Efficient Searchable Symmetric Encryption for Join Queries

Sikhar Patranabis (IBM Research, India)

Joint work with Charanjit Jutla (IBM Research, USA)

IACR ASIACRYPT 2022

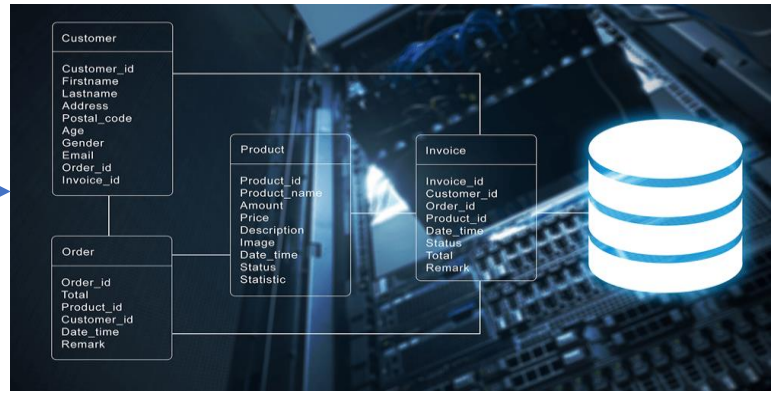
December 07, 2022

Outsourced Storage and Processing

Organization

Outsource

Database

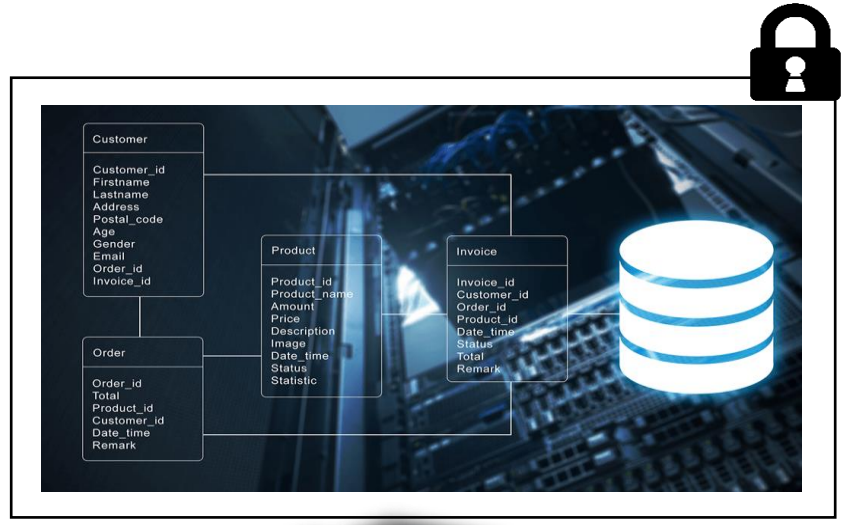


Auditor

Query Request

Result

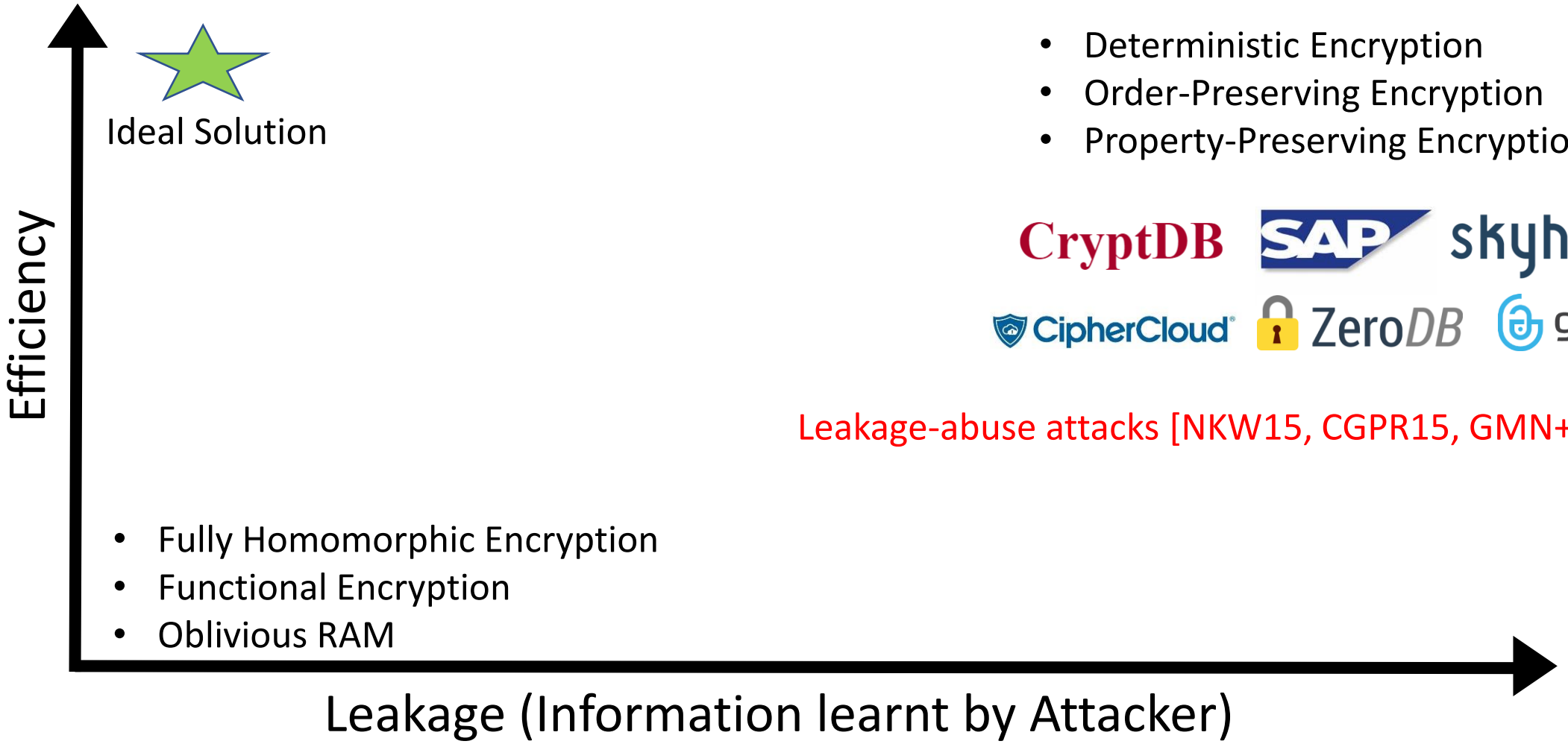
Computing on Encrypted Data



Query Request

Result

Efficiency v/s Leakage: Computing on Encrypted Data

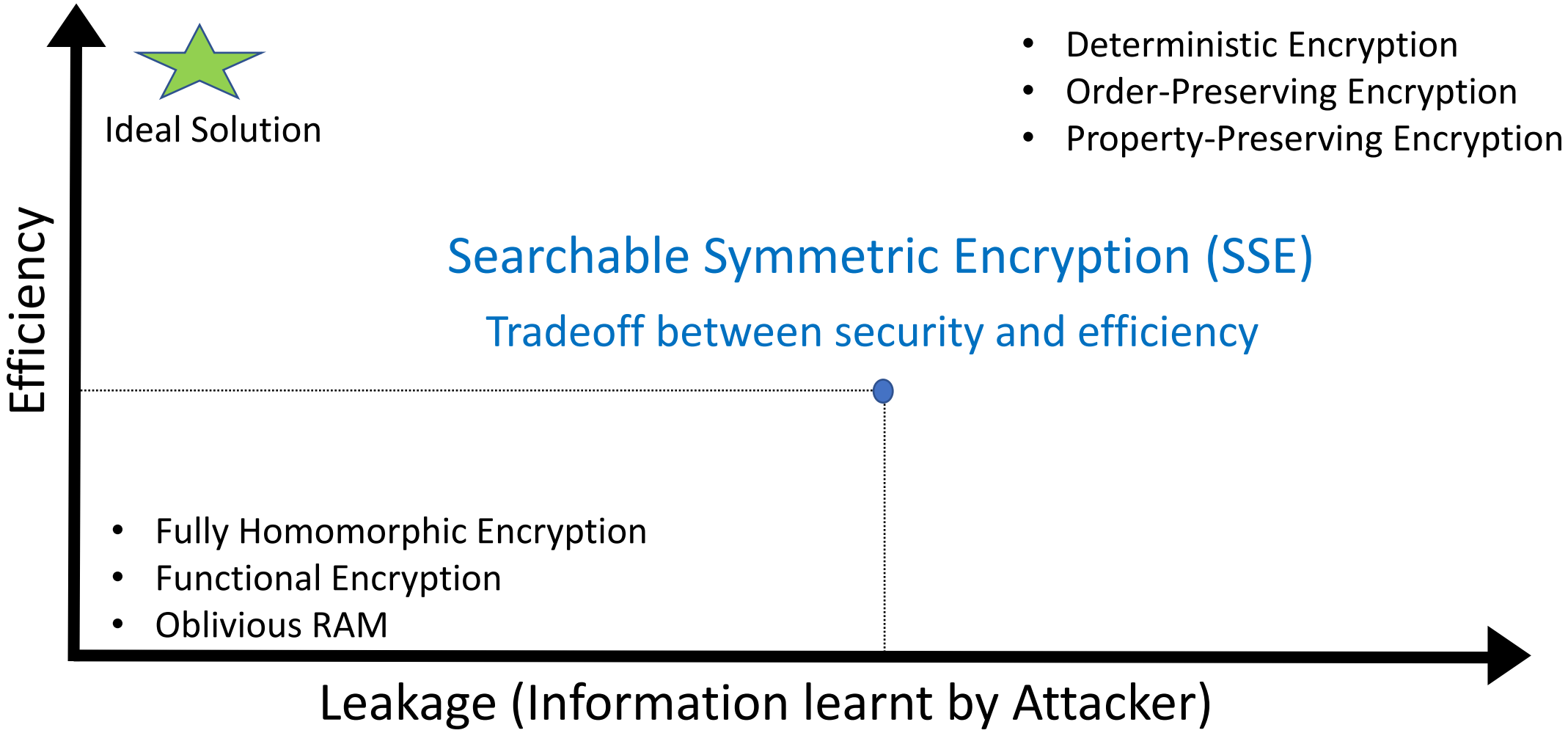


- Deterministic Encryption
- Order-Preserving Encryption
- Property-Preserving Encryption

CryptDB SAP skyhigh CipherCloud ZeroDB gitzero

Leakage-abuse attacks [NKW15, CGPR15, GMN+16,GSB+17]

Efficiency v/s Leakage: Computing on Encrypted Data



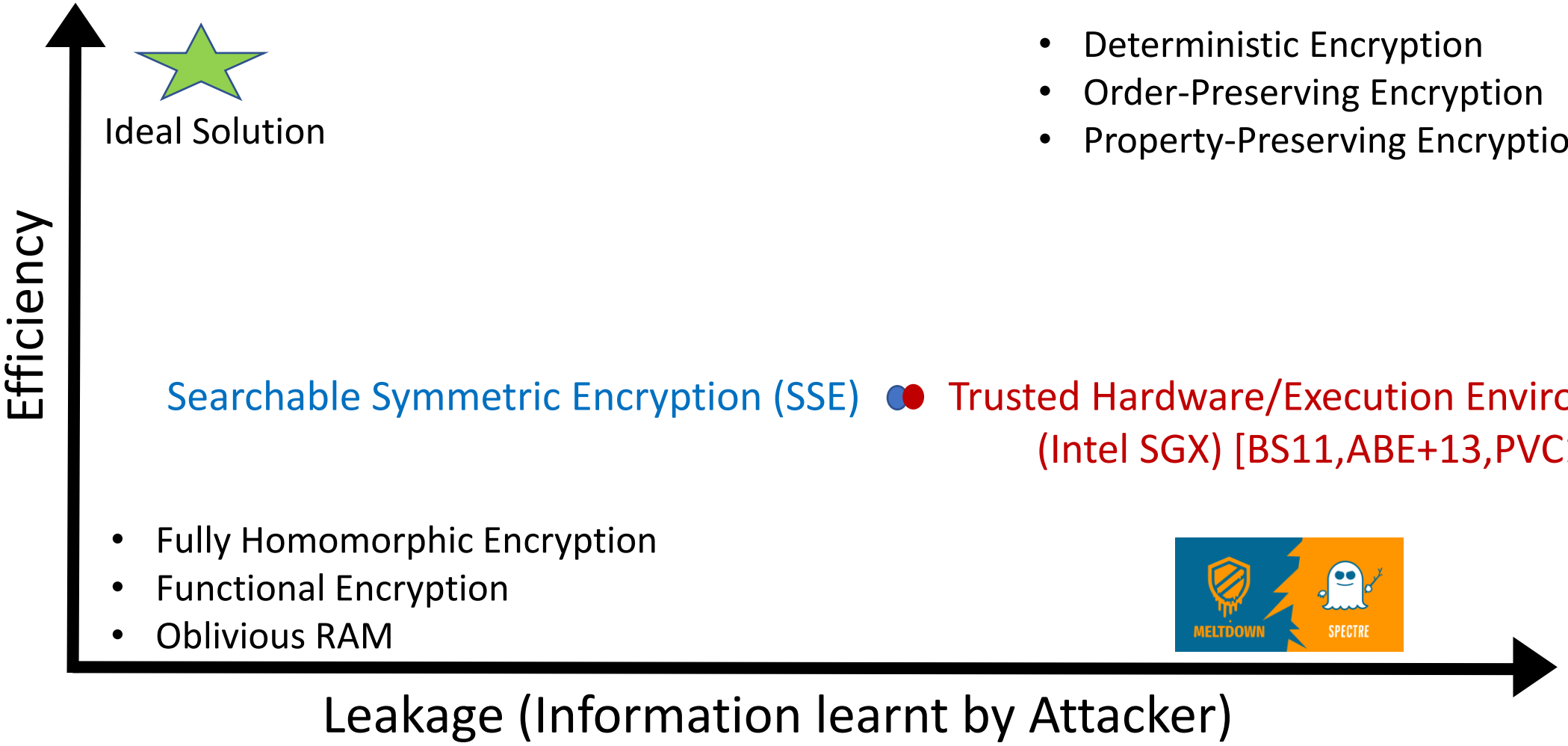
- Deterministic Encryption
- Order-Preserving Encryption
- Property-Preserving Encryption

Searchable Symmetric Encryption (SSE)

Tradeoff between security and efficiency

- Fully Homomorphic Encryption
- Functional Encryption
- Oblivious RAM

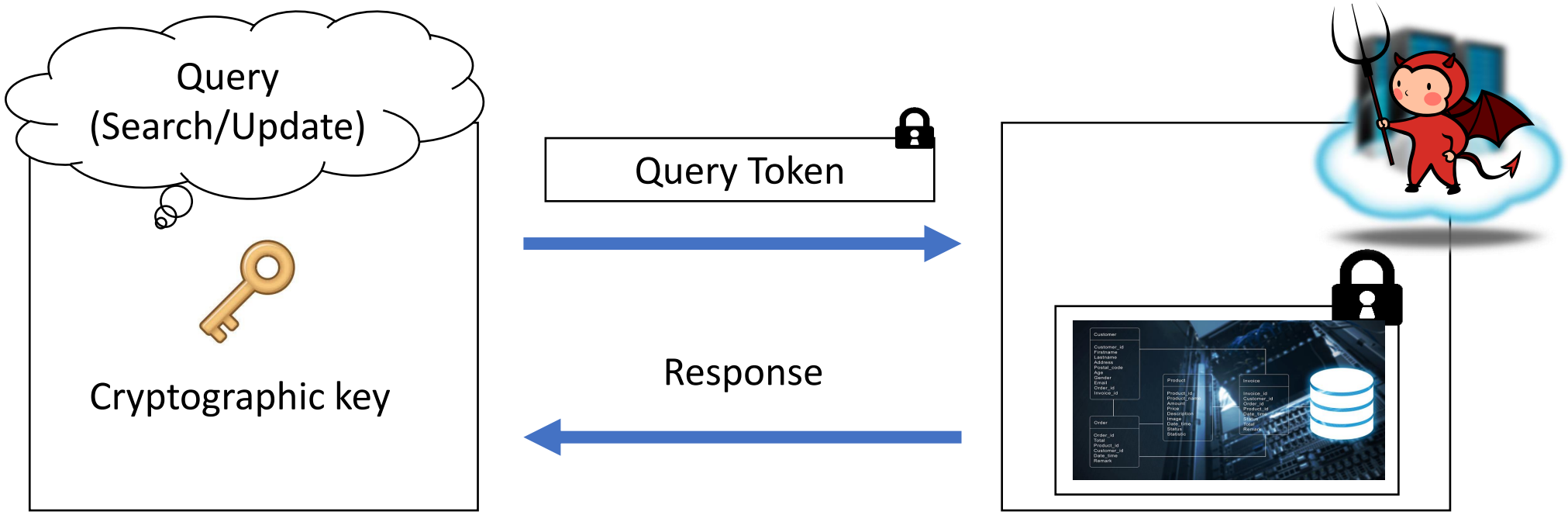
Efficiency v/s Leakage



- Deterministic Encryption
- Order-Preserving Encryption
- Property-Preserving Encryption

- Fully Homomorphic Encryption
- Functional Encryption
- Oblivious RAM

Searchable Symmetric Encryption (SSE)



Client

(Single-reader, Single-writer)

Server

(Honest but curious)

Our Contribution

Join Cross Tags (JXT)

A **practically efficient** SSE scheme for computing queries over
Equi Joins of pairs of **encrypted tables** in an **encrypted relational database**

Talk Outline

- Background: Equi Joins
- Join Cross-Tags (JXT)
- Overview of JXT
- Techniques
- Open Questions

Talk Outline

- Background: Equi Joins
- Join Cross-Tags (JXT)
- Overview of JXT
- Techniques
- Open Questions

Example of Equi Join Query

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

Join Attribute

Example of Equi Join Query

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

SELECT ID-1, ID-2

FROM Transactions JOIN Merchants ON Transaction ID

WHERE Date = 01/2022 AND Merchant Name = Apple

Talk Outline

- Background: Equi Joins
- Join Cross-Tags (JXT)
- Overview of JXT
- Techniques
- Open Questions

Our Proposal

Join Cross Tags (JXT)

A practically efficient SSE scheme for **Equi Join** queries over pairs of encrypted tables

- **No pre-processing** of joins at setup (i.e., during encrypted database generation)
  - Tables can be encrypted and added to the encrypted database independently
  - Only require tables to have a common designated join attribute
- Storage overhead – T-fold for T designated join attributes (typically small T)
- Uses **purely symmetric-key** primitives (can be implemented entirely using AES)
- Fully compatible with OXT [CJJ+13] for (conjunctive) Boolean queries

Comparison of JXT with Prior Art

Scheme	Supported Query Types	Requires Pre-Computation of Joins
SPX [KM18]	Boolean queries + Join queries	Yes
[CNR21]	Join queries	Yes
JXT (This Work)	Boolean queries + Join queries	No

JXT is the only SSE scheme supporting join queries and flexible table updates

Talk Outline

- Background: Equi Joins
- Join Cross-Tags (JXT)
- Overview of JXT
- Techniques
- Open Questions

Overview of JXT

No Pre-Processing of Joins

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

- Setup does not explicitly pre-process equi join of Transactions Table and Merchants Table
- Each table is processed **individually** – need not be created at the same time
- Only need each table to be configured to support join on attribute "Transaction ID"

Overview of JXT

No Pre-Processing of Joins

- At setup, each table is processed individually
- Each table has a designated set of join attributes
  - Is configured to support joins w.r.t. these attributes
- Consider a universe of 10 attributes {A1, A2, …, A10}:
  - Table T1 is configured to support joins over attributes A1, A3, A5 and A8
  - Table T2 is configured to support joins over attributes A2, A3, A7 and A9
  - JXT naturally supports queries over join of T1 and T2 on A3
  - No dedicated pre-processing of T1 and T2 w.r.t. A3 required

Overview of JXT

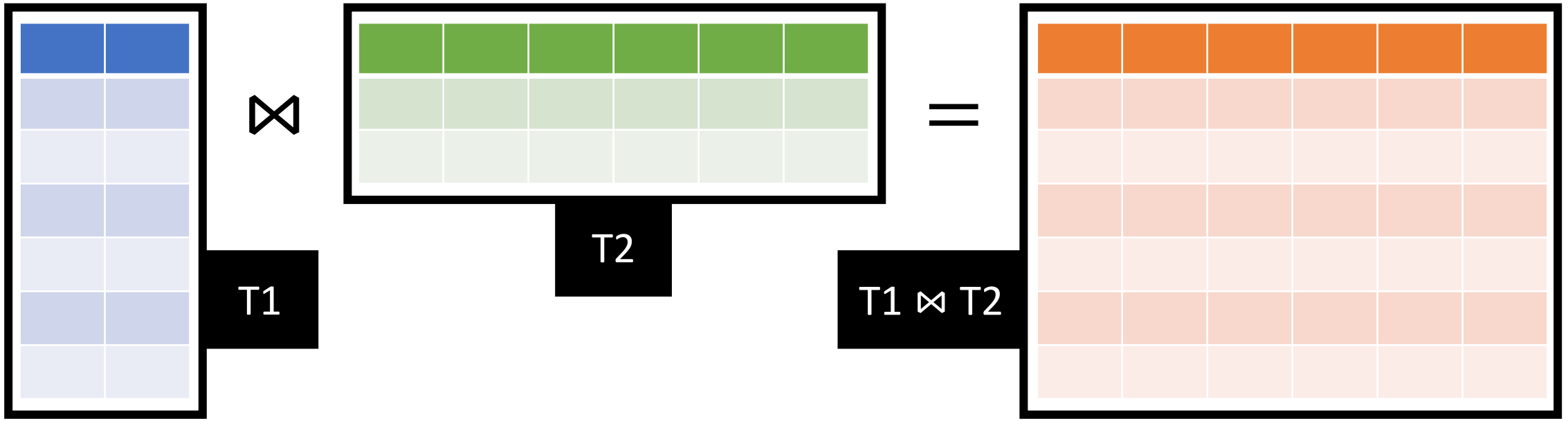
No Pre-Processing of Joins

- At setup, each table is processed individually
- Each table has a designated set of join attributes
  - Is configured to support joins w.r.t. these attributes
- In practice, number of candidate attributes for joins are small
  - Either primary/secondary key (unique value for each record)
  - Or a “high-entropy” attribute (unique values for “nearly” all records)

Overview of JXT

Comparison with Pre-Processing of Joins

- Avoids storage blow-ups due to explicit join computation:
  - Table T1 is tall and skinny (many records, few attributes)
  - Table T2 is short and wide (few records, many attributes)
  - Their equi join (for each common join attribute) maybe tall and wide



Overview of JXT

Comparison with Pre-Processing of Joins

- Avoids storage blow-ups due to explicit join computation:
  - Table T1 is tall and skinny (many records, few attributes)
  - Table T2 is short and wide (few records, many attributes)
  - Their equi join (for each common join attribute) maybe tall and wide
- Avoids re-computation of joins:
  - Suppose T1 needs to be re-setup at some point of time (updates/recovery)
  - Does not need re-computation of join with T2 (and other tables)
  - Different tables can be re-setup independently
- Avoids quadratic storage blow-up (in number of tables)

Overview of JXT

Storage and Search Overheads

- Storage Overhead:
  - Suppose a table has m rows/records and n columns/attributes
  - Suppose $\mathrm{T} \leq n$ are designated as join attributes
  - Storage: $O(mn\mathrm{T})$ i.e., $O(\mathrm{T})$-fold blowup

- Search Overhead:
  - Suppose ℓ_1 records in Transactions table matching Date = 01/2022
  - Suppose ℓ_2 records in Merchants table matching Merchant Name = Apple
  - Search cost:
    - Communication cost: $O(\ell_1 + \ell_2)$ (single round of communication)
    - Computation at client: $O(\ell_1 + \ell_2)$
    - Computation at server: $O(\ell_1 \cdot \ell_2)$

Overview of JXT

Compatibility with Oblivious Cross Tags (OXT) [CJJ+13]

- Suppose Transactions table and Merchants table had more attributes
- OXT + JXT supports more general conjunctive queries over joins of tables:

```
SELECT ID-1, ID-2

FROM Transactions JOIN Merchants ON Transaction ID

WHERE Date = 01/2022 AND Type = ACH AND Merchant Name = Apple AND City = SFO
```

Talk Outline

- Background: Equi Joins
- Join Cross-Tags (JXT)
- Overview of JXT
- Techniques
- Open Questions

Cryptoprimitives

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

- Let F be a PRF family and let k_1, k_2, k_3 and k_4 be randomly sampled keys
- Use $f(\cdot)$ to denote $F(k_1,\cdot)$ and $g(\cdot)$ to denote $F(k_2,\cdot)$
- Use $h_0(\cdot)$ to denote $F(k_3,\cdot)$ and $h_1(\cdot)$ to denote $F(k_4,\cdot)$

Processing Transactions Table

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

$$x_1 = f(\text{id} = \text{T1}||\text{Transaction ID})$$

$$y_1 = g(\text{Transaction ID} = \text{6F0X1})$$

$$z_1 = x_1 \oplus y_1$$

- Let F be a PRF family and let k_1, k_2, k_3 and k_4 be randomly sampled keys
- Use $f(\cdot)$ to denote $F(k_1,\cdot)$ and $g(\cdot)$ to denote $F(k_2,\cdot)$
- Use $h_0(\cdot)$ to denote $F(k_3,\cdot)$ and $h_1(\cdot)$ to denote $F(k_4,\cdot)$

Processing Merchants Table

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$$x'_1 = f(\text{id} = \text{M1}||\text{Transaction ID})$$

$$y'_1 = g(\text{Transaction ID} = \text{5S4D2})$$

$$z'_1 = x'_1 \oplus y'_1$$

- Let F be a PRF family and let k_1, k_2, k_3 and k_4 be randomly sampled keys
- Use $f(\cdot)$ to denote $F(k_1,\cdot)$ and $g(\cdot)$ to denote $F(k_2,\cdot)$
- Use $h_0(\cdot)$ to denote $F(k_3,\cdot)$ and $h_1(\cdot)$ to denote $F(k_4,\cdot)$

Matching Entries

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$$x_5 = f(\text{id} = \text{T5}||\text{Transaction ID})$$

$$y_5 = g(\text{Transaction ID} = \text{5S4D2})$$

$$z_5 = x_5 \oplus y_5$$

$$x'_1 = f(\text{id} = \text{M1}||\text{Transaction ID})$$

$$y'_1 = g(\text{Transaction ID} = \text{5S4D2})$$

$$z'_1 = x'_1 \oplus y'_1$$

Non-Matching Entries

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$$x_4 = f(\text{id} = \text{T4}||\text{Transaction ID})$$

$$y_4 = g(\text{Transaction ID} = \text{8P7A5})$$

$$z_4 = x_4 \oplus y_4$$

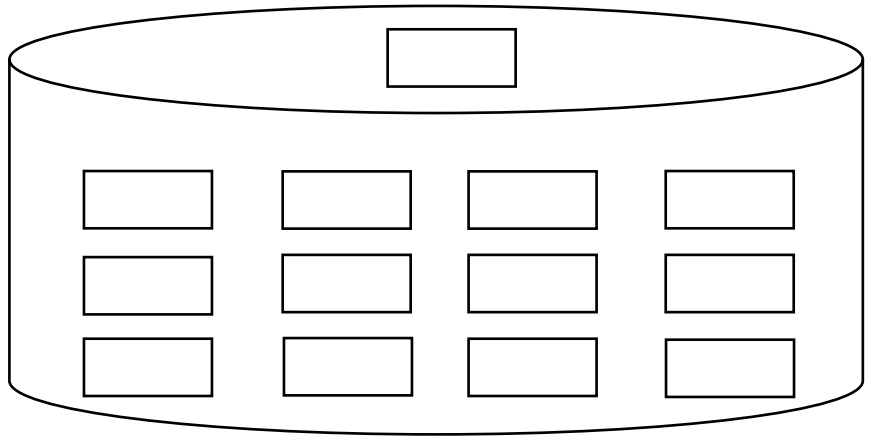
$$x'_4 = f(\text{id} = \text{M4}||\text{Transaction ID})$$

$$y'_4 = g(\text{Transaction ID} = \text{3L9M5})$$

$$z'_4 = x'_4 \oplus y'_4$$

Looking at each table in isolation

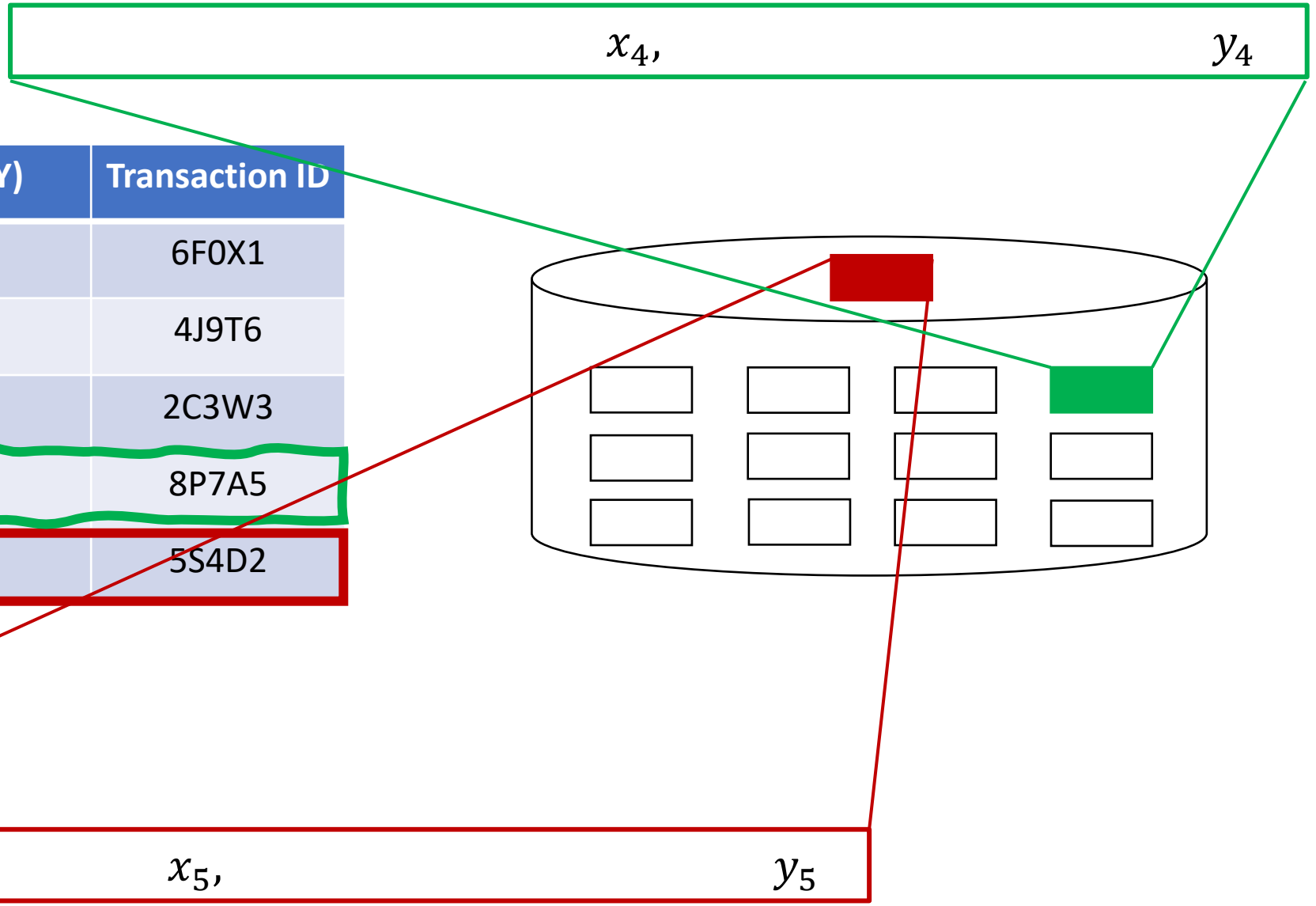
ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2



- Server stores a key-value store for each table
- Entries of the form (key, value) in random order
- Given a “valid” key, the associated value can be efficiently retrieved

Looking at each table in isolation

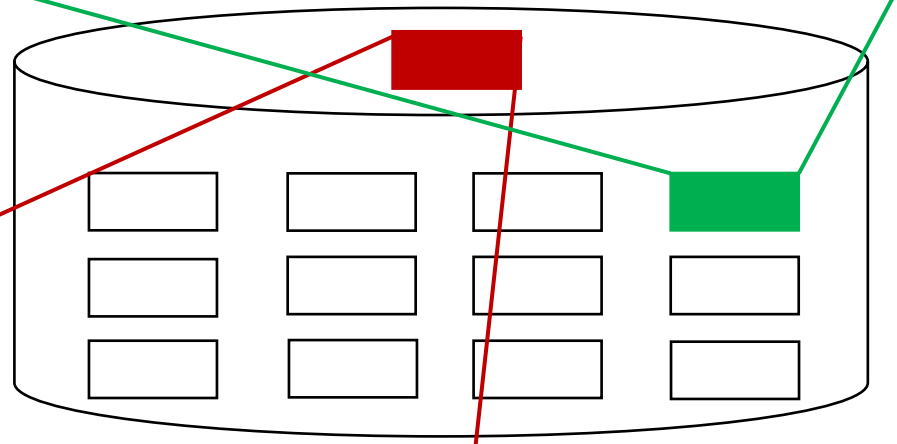
ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2



Looking at each table in isolation

$h_0(\text{Date} = 01/2022||1) \oplus x_4, h_1(\text{Date} = 01/2022||1) \oplus y_4$

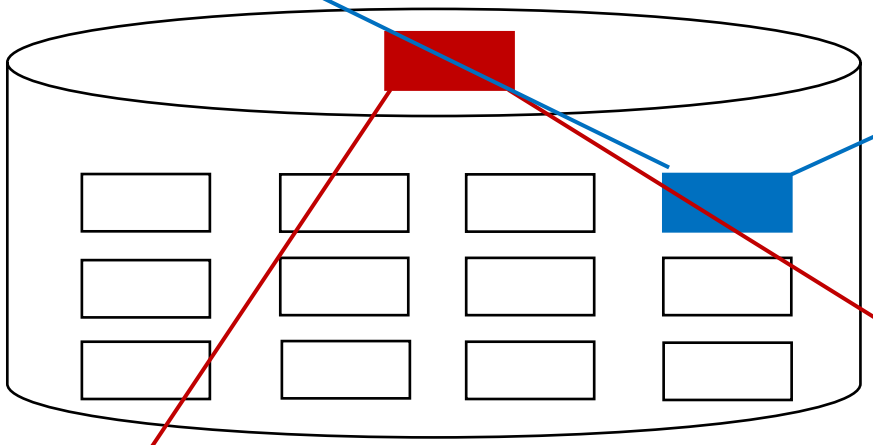
ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2



$h_0(\text{Date} = 01/2022||2) \oplus x_5, h_1(\text{Date} = 01/2022||2) \oplus y_5$

Looking at each table in isolation

$$h_0(\text{MN} = \text{Apple}||2) \oplus x'_4, h_1(\text{MN} = \text{Apple}||2) \oplus y'_4$$



ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$$h_0(\text{MN} = \text{Apple}||1) \oplus x'_1, h_1(\text{MN} = \text{Apple}||1) \oplus y'_1$$

Summary: What does the server store?

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

- Server stores masked x-values and masked y-values for each tuple of the form (record, join-attribute, attribute-value pair) in an encrypted key-value store
  - For any attribute-value pair, client should be able to generate tokens that allow the server to retrieve the relevant entries
  - Many realizations: T-Set (OXT), Encrypted Multi-Maps [KM19,PPYY19]

Summary: What does the server store?

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

- Server stores masked x-values and masked y-values for each tuple of the form (record, join-attribute, attribute-value pair) in an encrypted key-value store
- Server additionally stores the z-values directly in a Bloom Filter (or any data structure that allows efficient membership-testing)

Matching Entries: What server recovers from key-value store

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$$h_0(\text{Date} = 01/2022||2) \oplus x_5$$

$$h_1(\text{Date} = 01/2022||2) \oplus y_5$$

$$z_5 = x_5 \oplus y_5$$

$$h_0(\text{MN} = \text{Apple}||1) \oplus x'_1$$

$$h_1(\text{MN} = \text{Apple}||1) \oplus y'_1$$

$$z'_1 = x'_1 \oplus y'_1$$

Matching Entries: What server recovers from key-value store

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$$h_0(\text{Date} = 01/2022||2) \oplus x_5$$

$$h_1(\text{Date} = 01/2022||2) \oplus y'_1$$

$$z_5 = x_5 \oplus y_5$$

$$h_0(\text{MN} = \text{Apple}||1) \oplus x'_1$$

$$h_1(\text{MN} = \text{Apple}||1) \oplus y'_1$$

$$z'_1 = x'_1 \oplus y'_1$$

Matching Entries: What server checks in Bloom Filter

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$h_0(\text{Date} = 01/2022||2) \oplus x_5$

$h_1(\text{Date} = 01/2022||2) \oplus y'_1$

$z_5 = x_5 \oplus y_5$

$h_0(\text{MN} = \text{Apple}||1) \oplus x'_1$

$h_1(\text{MN} = \text{Apple}||1) \oplus y'_1$

$z'_1 = x'_1 \oplus y'_1$

Search steps: Overview

1. Client sends to server tokens to recover $h_1(\text{Date} = 01/2022||2) \oplus y'_1$ and $h_0(\text{MN} = \text{Apple}||1) \oplus x'_1$
2. Client also sends to the server $h_1(\text{Date} = 01/2022||2) \oplus h_0(\text{MN} = \text{Apple}||1)$
3. Server recovers z'_1 and checks its membership in the Bloom Filter
4. Membership test returns TRUE

$h_0(\text{Date} = 01/2022||2) \oplus x_5$

$h_1(\text{Date} = 01/2022||2) \oplus y'_1$

$z_5 = x_5 \oplus y_5$

$h_0(\text{MN} = \text{Apple}||1) \oplus x'_1$

$h_1(\text{MN} = \text{Apple}||1) \oplus y'_1$

$z'_1 = x'_1 \oplus y'_1$

Non-Matching Entries: Recap

Transactions Table

ID-1	Date (MM/YY)	Transaction ID
T1	01/2021	6F0X1
T2	05/2021	4J9T6
T3	07/2021	2C3W3
T4	01/2022	8P7A5
T5	01/2022	5S4D2

Merchants Table

ID-2	Merchant Name	Transaction ID
M1	Apple	5S4D2
M2	Exxon	9N2E8
M3	Gucci	7T8U3
M4	Apple	3L9M5
M5	Five Guys	6F0X1

$$x_4 = f(\text{id} = \text{T4}||\text{Transaction ID})$$

$$y_4 = g(\text{Transaction ID} = \text{8P7A5})$$

$$z_4 = x_4 \oplus y_4$$

$$x'_4 = f(\text{id} = \text{M4}||\text{Transaction ID})$$

$$y'_4 = g(\text{Transaction ID} = \text{3L9M5})$$

$$z'_4 = x'_4 \oplus y'_4$$

Search steps: Overview

1. Client sends to server tokens to recover $h_1(\text{Date} = 01/2022||1) \oplus y_5$ and $h_0(\text{MN} = \text{Apple}||2) \oplus x'_4$
2. Client also sends to the server $h_1(\text{Date} = 01/2022||1) \oplus h_0(\text{MN} = \text{Apple}||2)$
3. Server recovers $(x'_4 \oplus y_5)$ and checks its membership in the Bloom Filter
4. Membership test returns FALSE

$h_0(\text{Date} = 01/2022||1) \oplus x_4$

$h_1(\text{Date} = 01/2022||1) \oplus y_5$

$z_4 = x_4 \oplus y_4$

$h_0(\text{MN} = \text{Apple}||2) \oplus x'_4$

$h_1(\text{MN} = \text{Apple}||2) \oplus y'_4$

$z'_4 = x'_4 \oplus y'_4$

Search steps: Overview

1. Client sends to server tokens to recover $h_1(\text{Date} = 01/2022||1) \oplus y_5$ and $h_0(\text{MN} = \text{Apple}||2) \oplus x'_4$
2. Client also sends to the server $h_1(\text{Date} = 01/2022||1) \oplus h_0(\text{MN} = \text{Apple}||2)$
3. Server recovers $(x'_4 \oplus y_5)$ and checks its membership in the Bloom Filter
4. Membership test returns FALSE

- Pairwise XOR of masks incurs quadratic overhead
- Additional techniques to delegate pairwise XOR computations to the server while client does linear amount of work
  - Details in the paper

Leakage of JXT: Join Attribute Distribution Pattern

- Leaks frequency distribution of values taken by Transaction ID across records matching Date = 01/2022 in Transactions Table
- Unique value for each record if join attribute is primary/secondary key
  - No non-trivial leakage in this case
- Leakage is not significant if join attribute is high-entropy
  - Unique value for nearly each record, so each value occurs only once with high probability
- Impact of leakage less clear if join attribute is low-entropy
  - Strategy: Structure query such that join-attribute is high-entropy in first table

Talk Outline

- Background: Equi Joins
- Join Cross-Tags (JXT)
- Overview of JXT
- Techniques
- Open Questions

Open Questions and Future Directions

- Extend JXT to flexibly support joins over **arbitrary** attributes
- Extend JXT to support queries over equi joins of **three (or more) tables**
- Analyze impact of leakage for **low-entropy** join attributes
- Extend JXT to support **dynamic** addition/deletion of records
- **Implementation and experimentation** over large databases

Full version: https://eprint.iacr.org/2021/1471

Thank You! Questions?