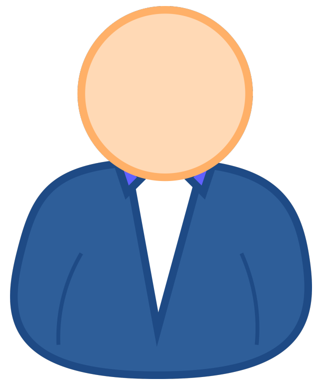
Leakage-Abuse Attacks Against Structured Encryption for SQL

Alex Hoover, Ruth Ng,
Daren Khu, Yao'An Li, Joelle Lim, Derrick Ng,
Jed Lim, and Yiyang Song

Cloud-Hosted SQL Databases

- Modern systems currently outsource sensitive data to cloud providers in the clear (e.g data for medical, financial, sales, human resources, etc)
- Services provide SQL query access to data



```
SELECT * FROM TaxiRides
WHERE Pickup = Hyde Park
```

Taxi_ID	Pickup	Dropoff
124	Hyde Park	Pilsen

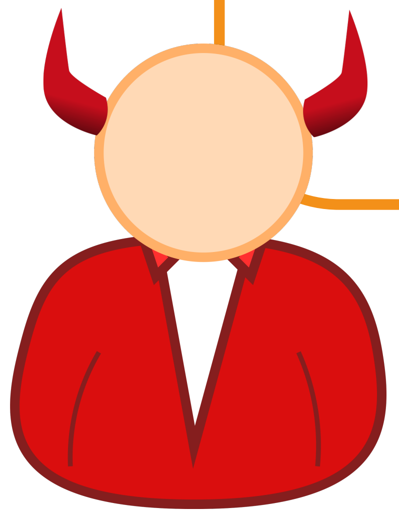
ORACLE DATABASE
Amazon Athena
Microsoft SQL Azure
...

Cloud DB Provider

Taxi_ID	Pickup	Dropoff
429	Bucktown	Pilsen
319	Bucktown	Hyde Park
124	Hyde Park	Pilsen
077	Pilsen	Bucktown

Crime	Location
Robbery	Bucktown
Speeding	Hyde Park
Gambling	Pilsen

...

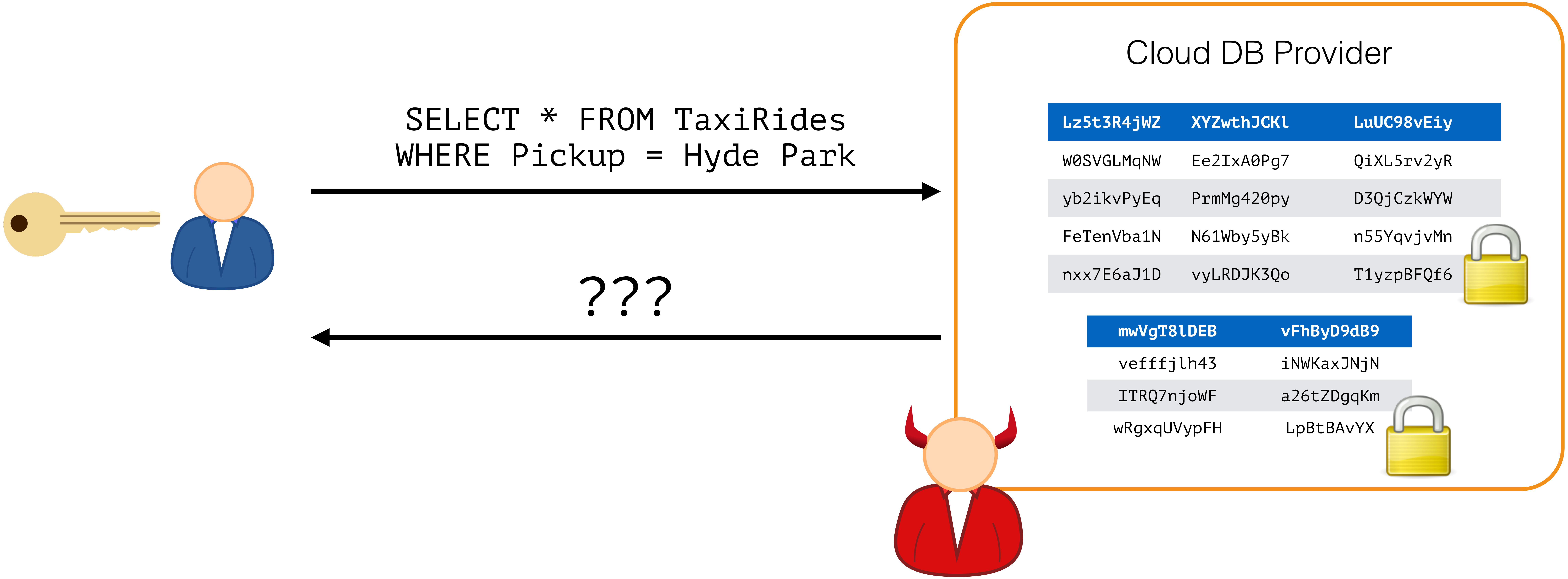


Hackers and insiders can access all data

Client-Side Encryption

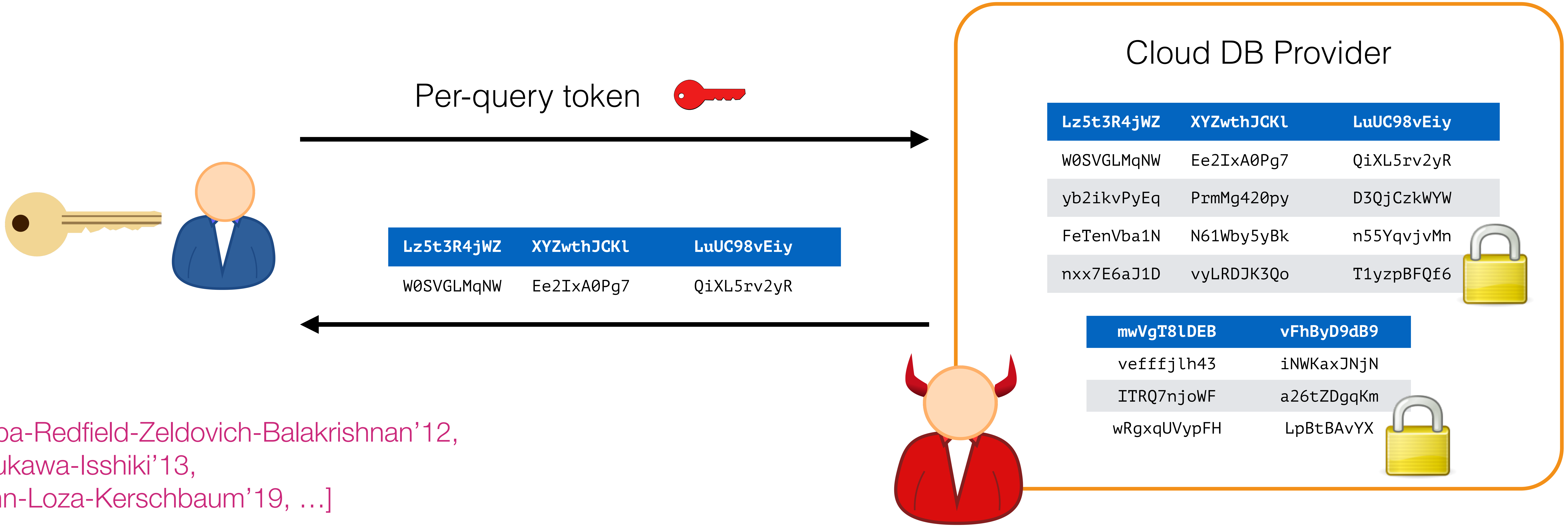
- In **client-side** encryption, key resides at client and is not available to hackers

- But, how does the server <u>process the query</u> with standard encryption?



Queryable Encrypted Databases

- A client can use more complex cryptography to a store and query database
- Best solutions also hide query activity and data from the DB provider
- The supported query types vary depending on the scheme used



[Popa-Redfield-Zeldovich-Balakrishnan'12,
Furukawa-Isshiki'13,
Hahn-Loza-Kerschbaum'19, …]

Structured Encryption

[Song-Wagner-Perrig'00, Curtmola-Garay-Kamara-Ostrovsky'06, Chase-Kamara'10, …]

Structured Encryption (StE) is a symmetric-key scheme with three protocols:

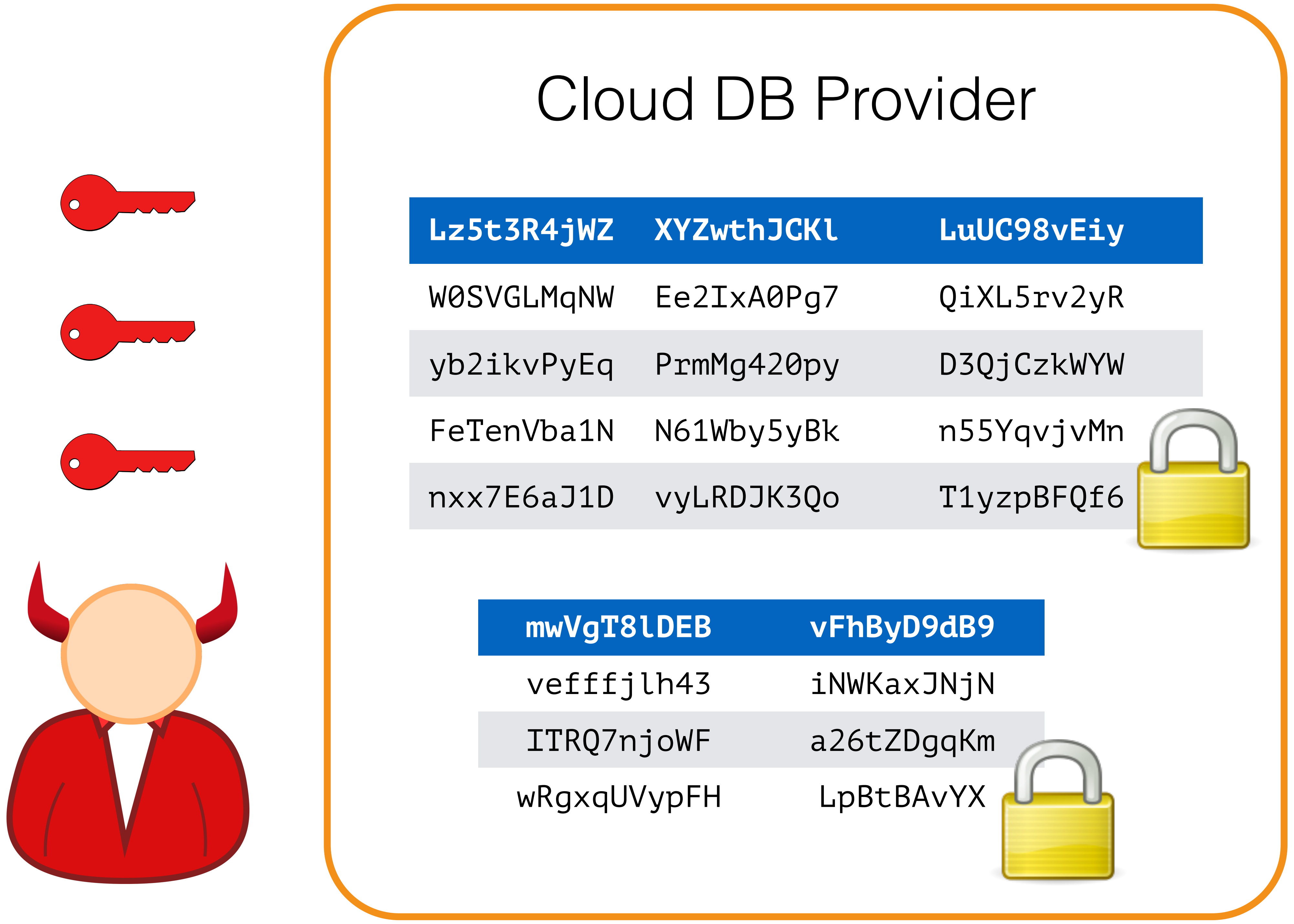
1. **Setup**: Build encrypted data structures under a client-held key
2. **Query-token generation**: Derive a query-specific token to send to server, from client-held key
3. **Encrypted query processing**: Compute the encrypted response, from a token and encrypted data structures, to send to the client

We focus (primarily) on a few schemes which support simple SQL queries such as **selections** and **joins**. [Kamara-Moataz'18, Kamara-Moataz-Zdonik-Zhao'20, Cash-Ng-Rivkin'21,…]

Security for StE Schemes

- Parameterized by a ***leakage profile*** $\mathscr{L}$ that describes what a server can learn by analyzing encrypted data structures and query tokens

- Formally, the "view" of a server can be simulated using output of $\mathscr{L}$ only



- A typical leakage profile $\mathscr{L}$ may include:
  - Bit-size of data
  - Number of tables
  - Number of rows
  - Number of rows matching a query
  - When queries are equal
  - Access pattern of processing
  - …

Leakage-Abuse Attacks (LAAs)

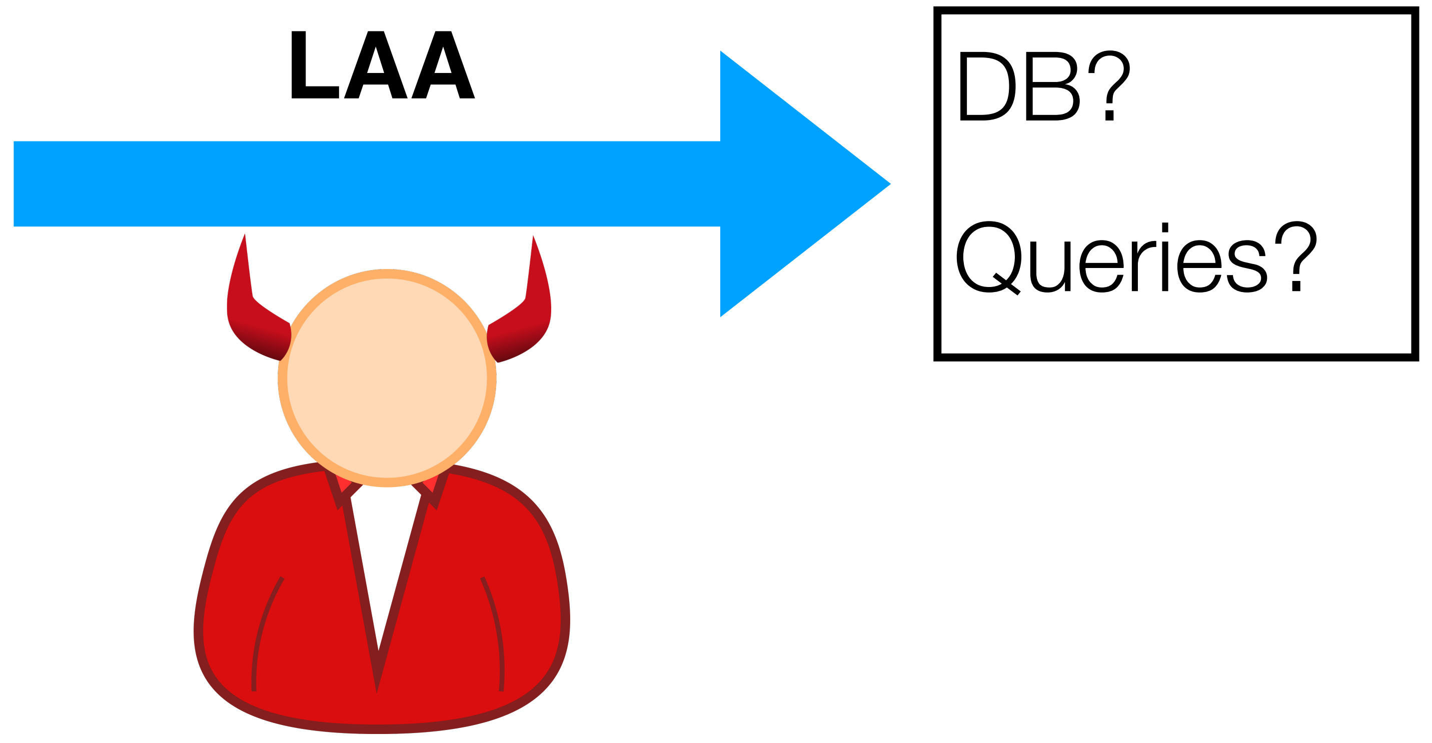
- In real world attacks, we cannot assume the adversary **only has** the leakage
- So, we consider: what other information could an adversary *already know*?

Leakage observed: $\mathscr{L}$(DB, query1, query2, …)

Prior distributions:

ID	Pickup	Dropoff
869	Hyde Park	Hyde Park
192	Hyde Park	Bucktown
214	Pilsen	Lake View
214	Bucktown	River North

Crime	Location
Robbery	Pilsen
Speeding	Lincoln Park
Speeding	Lake View



- For this talk, assume the adversary has **distributional information**
- We model this as access to some previous year's database

[Naveed- Kamara-Wright'15, Bindschaedler-Grubbs-Cash-Ristenpart-Shmatikov'17,…]

New LAAs in Our Paper

Attacking SQL Selection Queries (column equality)

- Generalize prior LAAs against deterministic encryption
- Infer likely client query activity just a few selection queries and distribution

Attacking SQL Join Queries (cross-column equality)

- We identify how SQL join leakages differs depending on the type of join
- Give the first attacks against the the unique join leakage in StE for SQL
- Infer likely plaintext from access pattern and prior distribution

SQL Joins

Taxis **JOIN** Crimes **ON** Taxis.Pickup = Crimes.Location

Taxis

Taxi_ID	Pickup	Dropoff
429	Bucktown	Pilsen
319	Bucktown	Hyde Park
124	Hyde Park	Pilsen
077	Pilsen	Bucktown

Crimes

Crime	Location
Robbery	Bucktown
Speeding	Hyde Park
Gambling	Hyde Park

Taxi_ID	Pickup	Dropoff	Crime	Location
429	Bucktown	Pilsen	Robbery	Bucktown
319	Bucktown	Hyde Park	Robbery	Bucktown
124	Hyde Park	Pilsen	Speeding	Hyde Park
124	Hyde Park	Pilsen	Gambling	Hyde Park

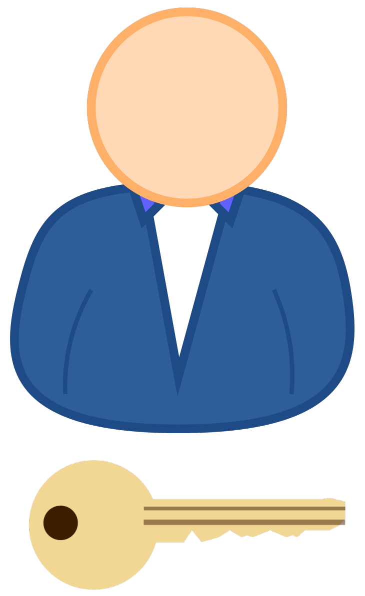
- Amongst all possible ways of pairing a row from Taxis with a row from Crimes, keep those Pickup and Location match
- This is an *inner equi-join* (simple but common kind of join)

StE for SQL Joins

[Kamara-Moataz'18, Kamara-Moataz-Zdonik-Zhao'20, Cash-Ng-Rivkin'21,…]

Taxis **JOIN** Crimes **ON**
Taxis.Pickup = Crimes.Location

(cBQlT9GJ7J,)



Lz5t3R4jWZ	XYZwthJCKl	LuUC98vEiy	mwVgT8lDEB	vFhByD9dB9
W0SVGLMqNW	Ee2IxA0Pg7	QiXL5rv2yR	vefffjlh43	iNWKaxJNjN
yb2ikvPyEq	PrmMg420py	D3QjCzkWYW	vefffjlh43	iNWKaxJNjN
FeTenVba1N	N61Wby5yBk	n55YqvjvMn	ITRQ7njoWF	a26tZDgqKm
nxx7E6aJ1D	vyLRDJK3Qo	T1yzpBFQf6	wRgxqUVypFH	LpBtBAvYX
…	…	…	…	…

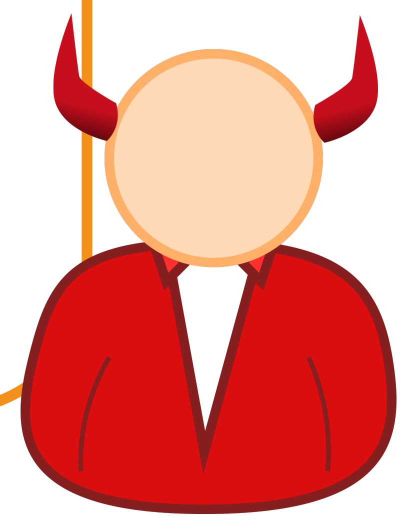
Encrypted Multimap

Key	row_ids
cBQlT9GJ7J	(1,1),(2,1),(3,2),(4,3)
BKi4dwQ04V	jAcDWz8VA4
…	…

row_id	Lz5t3R4jWZ	XYZwthJCKl	LuUC98vEiy
1	W0SVGLMqNW	Ee2IxA0Pg7	QiXL5rv2yR
2	yb2ikvPyEq	PrmMg420py	D3QjCzkWYW
3	FeTenVba1N	N61Wby5yBk	n55YqvjvMn
4	nxx7E6aJ1D	vyLRDJK3Qo	T1yzpBFQf

row_id	mwVgT8lDEB	vFhByD9dB9
1	vefffjlh43	iNWKaxJNjN
2	ITRQ7njoWF	a26tZDgqKm
3	wRgxqUVypFH	LpBtBAvYX

- Client tokens queried list of row pairs to server
- Server learns **all pairs with matching values**
- Server combines the encrypted row pairs and returns them to Client

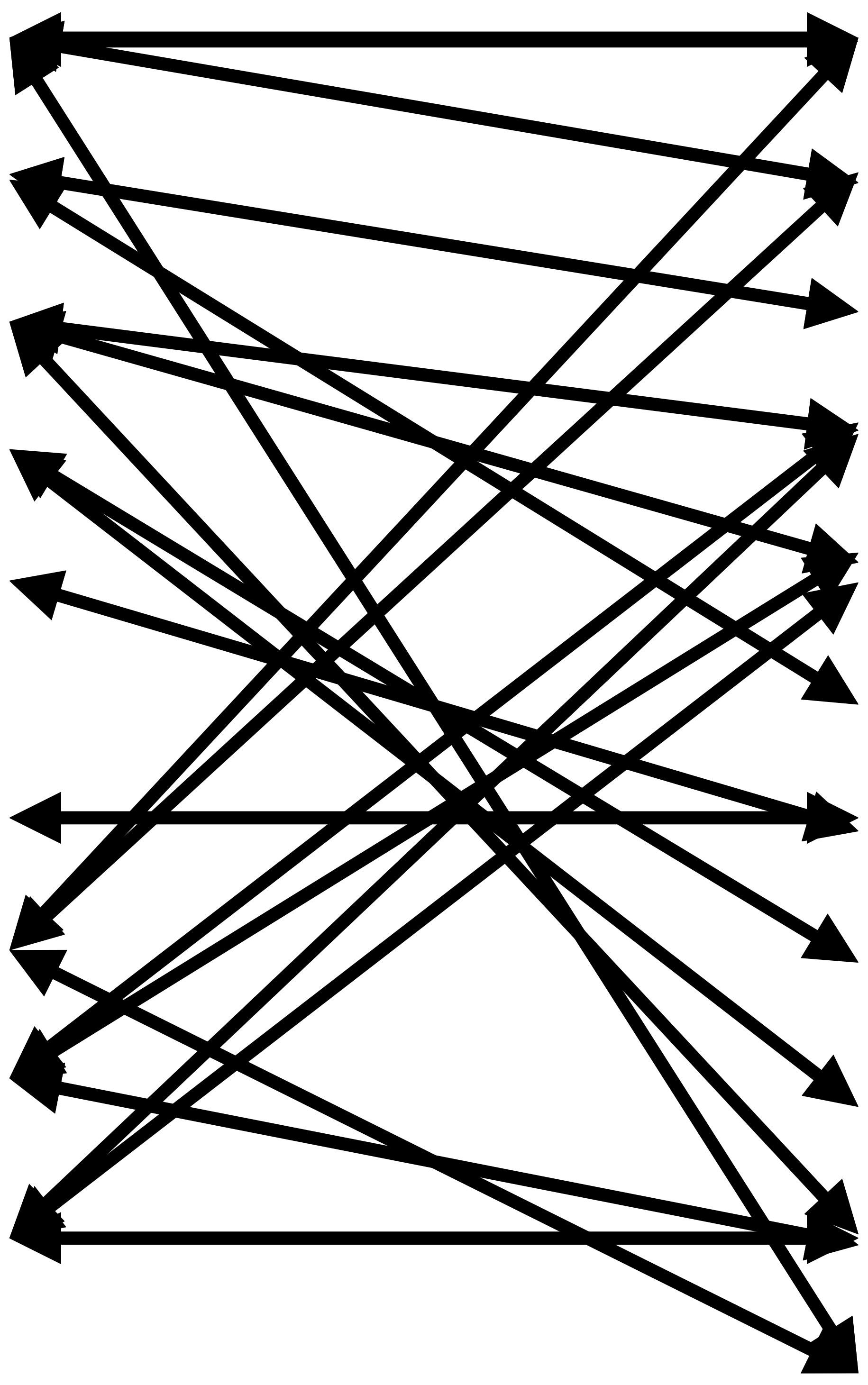


Cloud DB Provider

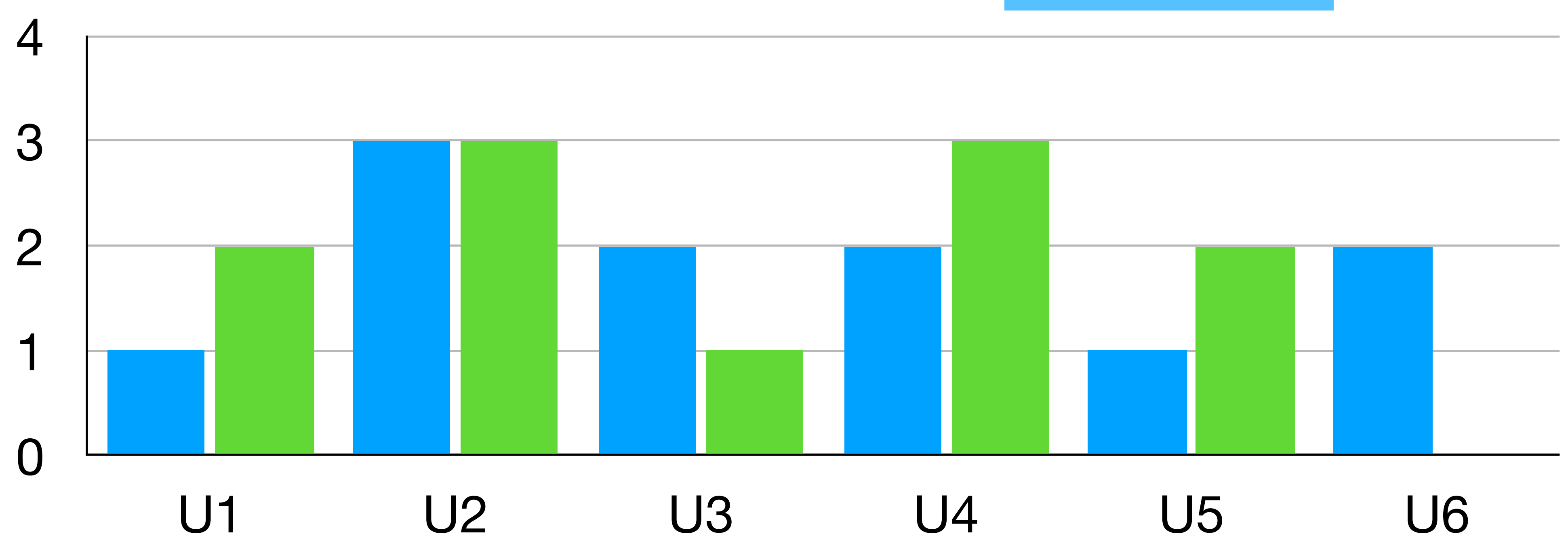
Cross-column Equality

Taxis **JOIN** Crimes **ON**
Taxis.Pickup = Crimes.Location

Crime	Location
vefffjlh43	Unknown 4
ITRQ7njoWF	Unknown 1
wRgxqUVypFH	Unknown 2
Eet6iyFCyg	Unknown 5
6qqKWoAxqj	Unknown 3
OtagOrXXNx	Unknown 6
HJoX4ZlKRQ	Unknown 3
fyb6527yJU	Unknown 4
gNrwXDNrw8	Unknown 2
b6AQts1JKI	Unknown 2
9o0PWRk069	Unknown 6

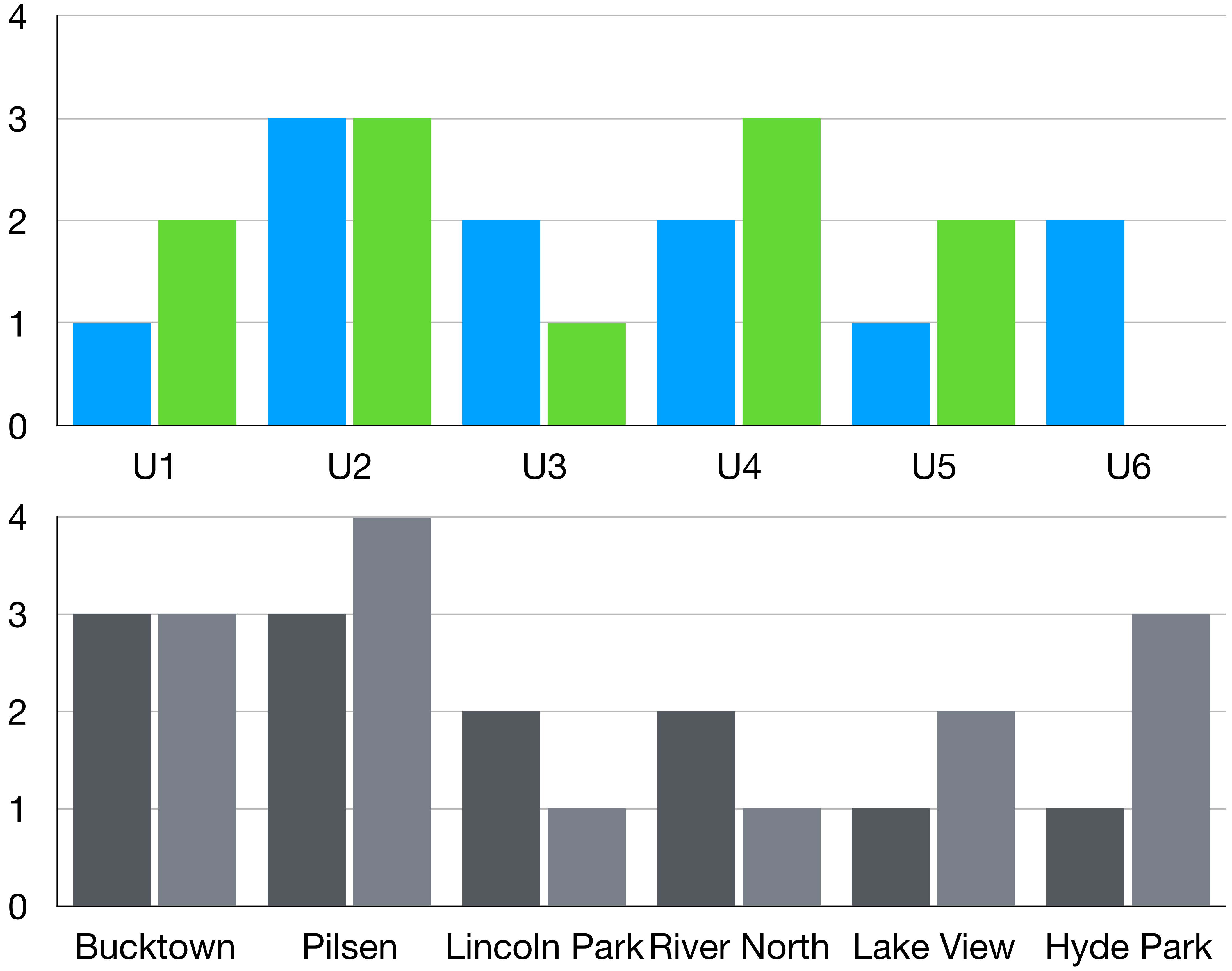


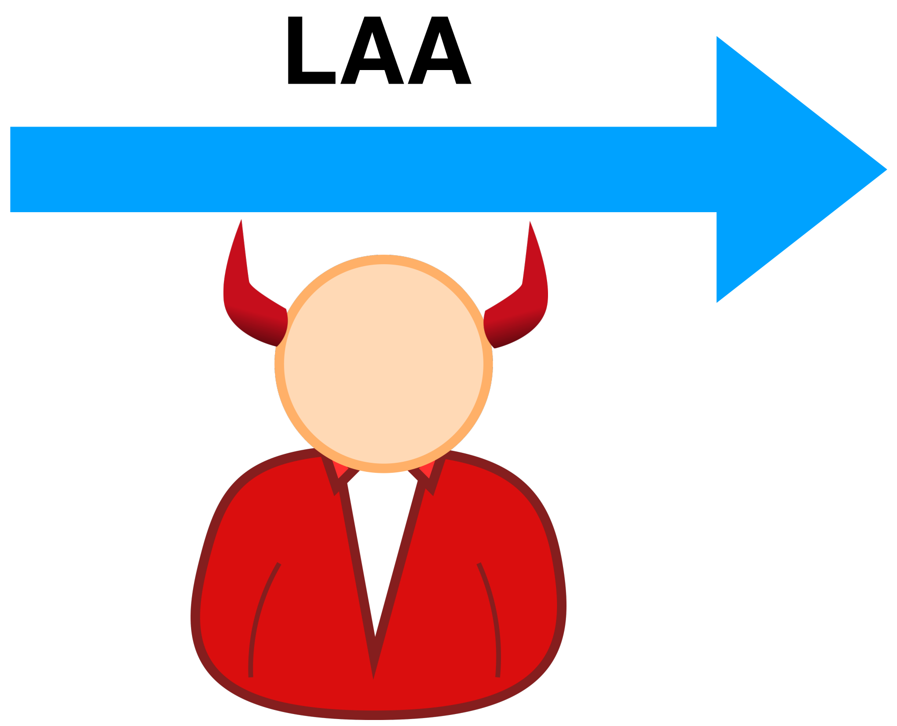
Taxi_ID	Pickup	Dropoff
W0SVGLMqNW	Unknown 4	QiXL5rv2yR
yb2ikvPyEq	Unknown 4	D3QjCzkWYW
FeTenVba1N	Unknown 1	n55YqvjvMn
nxx7E6aJ1D	Unknown 2	T1yzpBFQf6
HMjj07i5aI	Unknown 2	ISexxniaZf
6kS2DHX4tR	Unknown 1	x2cgazIIAX
H1XbIQrE2Y	Unknown 3	MJp7jn6hFu
4aqIyAfhnw	Unknown 5	YdBa3UxcV5
aNRtOQQM7K	Unknown 5	h0C7JDbfE6
NnLojmIXWV	Unknown 2	ZIwGixlz2U
b9GucpDkxG	Unknown 4	k8N2X3KOry



- From equality pairs, adversary can learn the **size of each "equality group"** in *both* tables
- We represent this information with a pair of aligned histograms
- Cross-column leakage appears hard to remove without heavy-weight crypto

Attacks against Join Leakage





Unknown 1 = ?

Unknown 2 = ?

Unknown 3 = ?

Unknown 4 = ?

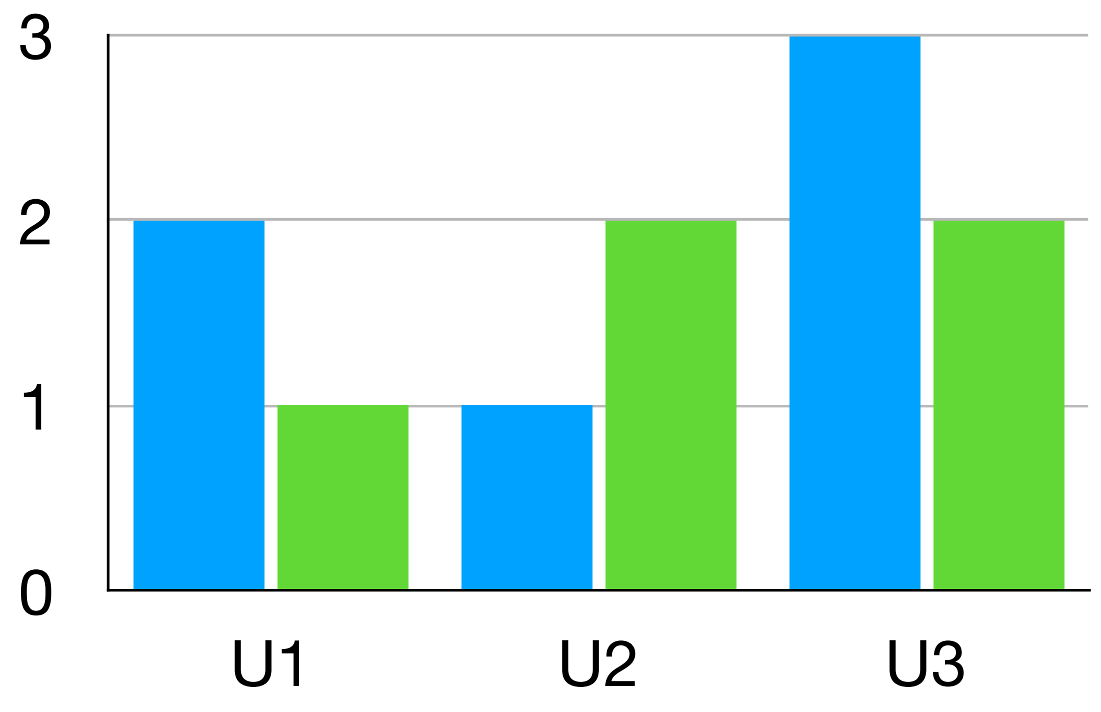
Unknown 5 = ?

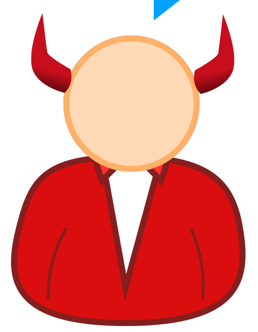
Unknown 6 = ?

- An LAA in this context <u>infers the most likely underlying observed values</u> for each group
- We give **three attacks** with new techniques tailored to this specific leakage

Empirical Evaluation

- We evaluate our attacks on publicly available Chicago data,
    - e.g. Crime, Crash, Taxi, and Rideshare tables
- We simulated the leakage for a variety of possible joins in the data
- The **value recovery rate** is the percent of <u>values correctly identified</u>
- The **row recovery rate** is the precent of <u>rows correctly identified</u>





Unknown 1 = Pilsen
Unknown 2 = Lake View
Unknown 3 = Bucktown

Guess

Unknown 1 = Pilsen
Unknown 2 = Lake View
Unknown 3 = Hyde Park

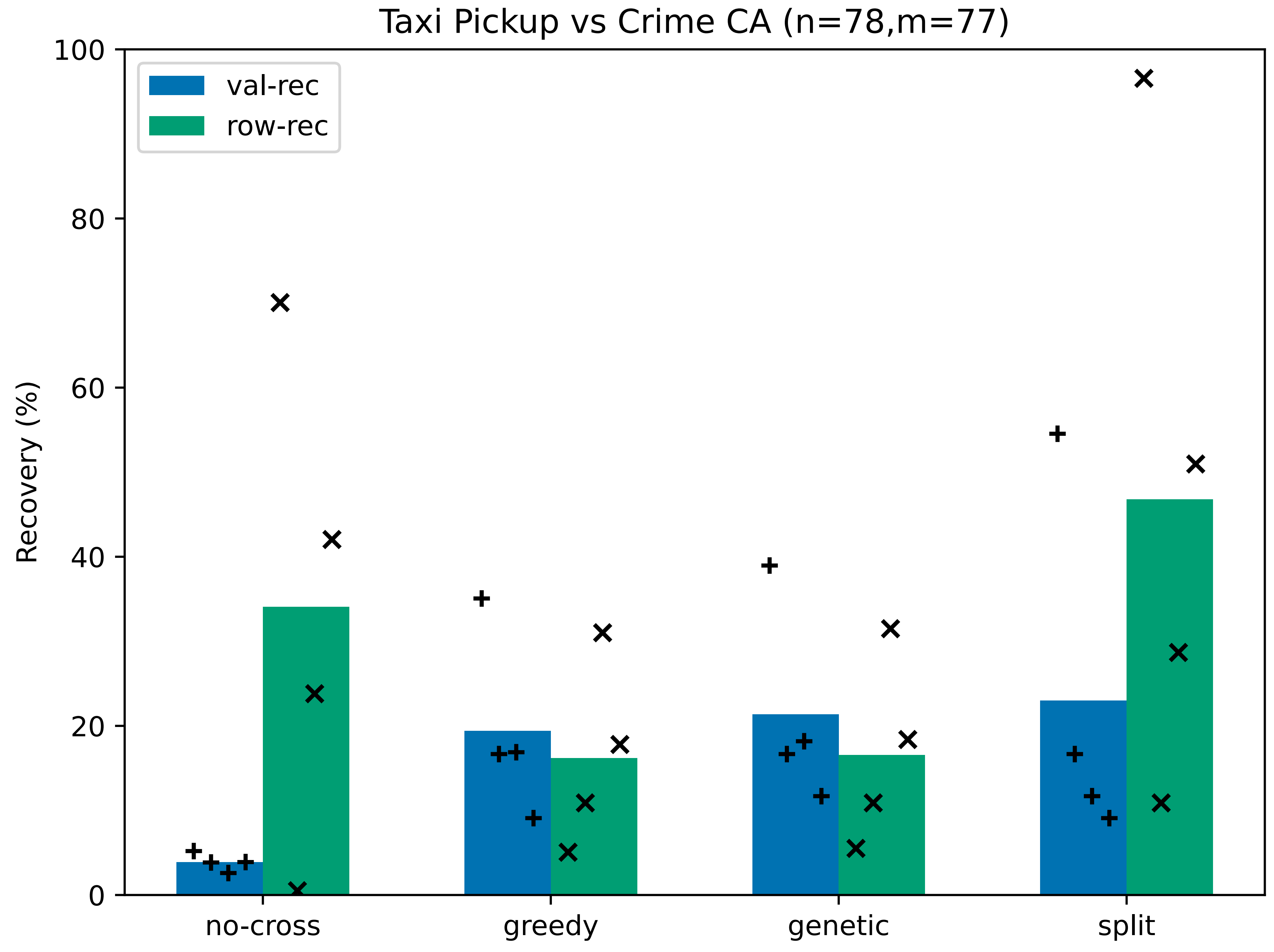
Truth

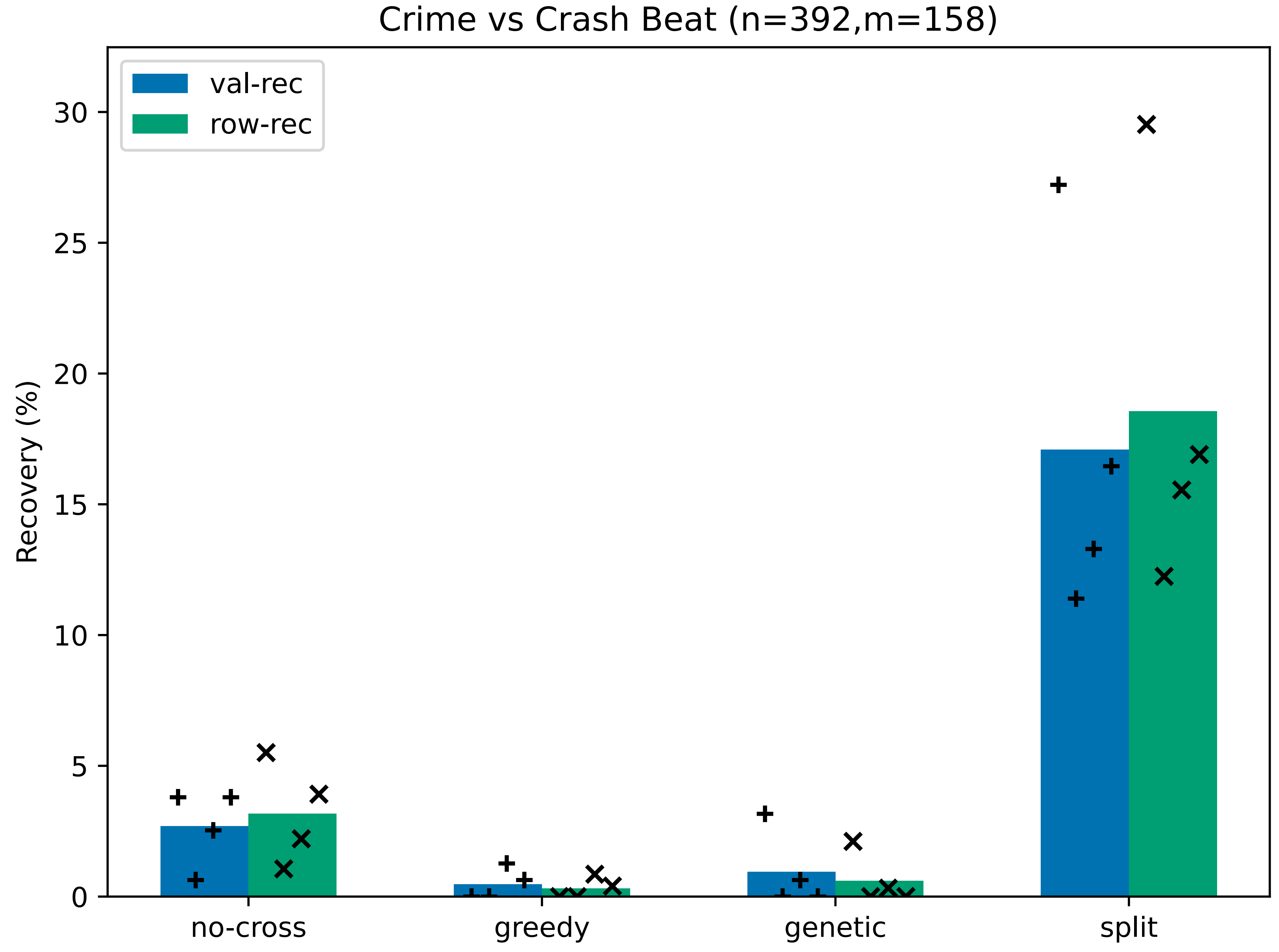
Val-rec: 2/3

Row-rec: 6/11

Join Attack Results

- We tested 3 different cross-column attacks: “greedy,” “genetic,” and “split”
- Also tested optimal “no-cross” attacks which **ignore correlation** between columns
- Found that **using correlation** lead to much higher **high value recovery**
  - These correlation also lead to **higher row recovery**
- Even in our hardest test (Crime vs Crash Beats), our split attack recovered more than 15% of the values and rows





Theoretical Techniques

- Our SQL selection attack generalizes frequency analysis to work without every frequency in the table and prove it is near-optimal
- Analyze the different between different join types
    - Prove that our attacks are optimal against “complete” joins
    - Prove that “incomplete” joins are NP-hard to infer optimally
- We give new optimization algorithms for partitioning sets with respect to the LAA inference objective
- Many other interesting algorithmic ideas to perform these attacks!

Thanks for listening!

Read the paper: **ia.cr/2024/554**

Read about me: **axhoover.com**

Authors:

Alex Hoover (me)
Ruth Ng
Daren Khu
Yao'An Li
Joelle Lim
Derrick Ng
Jed Lim
Yiyang Song

Questions?

Feel free to reach out about any future questions too!

THE UNIVERSITY OF CHICAGO 1890
DSO NATIONAL LABORATORIES