available on Github

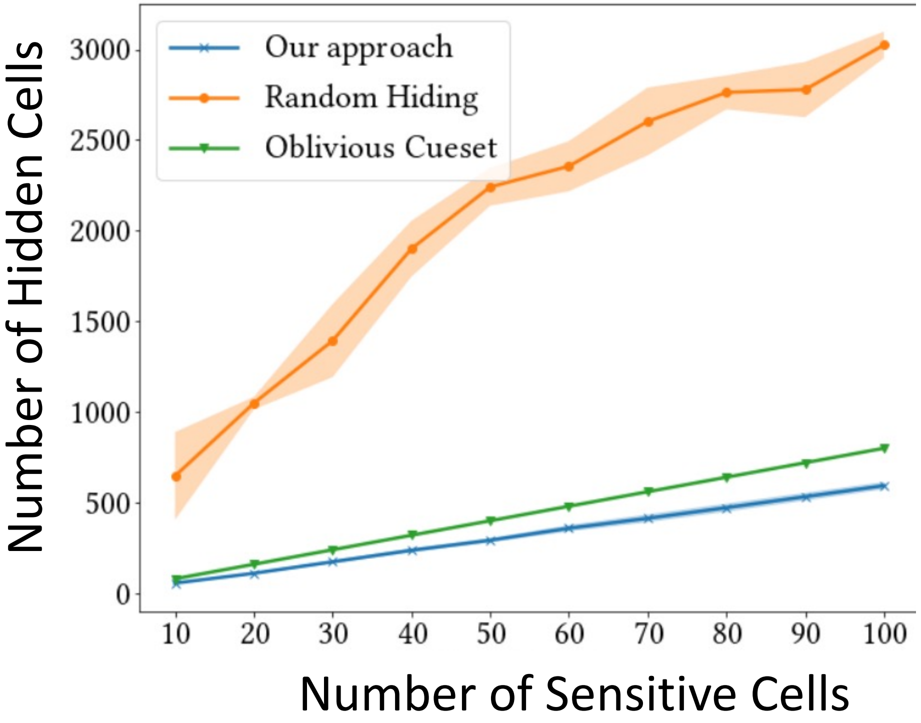
[1] Chu et al. 2013 Discovering Denial Constraints [2] Metanome

Impact of dependencies



If **a sensitive cell participates in more dependencies**, number of hidden cells increases!

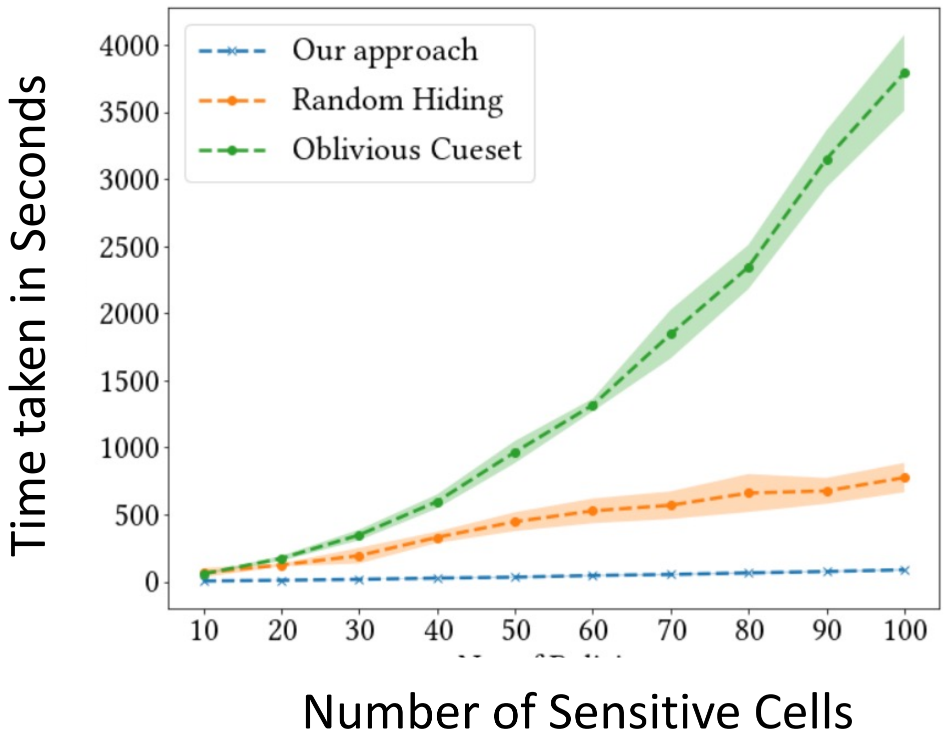
Utility Impact



Number of hidden cells increases linearly with our approach

Number of hidden cells increases exponentially when Random hiding used for Inference Protection!

Performance Impact

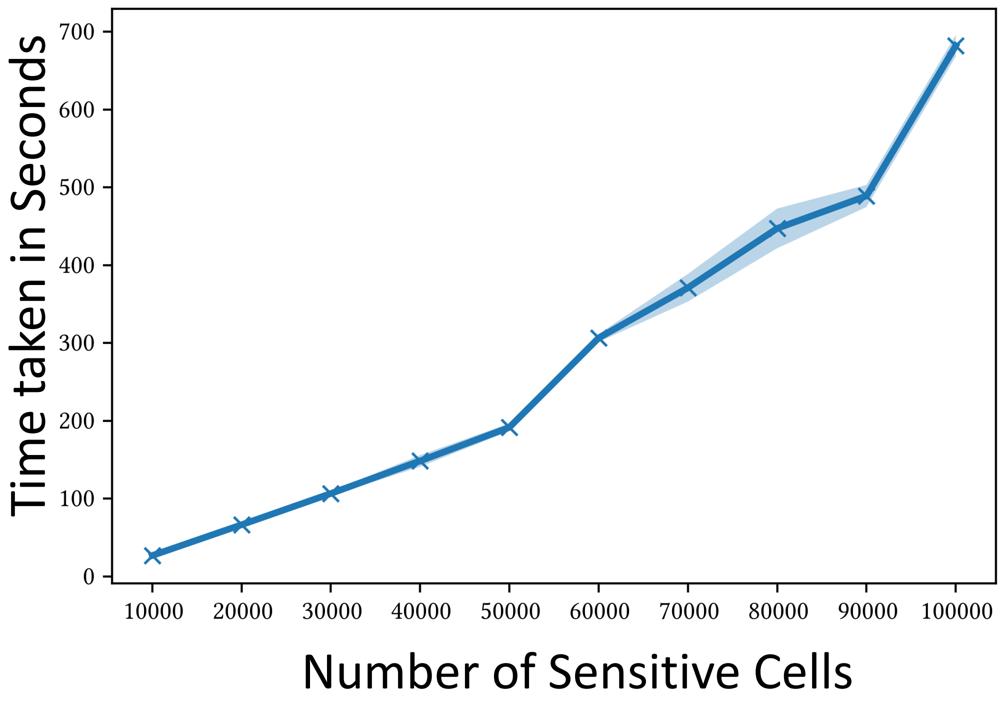


What happens if **when compared against the baselines**?

Overhead minimal in our approach

High overhead when Tattle-Tale condition not used for generating cuesets

Performance Impact



What happens if **when size of the database is increased**?

Our approach scales linearly with respect to size of the database

Takeaways

- Formalized a new type of leakage attacks based on dependencies such as **Denial Constraints and Function-based Constraints**
- Defined a new security model of **Full Deniability (FD)** and **Tattle-Tale Condition** for achieving FD
- Implemented algorithmic solutions for achieving FD on a given view
- Several new research directions
  - Leakage with soft dependencies
  - Combining FD with randomized response methods such as DP, OSDP to release non-sensitive data partially