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
What is a Tattle-Tale?

Why Your Kid Is Such a Tattletale
There is a developmental reason behind children's obsession with rules.

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

<table>
<thead>
<tr>
<th>Eid</th>
<th>EName</th>
<th>Zip</th>
<th>State</th>
<th>Role</th>
<th>WorkHrs</th>
<th>SalPerHr</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>34</td>
<td>Tina</td>
<td>45678</td>
<td>WA</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>t₂</td>
<td>56</td>
<td>Bobby</td>
<td>54321</td>
<td>CA</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>t₃</td>
<td>78</td>
<td>Dale</td>
<td>53567</td>
<td>CA</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>t₄</td>
<td>12</td>
<td>Khan</td>
<td>54321</td>
<td>CA</td>
<td>30</td>
<td>70</td>
</tr>
</tbody>
</table>

[State = CA, Role] → [SalPerHr]
### Tattle-Tale in Databases

<table>
<thead>
<tr>
<th>Eid</th>
<th>EName</th>
<th>Zip</th>
<th>State</th>
<th>Role</th>
<th>WorkHrs</th>
<th>SalPerHr</th>
</tr>
</thead>
<tbody>
<tr>
<td>t_1</td>
<td>34</td>
<td>Tina</td>
<td>45678</td>
<td>WA</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>t_2</td>
<td>56</td>
<td>Bobby</td>
<td>54321</td>
<td>CA</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>t_3</td>
<td>78</td>
<td>Dale</td>
<td>53567</td>
<td>CA</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>t_4</td>
<td>12</td>
<td>Khan</td>
<td>54321</td>
<td>CA</td>
<td>30</td>
<td>70</td>
</tr>
</tbody>
</table>

Using \([\text{State = CA, Role}] \rightarrow [\text{SalPerHr}]\) 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

<table>
<thead>
<tr>
<th>Eid</th>
<th>EName</th>
<th>Zip</th>
<th>State</th>
<th>Role</th>
<th>WorkHrs</th>
<th>SalPerHr</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>34</td>
<td>45678</td>
<td>WA</td>
<td>Student</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>t₂</td>
<td>56</td>
<td>54321</td>
<td>CA</td>
<td>Faculty</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>t₃</td>
<td>78</td>
<td>53567</td>
<td>CA</td>
<td>Faculty</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>t₄</td>
<td>12</td>
<td>54321</td>
<td>CA</td>
<td>Staff</td>
<td>30</td>
<td>70</td>
</tr>
</tbody>
</table>

[State = CA, Role] \(\rightarrow\) [SalPerHr]
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

<table>
<thead>
<tr>
<th>Eid</th>
<th>EName</th>
<th>Zip</th>
<th>State</th>
<th>Role</th>
<th>WorkHrs</th>
<th>SalPerHr</th>
</tr>
</thead>
<tbody>
<tr>
<td>t₁</td>
<td>34</td>
<td>Tina</td>
<td>WA</td>
<td>Student</td>
<td>20</td>
<td>40</td>
</tr>
<tr>
<td>t₂</td>
<td>56</td>
<td>Bobby</td>
<td>CA</td>
<td>Faculty</td>
<td>40</td>
<td>200</td>
</tr>
<tr>
<td>t₃</td>
<td>78</td>
<td>Dale</td>
<td>CA</td>
<td>Faculty</td>
<td>40</td>
<td>180</td>
</tr>
<tr>
<td>t₄</td>
<td>12</td>
<td>Khan</td>
<td>CA</td>
<td>Staff</td>
<td>30</td>
<td>70</td>
</tr>
</tbody>
</table>
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
None of the prior works have studied leakage on sensitive data due to data dependencies with strong security guarantees and practical utility.
Main Contributions

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

*Covered in this presentation
**Refer to the full paper
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)

\[ \delta_1 \sim \forall t_i, t_j \neg ((t_i[A] = t_j[A]) \land (t_i[B] \neq t_j[B])) \]
Formalizing Leakage Attack

<table>
<thead>
<tr>
<th></th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$c_4$</th>
<th>$c_5$</th>
<th>$c_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$\forall A_i \text{Dom}(A_i) = {1, 2, 3}$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$A_3$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$V_0$

<table>
<thead>
<tr>
<th></th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$c_4$</th>
<th>$c_5$</th>
<th>$c_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base view</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

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

$V_1$

<table>
<thead>
<tr>
<th></th>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$c_4$</th>
<th>$c_5$</th>
<th>$c_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adversary Infers</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>*</td>
</tr>
</tbody>
</table>

$V_1$
Formalizing Leakage Attack

\[ V_0 \]

\[
\begin{array}{cccccc}
\ast & \ast & \ast & \ast & \ast & \ast \\
\end{array}
\]

Base view

Adversary Infers

\[ c_6 = \{1, 2, 3\} \]

\[ V_1 \]

\[
\begin{array}{cccccc}
1 & 2 & 3 & 1 & 2 & \ast \\
\end{array}
\]

Adversary Infers

\[ c_6 = \{3\} \]

\[ \delta \sim : A_1 \to A_3 \]

\[ \delta_1 : \neg (c_1 = c_4 \land c_3 \neq c_6) \]
Formalizing Leakage Attack

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

$V_0$

- $V_2$ achieves Full Deniability
- $\forall c_i \in C^S, \forall \delta \in S_\Delta$

**Base view**

Achieves Full Deniability, i.e., adversary is unable to infer nothing more than the base view $V_0$

**Adversary Infers** $c_6 = \{1, 2, 3\}$

$\rho_1: (c_1 = c_4 \land c_3 \neq c_6)$
What caused leakage?

Shared View

\[
V_1
\begin{array}{cccc}
1 & 2 & 3 & 1 & 2 & \ast
\end{array}
\]

\[\delta_1: \neg(c_1 = c_4 \land c_3 \neq c_6)\]

\[\neg((1 = 1) \land (3 \neq \ast))\]

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 \(c_6 = \{1, 2, 3\}\)

**Tattle-Tale** is True when all the other predicates, except the one with the sensitive cell, evaluate as True
What prevented leakage?

Shared View

\[ V_2 \]

\[
\begin{array}{cccccc}
* & 2 & 3 & 1 & 2 & * \\
\end{array}
\]

\[ \delta_1 : \neg (c_1 = c_4 \land c_3 \neq c_6) \]

\[ \neg ((* = 1) \land (3 < *) \]

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...!

- 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.

\[
\delta_1: \neg((c_1 = c_4) \land (c_2 = c_5) \land (c_3 < c_6))
\]
\[
\delta_2: \neg((c_7 = c_4) \land (c_8 = c_5) \land (c_9 < c_6))
\]
\[
\vdots
\]
\[
\delta_{|D|^2}: \neg((\ldots \ldots \ldots) \land (\ldots \ldots \ldots) \land (\ldots < c_6))
\]
Step 2: Who are the Tattle-Tales?

- 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.

\[ \delta_1: \neg((c_1 = c_4) \land (c_2 = c_5) \land (c_3 < c_6)) \]
\[ \neg((* = 1) \land (2 = 2) \land (3 < *)) \]
Step 2: Who are the Tattle-Tales?

- 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

\[ \delta_1: \neg((c_1 = c_4) \land (c_2 = c_5) \land (c_3 < c_6)) \]
\[ \neg((1 = 1) \land (2 = 2) \land (3 < *)) \]
Step 3: Cue them up!

Dependency Instantiation

Tattle-Tale Condition?
False

True

Inference Detection

• 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)

Diagram:
- Dependency Instantiation
  - Tattle-Tale Condition? (False → Inference Detection)
    - True → Inference Protection

Cells:
- $c_1$, $c_4$, $c_2$, $c_5$, $c_6$, $c_n$, $c_7$, $c_2$, $c_4$, $c_{11}$, $c_{12}$
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

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

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