Database Query Authentication

Ahmed Kosba

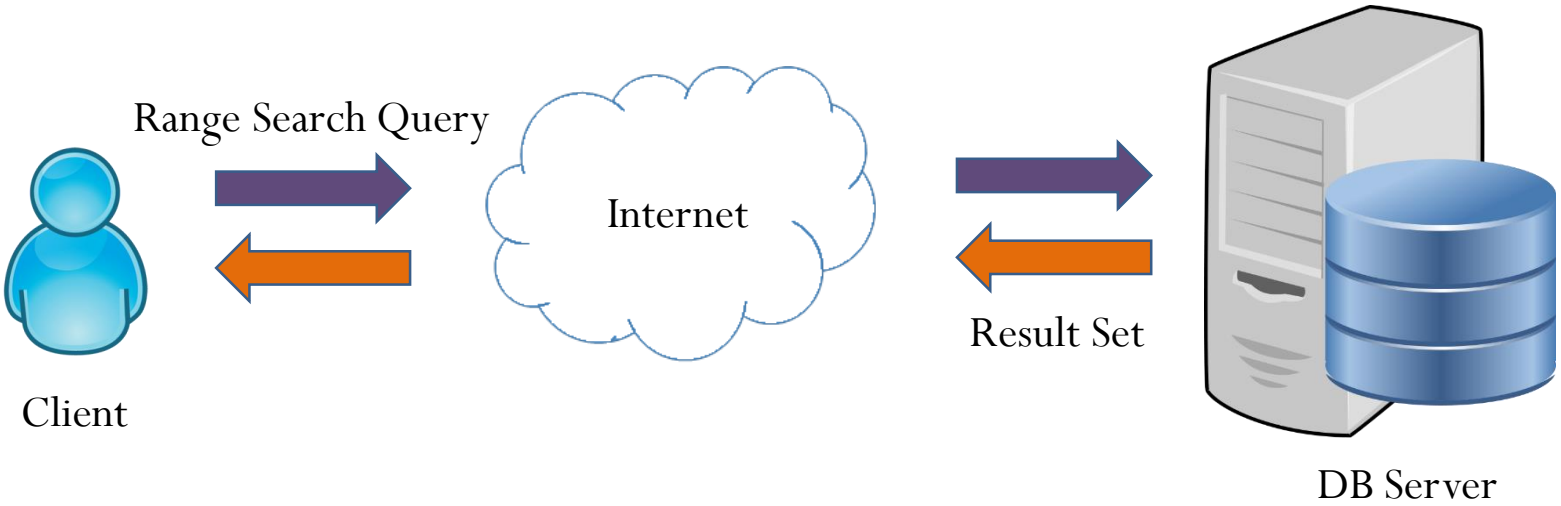
Based on:

CorrectDB: SQL Engine with Practical Query Authentication

Sumeet Bajaj, Radu Sion

VLDB, 2013

Problem Overview



What if the server is not trusted?

Query Authentication (QA): Requirements

- Correctness
  - Authentic elements (i.e. belong to the original data)
  - Satisfying predicates
- Completeness

Outline

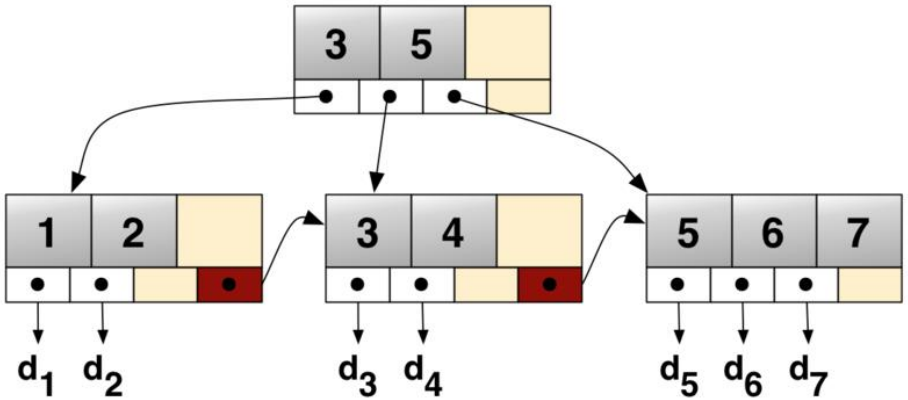
- Existing QA Solutions
  - Tree-based Solutions
  - Signature-based Solutions
- Motivation of using Secure-hardware
- CorrectDB Architecture
- Experimental Evaluation

Existing QA Solutions

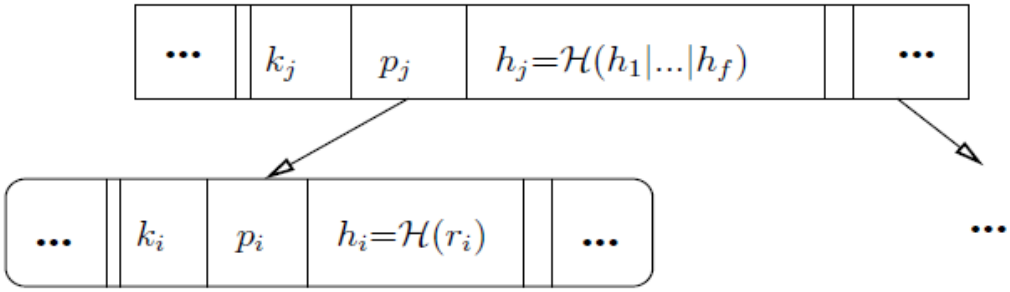
- Generally, QA approaches use the following:
  - Authentication Data Structure (ADS)
  - Verification Object (VO)
- Techniques of query authentication can be categorized into:
  - Tree-based solutions.
  - Signature-based solutions
  - Secure hardware-based.

Tree-based Solutions

- The Merkle hash tree is the basis for many solutions
- Variants:
  - B+ Tree based: MB-Tree – Verifiable B-Tree (VBT)



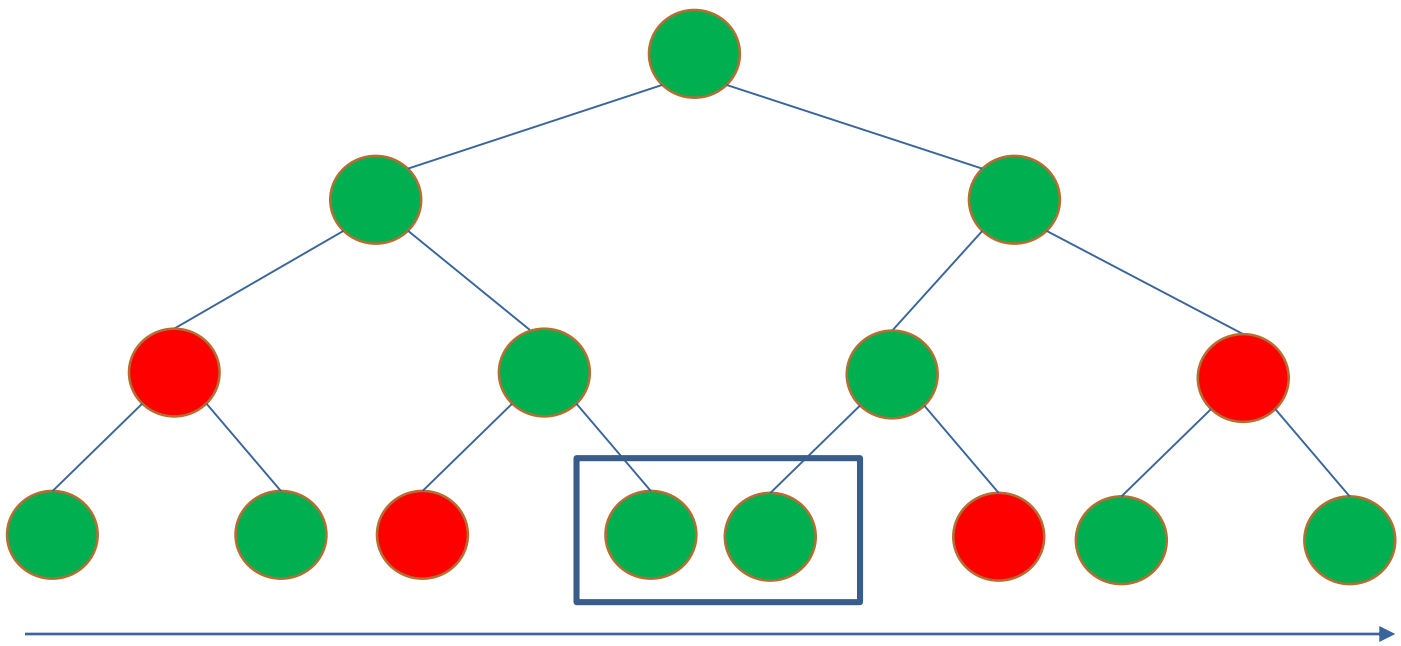
B+ Tree Image from WIKIPEDIA



MB Tree, Li et al (SIGMOD'06)

Tree-based Solutions

- Authentication of Typical DB Queries:
  - Range Queries



  - Question: How to authenticate an empty result set?

Tree-based Solutions

- Authentication of Typical DB Queries:
  - Join Queries:
    - Naïve Solution: Materialize the cross product, and build an ADS on it.

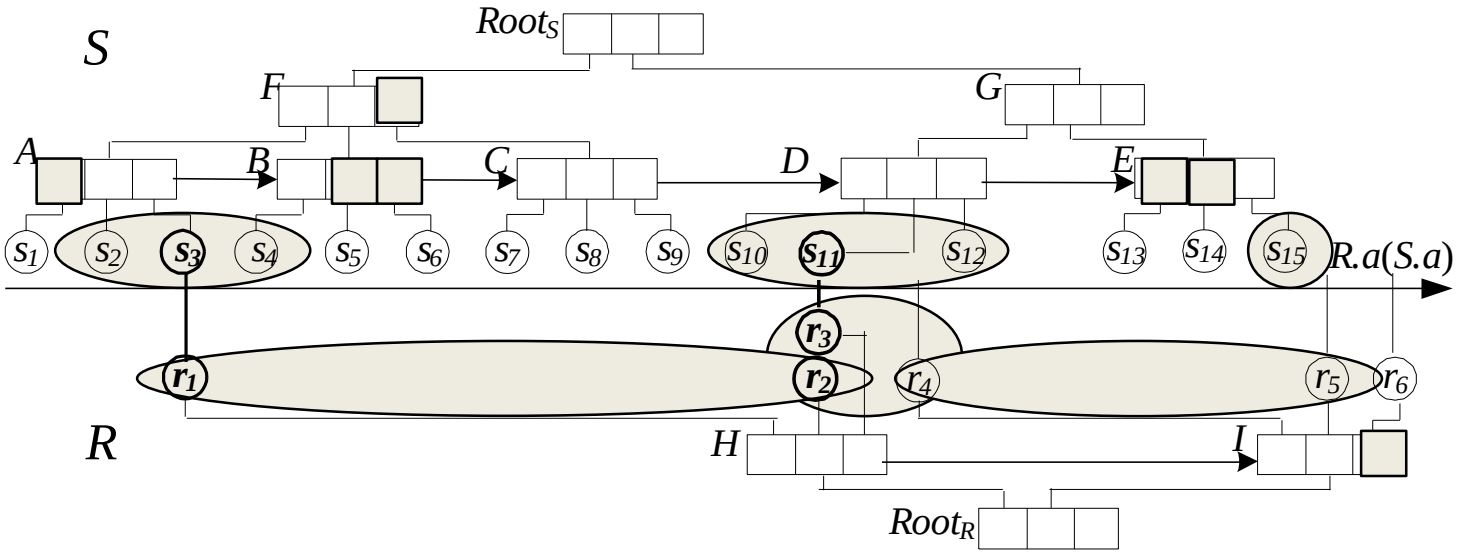
      Very Expensive
    - Solution 2: Consider a join query between R_1 and R_2 such that $R_1.A = R_2.B$,
      - Setup: Build ADS for both relations on the attributes A and B.
      - In the query time:
        - send the smallest relation to the client.
        - Per each element of the retrieved relation, perform a look up range query in the bigger relation.

Tree-based Solutions

- Authentication of Typical DB Queries:
  - Join Queries: (Yang et al, SIGMOD 2009) presented methods for:
    - Binary Join Query Authentication: AISM, AIM, ASM

      These techniques mainly **differ** in the requirement of having an ADS available for one or two relations.
    - Complex Join Query Authentication: e.g. Multi-way joins.

Example of AIM (Yang et al, SIGMOD'09) – From Authors' Slides



VO: root signature of T_S, root signature of T_R, r_1

1. h_{s_1}, s_2, s_3, s_4;
2. r_2;
3. h_{s_5}, h_{s_6}, h_C, s_{10}, s_{11}, s_{12};
4. r_3, r_4;
5. r_5;
6. $h_{s_{13}}$, $h_{s_{14}}$, s_{15};
7. h_{r_6};

Signature-based Approaches

- Recall the MAC approach discussed in class before:
  - Merits - Drawbacks
  - How to modify it to be able to handle:
    - Range queries?
    - Join queries?

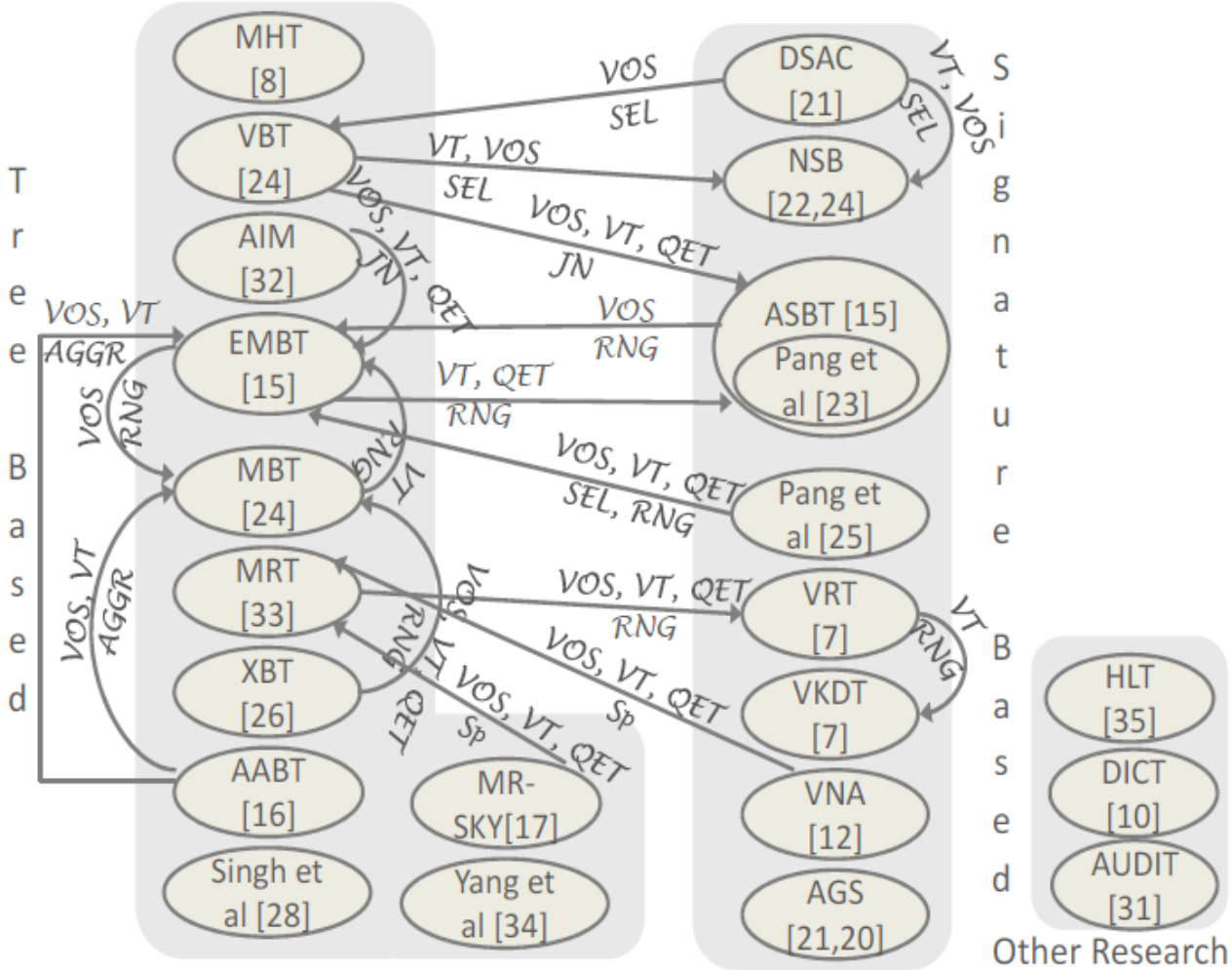
Challenges with Update Operations

- Providing update capabilities while ensuring QA can add much complexity.
- A subset of the proposed techniques do not address this problem at all, or assume a static or an infrequently-updated database.
- Challenges:
  - Concurrent multi-client write operations.
  - Replay Attacks.
  - Side Effects.

Evaluation Metrics

- Verification Object Size (VOS)
- Query Execution Time (QET)
- Verification Time (VT)

Comparative Summary



Motivation for Secure Hardware

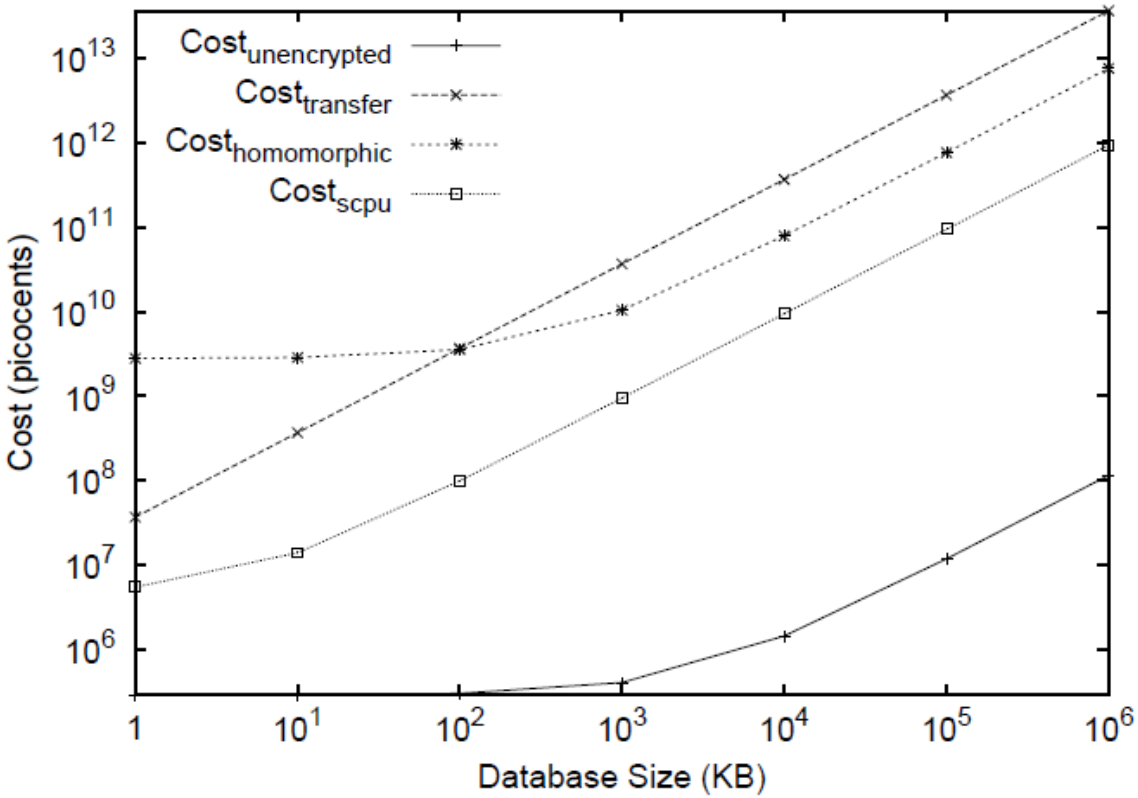
- Drawbacks of previous QA approaches:
  - The VO can be large in the case of tree-based solutions.
  - Indexing solution is very inefficient for signature-based approaches.
  - Solutions are usually customized to certain types of queries (typically simple).
  - Limitations of Update Operations.

Motivation for Secure Hardware

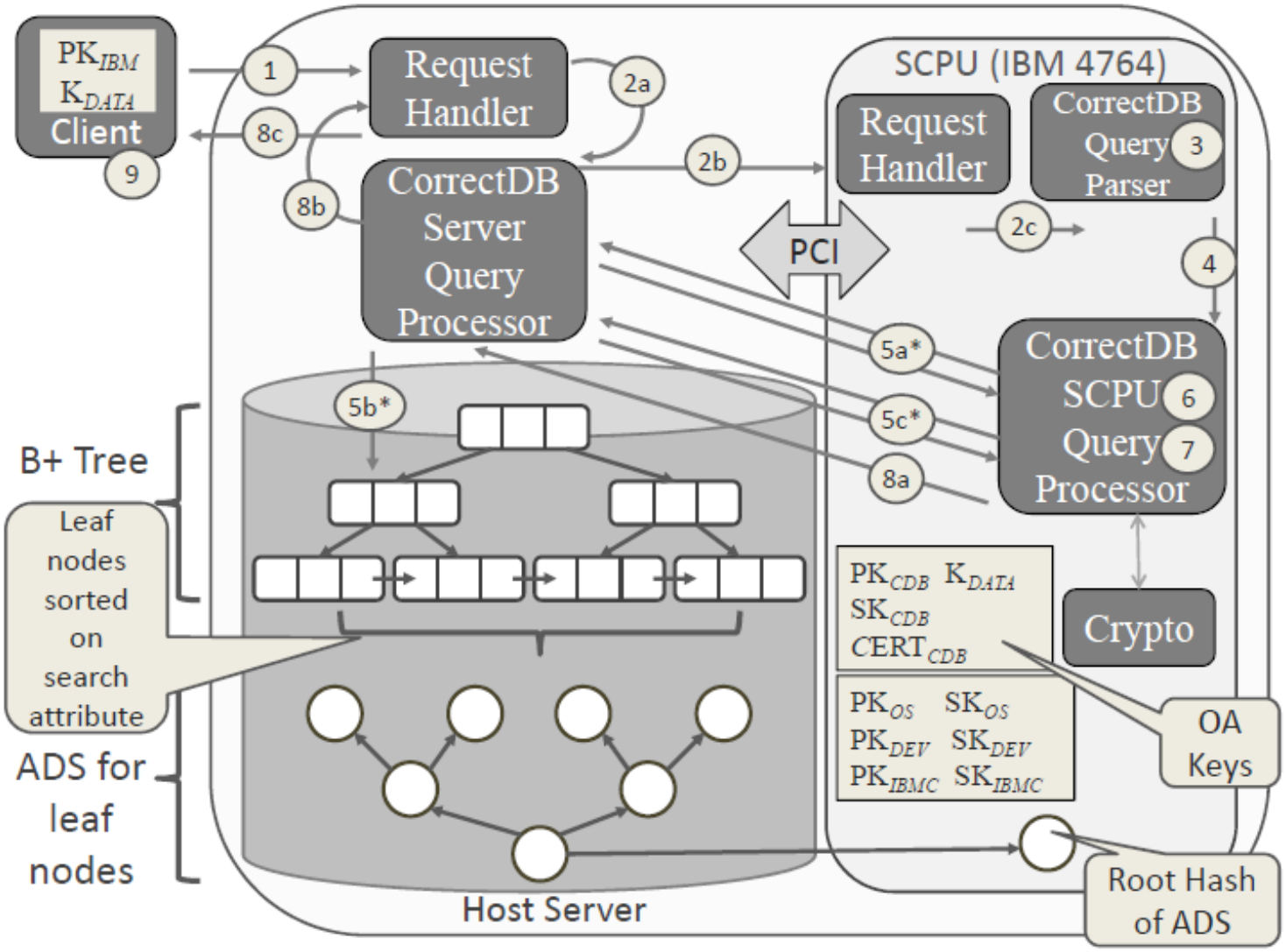
- Usage of secure hardware provides:
  - Data Proximity (eliminating costs for transferring VOs).
  - Query Expressiveness (The SCPUs will be able to execute arbitrary queries).
  - Flexibility for Data Update Operations.
  - Querying attributes without ADS
  - Maintaining Privacy: Remote Processing of Encrypted Data
- Drawback: High Acquisition Cost – Upgrade Cost

Motivation for Secure Hardware
Real costs of Security

- In an earlier study [TrustedDB, SIGMOD 2011], the authors compared cryptographic approaches to secure hardware solution under aggregation queries.



CorrectDB Architecture



Query Parsing and Execution

- Objective: Rewrite the original client query into sub-queries, achieving the following objectives:
  - Processing within the SCPU is **minimized**.
  - Any intermediate results generated by server-side query processing can be validated by the SCPU .
  - Any operations that cannot be authenticated are processed on the intermediate results inside the SCPU.
  - The result of the sub-queries is the same as if the original client query was executed without any re-writes.

Query Parsing and Execution

- Example query from TPC-H:

```
SELECT sum(l_extendedprice*l_discount), o_priority
FROM   lineitem, orders
WHERE  l_shipdate >= '1993-01-01'
AND    l_shipdate < '1994-01-01'
AND    o_orderdate between '1992-01-01' AND '1993-01-01'
AND    l_discount between 0.05 AND 0.07
AND    l_orderkey = o_orderkey
AND    o_priority in ('W', 'R', 'Q')
```

- Assuming having an ADS on l_shipdata and o_orderdate

Query Parsing and Execution

- The query will then be rewritten, such that the untrusted server-side searches for tuples achieving:

```
    l_shipdate >= '1993-01-01'
AND l_shipdate < '1994-01-01'
AND o_orderdate between '1992-01-01' AND '1993-01-01'
```

, while the SCPU side will check for the following predicates

```
SELECT sum(l_extendedprice*l_discount), o_priority
FROM   lineitem, orders
AND    l_discount between 0.05 AND 0.07
AND    l_orderkey = o_orderkey
AND    o_priority in ('W', 'R', 'Q')
```

Queries

- The authors present arguments for the correctness and the completeness of the typical type of DB queries:
  - Range queries.
  - Projections: done directly by the SCPU without the need to an ADS.
  - Aggregations, Grouping, ..etc: Can be handled by the SCPU without passing all the data to the client. (Compare to previous approaches)
  - Join operations: Applies sort-merge and nested-loop joins.

Client Side Verification

- Assume the result R contains the tuples t_1, t_2, .. t_r , the digest calculated by the SCPU will be:
  - $D(R) = H(C_{id} || Q_c || Nonce || H(t_1) || .. || H(t_r))$
- The SCPU signs the digest using its secret key, and returns the signed result:
  - $S(D(R), SK)$
- The client can then verify the result through the public key of the SCPU.

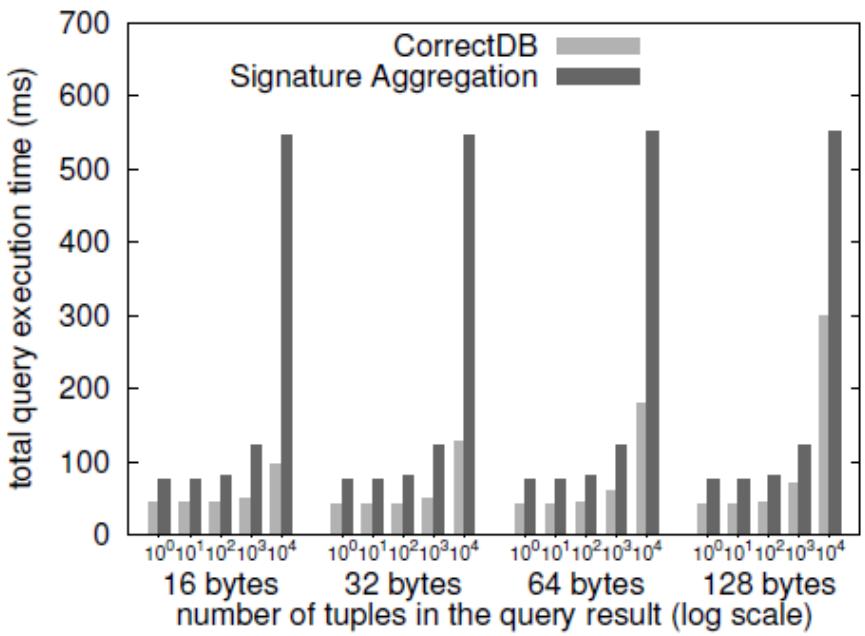
Experimental Evaluation

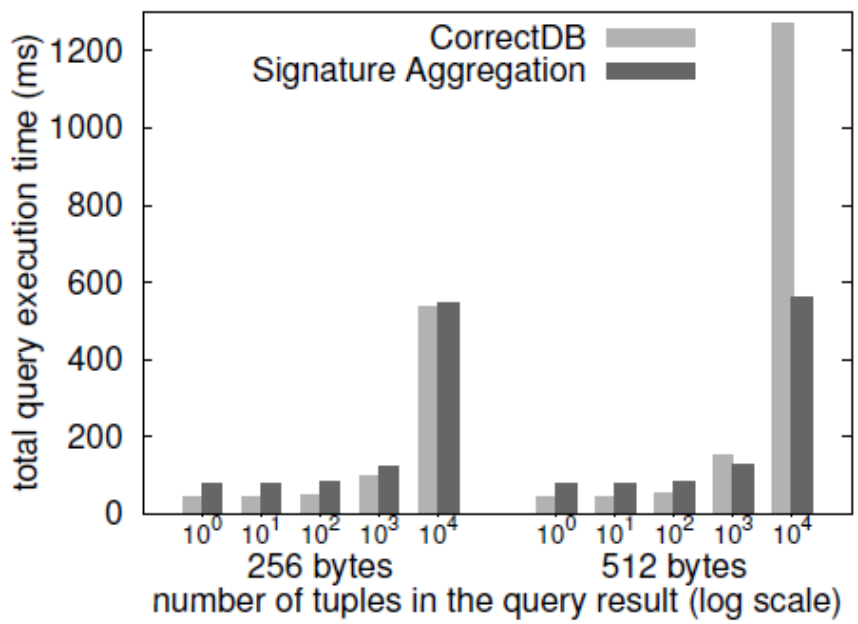
- Comparisons were done with respect to most efficient range query mechanisms [Signature-based], and with the most efficient join query mechanism [Tree-based].
- Metrics as before:
  - Query Execution Time: QET + VT.
  - Verification Object Size (VOS)
- Basic representative queries were selected for evaluation, as other approaches only support basic operations.

Experimental Evaluation

- Range Queries:

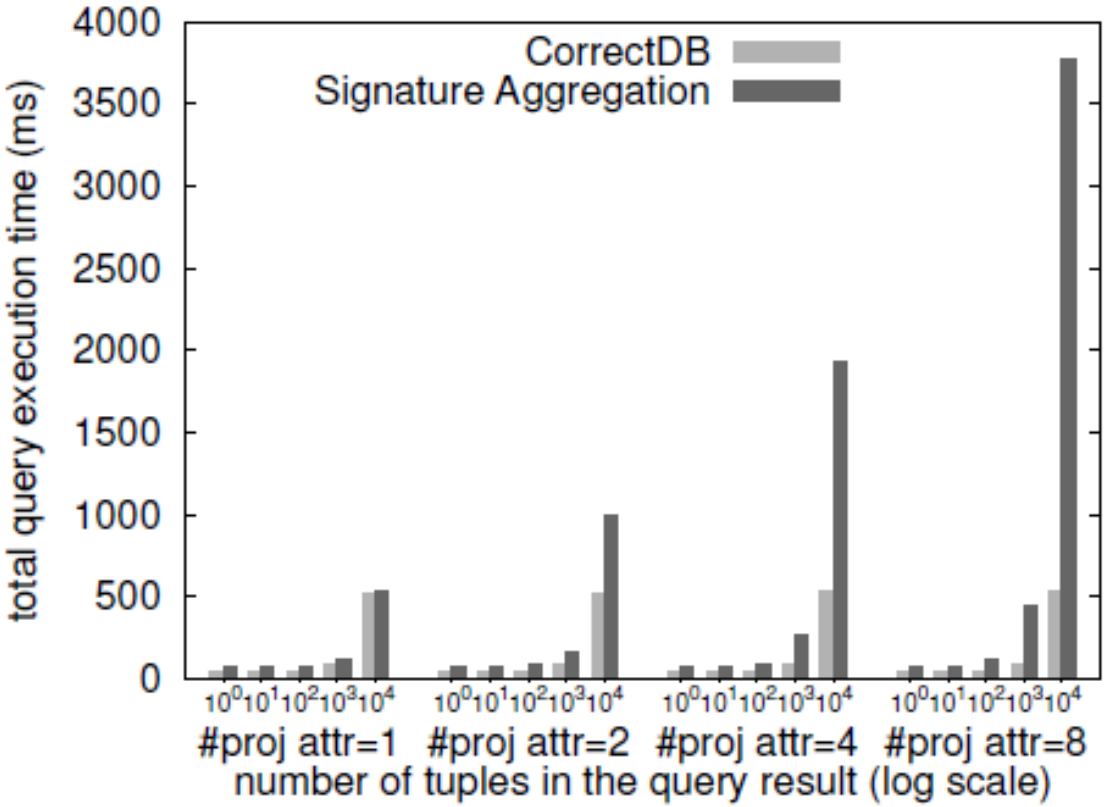
```
SELECT * FROM R where R.key > 'LB' AND R.key < 'UB'
```





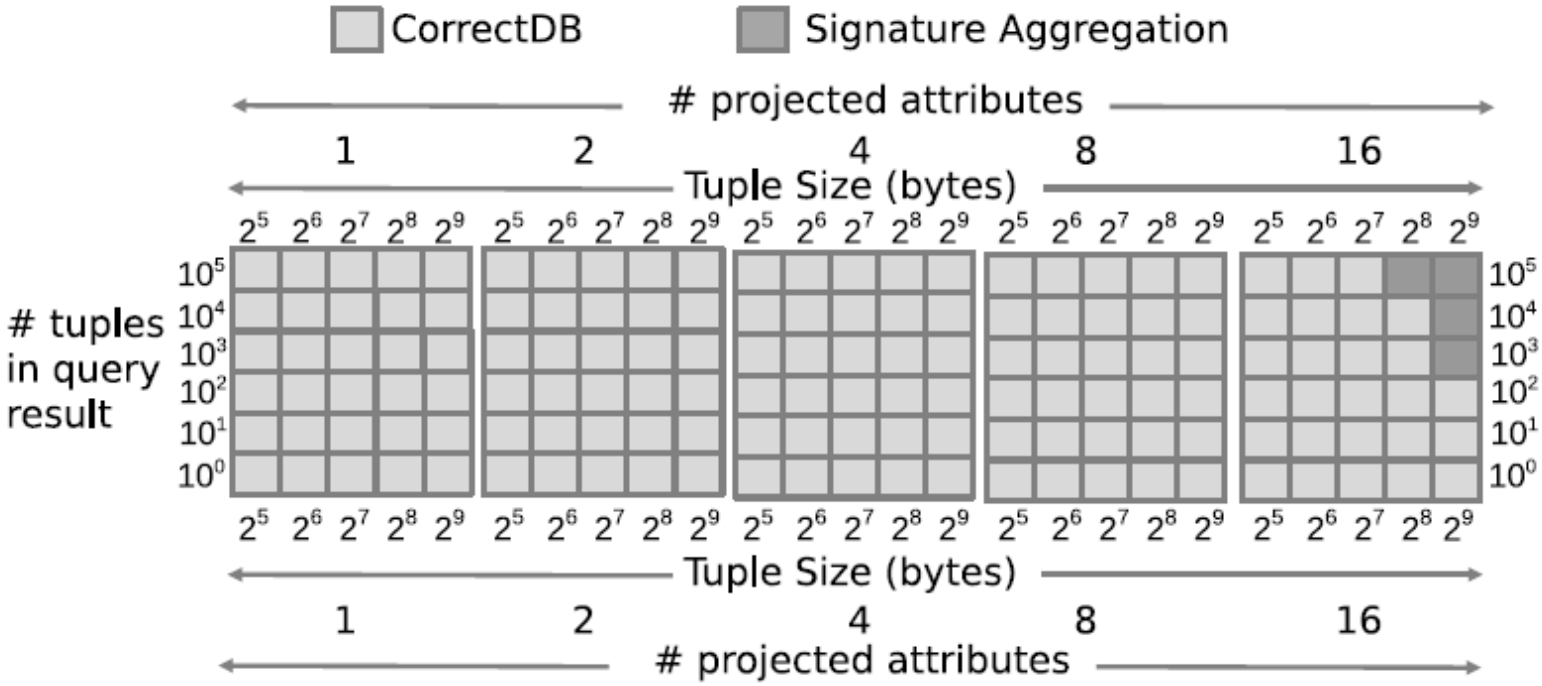
Experimental Evaluation

- Range queries while varying the number of projected attributes



Experimental Evaluation

- Comparison under Range Queries

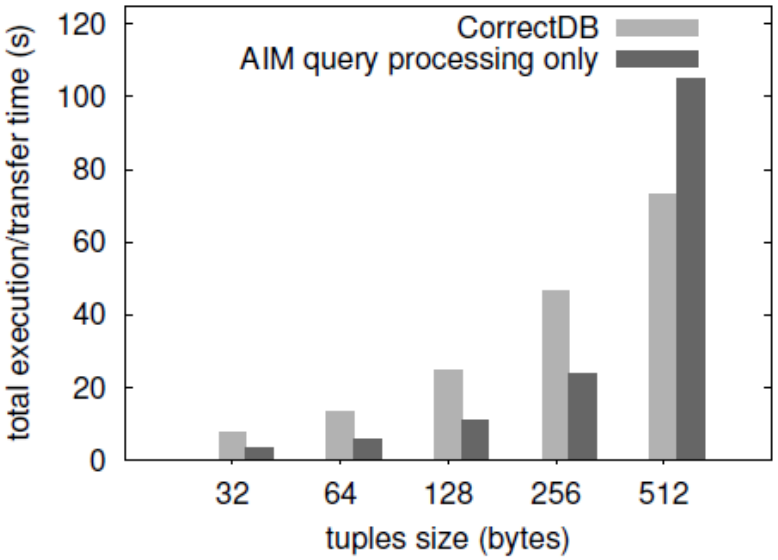


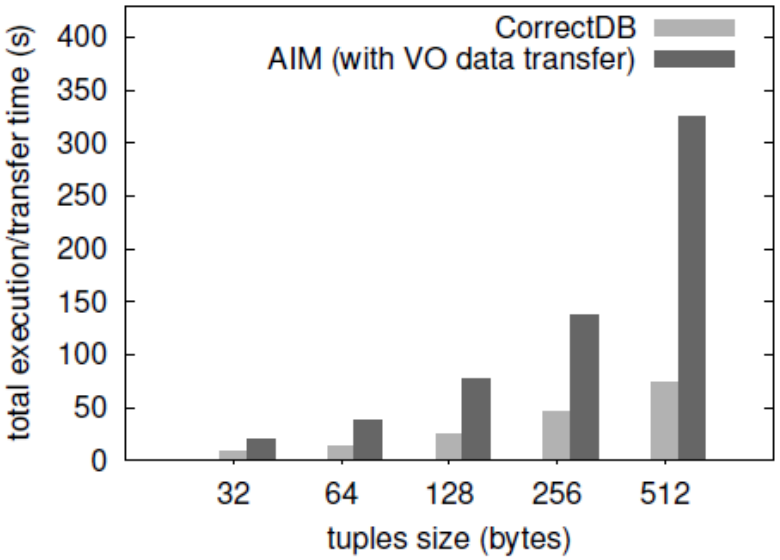
Experimental Evaluation

- Join Query:

```
SELECT * FROM R, S WHERE R.key = S.key AND R.key > 'LB'
AND R.key < 'UB'
```

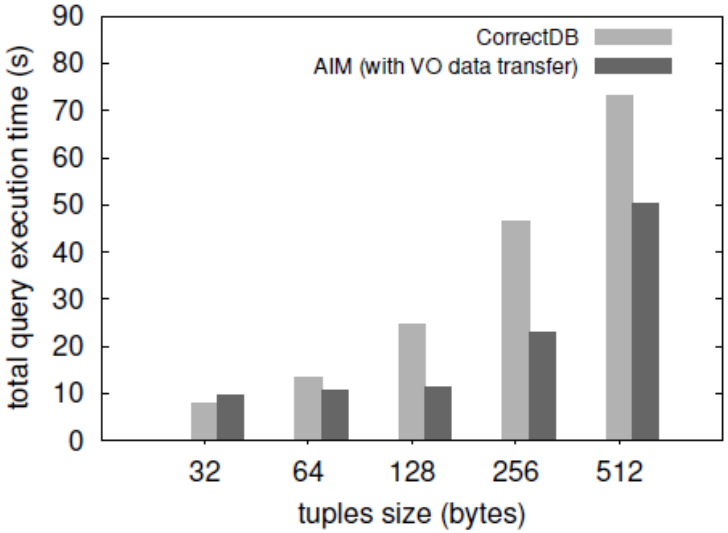
  - Foreign Key Join:

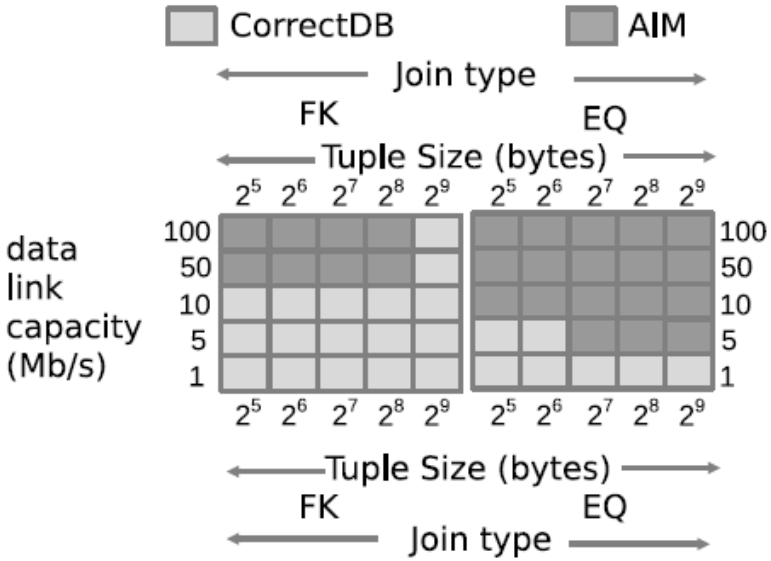




Experimental Evaluation

- Join Query (Equi-Join):

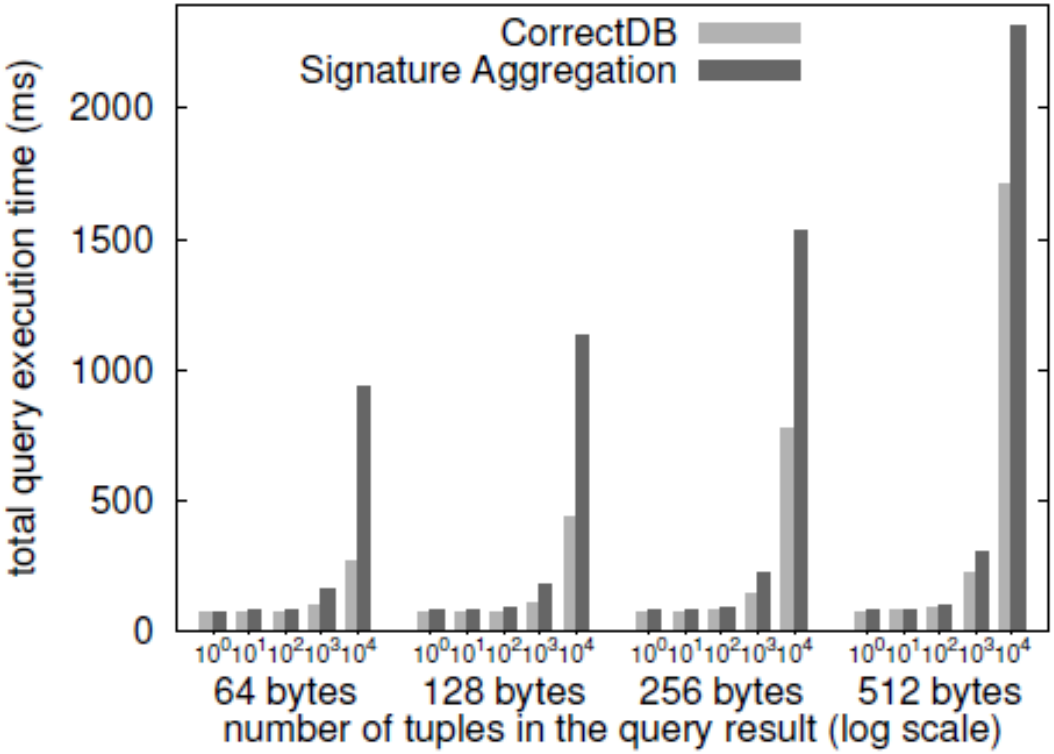




Experimental Evaluation

- Update Operation:

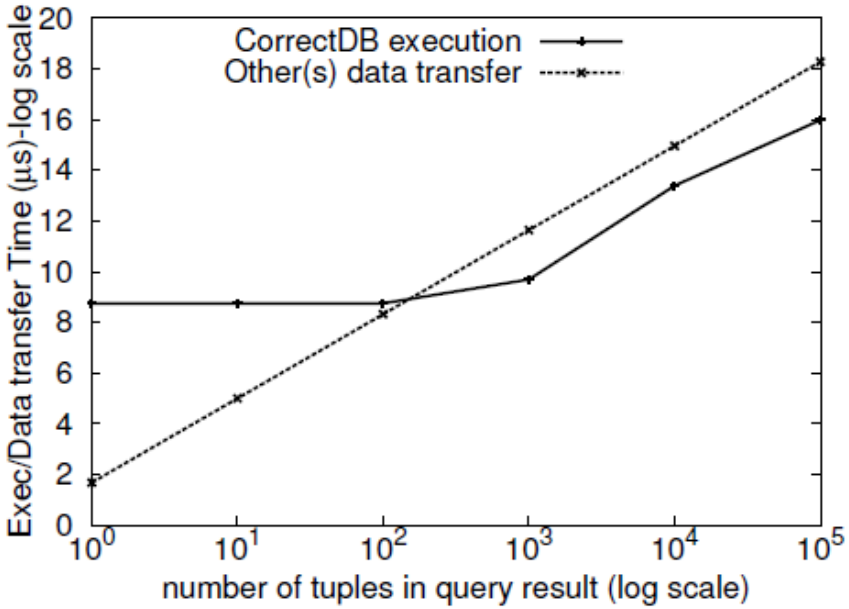
```
UPDATE key=key+1 FROM R where R.key>'LB' AND R.key<'UB'
```

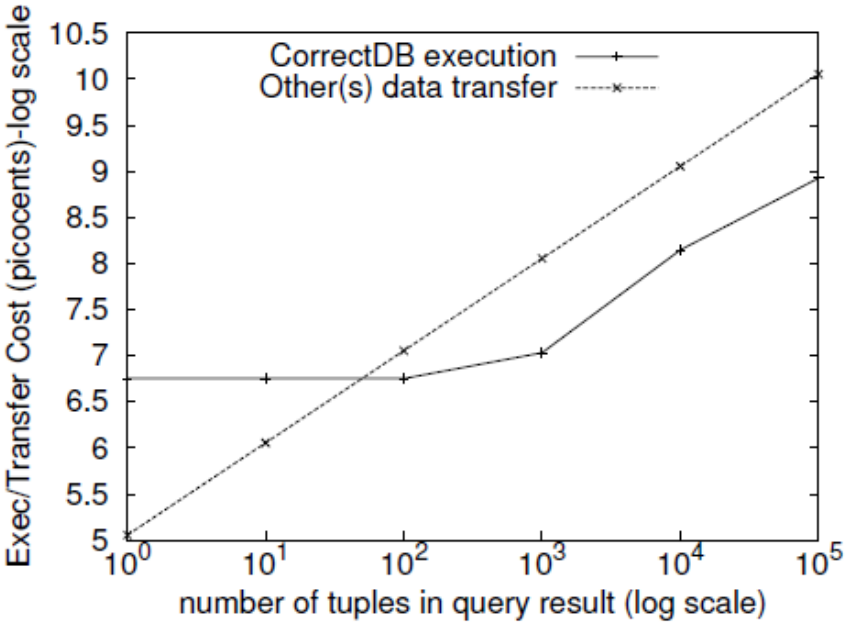


Experimental Evaluation

- Aggregate Queries:

```
SELECT SUM(key) FROM R where R.key > 'LB' AND R.key < 'UB'
```





Thank You ☺