T. J. Watson Research Center

Integrity Auditing of Outsourced Data

Min Xie*, Haixun Wang, Jian Yin**, Xiaofeng Meng***

* Renmin University of China
** IBM T.J. Watson Research Center

Outline

- Concerns in Database Outsourcing
- Our Motivation
- A Probabilistic Approach
- Proof of Security
- Experimental Results
- Future Work

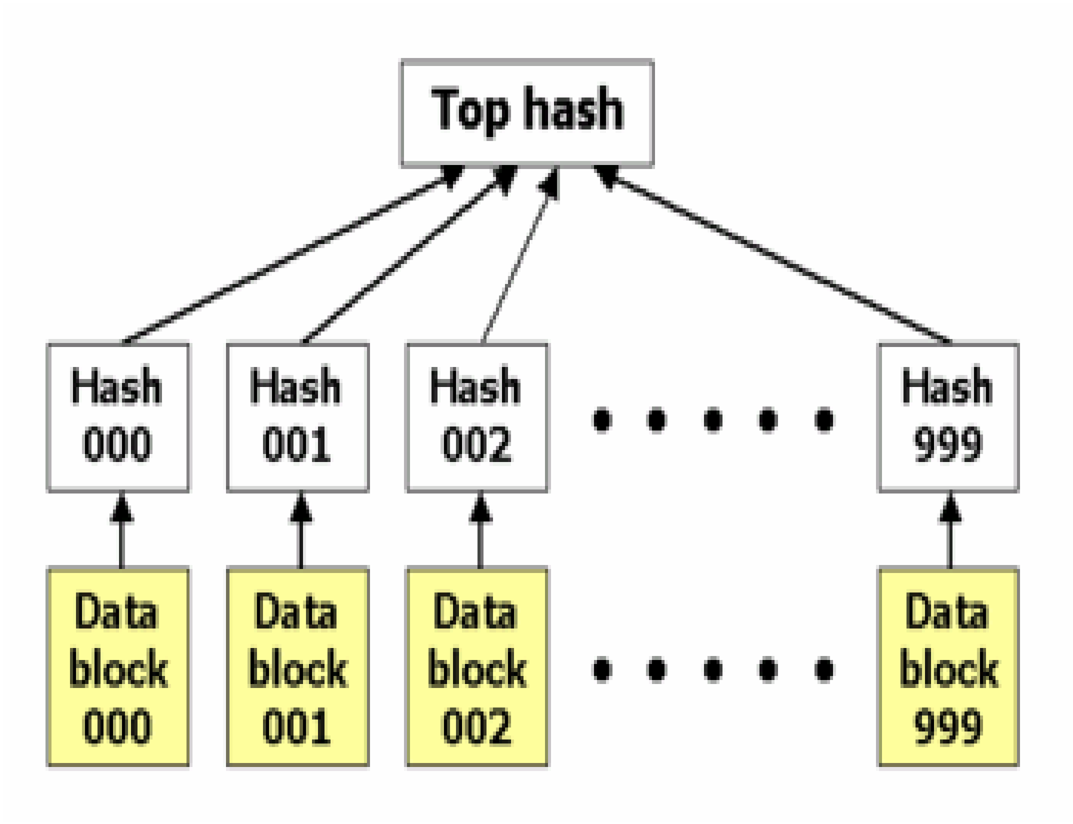
Database outsourcing

- Push:
  - Lowered network cost
- Pull:
  - Expanding market
- Obstacles:
  - Security

Two aspects of security concerns

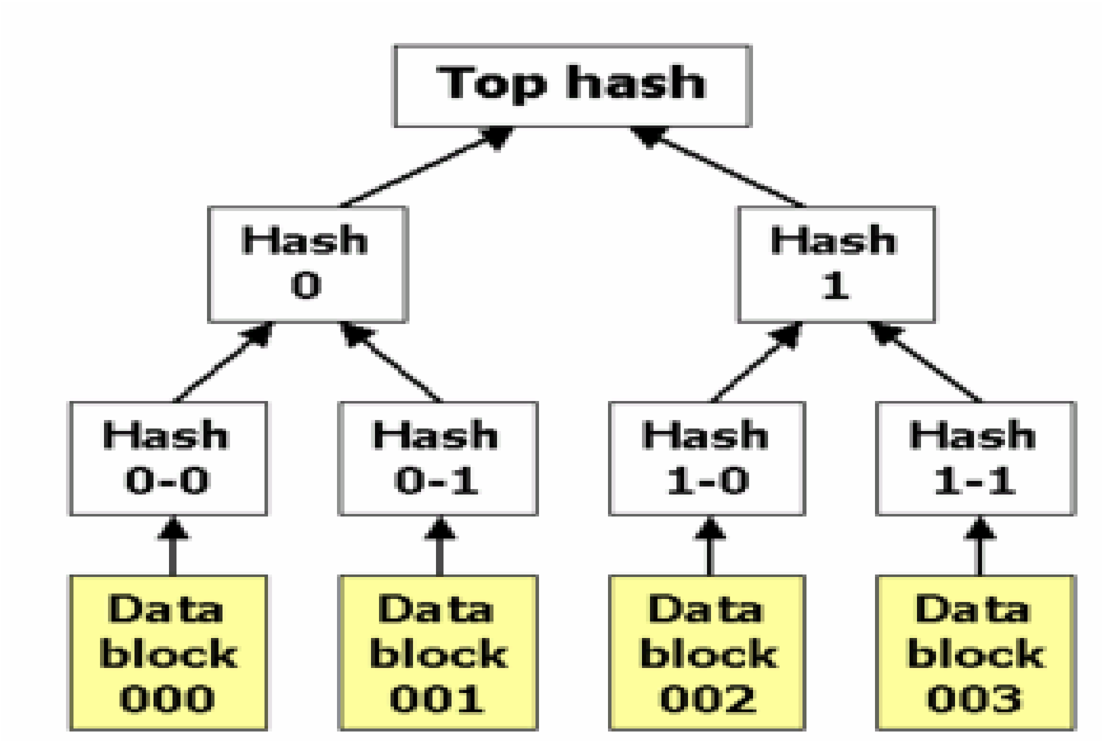
- Privacy
  - Safeguard confidential data against unauthorized accesses.
  - Rely on data encryption.
- Integrity
  - Ensure query results are the same as if the data owner would have produced them.
    - Inclusive
    - Complete

Current approach – signature based



- Hash List

Current approach – signature based



- Merkle tree

Authenticate data structure based approaches

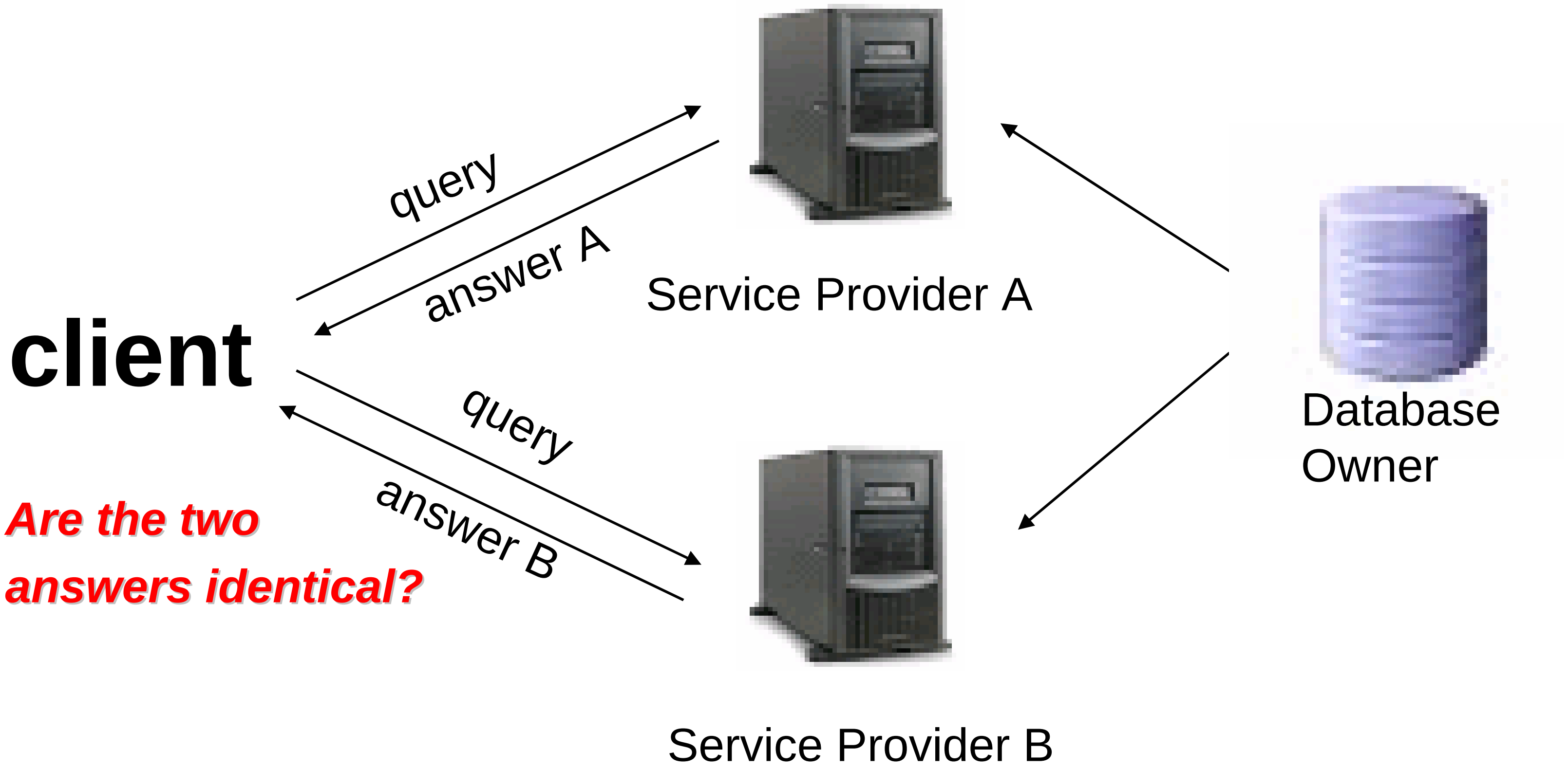
- For large databases, a lot of signatures need to be maintained.

- How to do join?

$$\text{SELECT *}$$
$$\text{FROM } \mathbf{T_1} \text{ AND } \mathbf{T_2}$$
$$\text{WHERE } \mathbf{T_1.B} = \mathbf{T_2.B}$$

- Changes must be made in DBMS engines to support the scheme.

Think out of the box – step 1
Cross examination

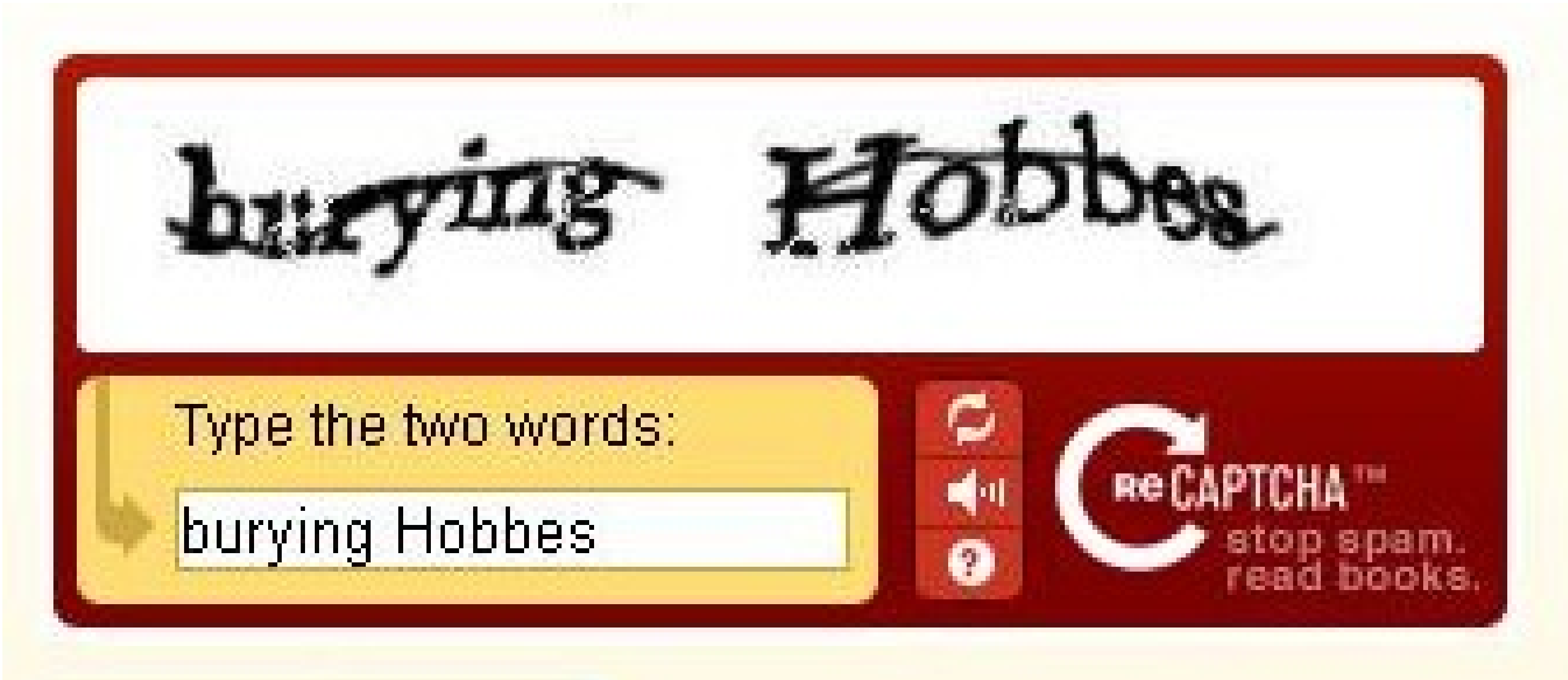


Are the two answers identical?

It opens a can of worms

- Security
  - What if the two service providers conspire in cheating?
- Cost
  - Using two service providers incurs double cost.
  - Run time cost is also high.

Think out of the box – step 2
Are you human?



It is a probabilistic approach!

- Wrong doings may be caught immediately
  - The answer to the known word is wrong
- There is a chance that wrong doings can evade detection
  - The answer to the known word is correct
- In the long run, the probability that wrong doings can evade detection is very small
  - If it's only guessing at which word is the known word

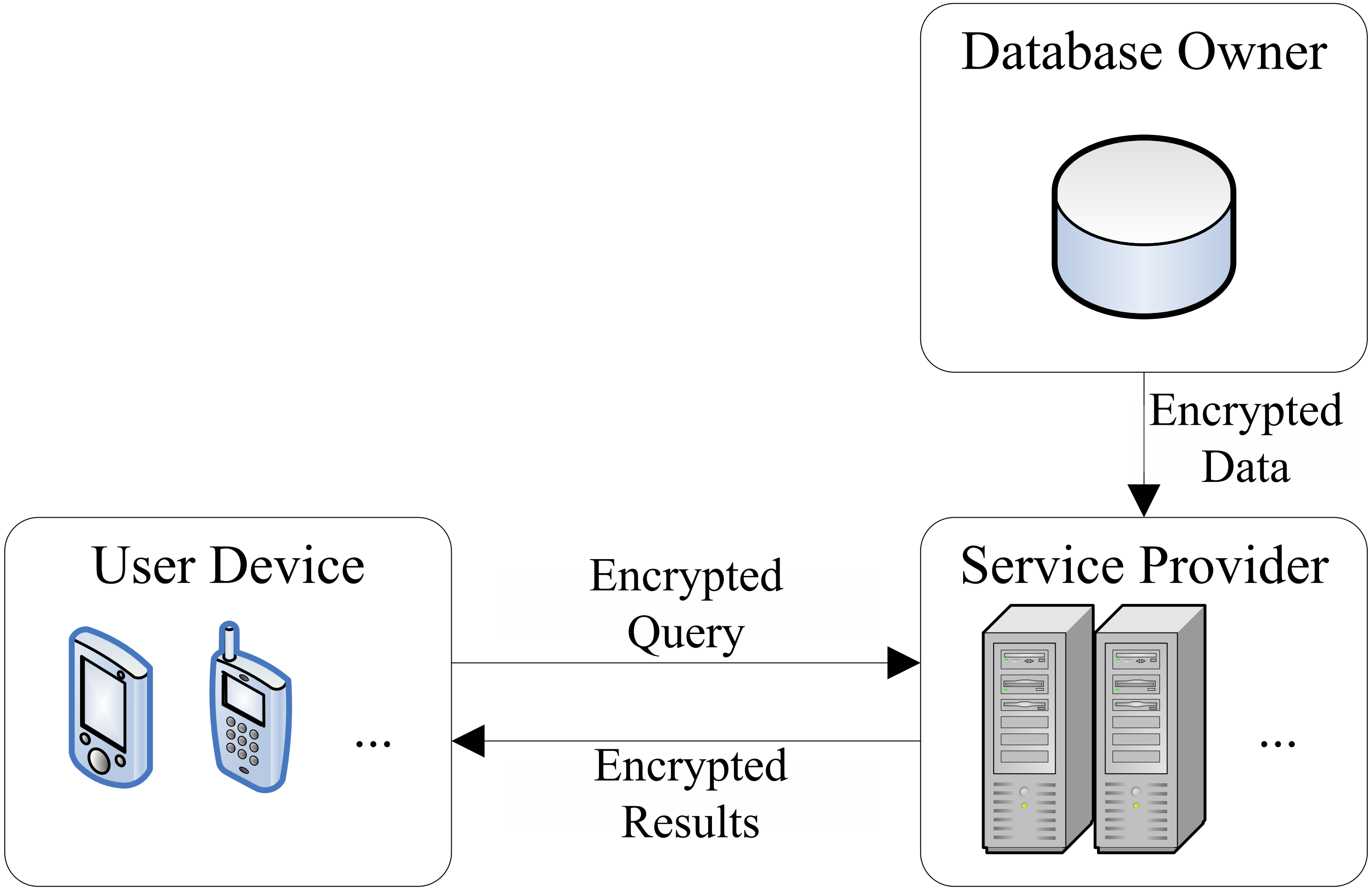
Our approach

- Add a small set of “fake” tuples to the database.
- Encrypt the database for privacy.
- The attackers do not know what tuples are “fake”.
- The service provider executes queries in a DBMS (with support of encryption).
- All “fake” tuples that satisfy the query must be returned.

Our approach

- No need to maintain local databases
  - We do not store “fake” tuples.
- Deterministic “faking”
  - We use a function, which is determined by a secret key, to generate “fake” tuples.
- Low cost
  - Each client remembers only the function
- It is extendible
  - For joins, updates, etc.

Architecture



Privacy

- Encryption w/ order preserving features
  - Orthogonal to our work

- Executing SQL over encrypted data in the database-service-provider model *[Hacigumus, SIGMOD 2002]*

- Order-preserving encryption for numeric data *[Agrawal, SIGMOD 2004]*

- Multi-dimensional range query over encrypted data *[Shi, Oakland 2007]*

Privacy

- Our approach is based on the **Order-Preserving Encryption** (OPE) scheme
  - Every attribute is encrypted using OPE independently
  - Only authorized users/administrators have the key

Protect data from being tampered

- We encrypt data using OPE
  - A tuple t $(a_1, a_2, \ldots, a_n)$ is encrypted to t' $\{a_1', a_2', \ldots, a_n'\}$

- We assume the tuple has an additional field, which allows us to easily check the authenticity of the tuple. For example, the field can be computed as :

$$H(a_1 \oplus a_2 \oplus \ldots \oplus a_n)$$

where H is a one-way hashing function.

Query completeness

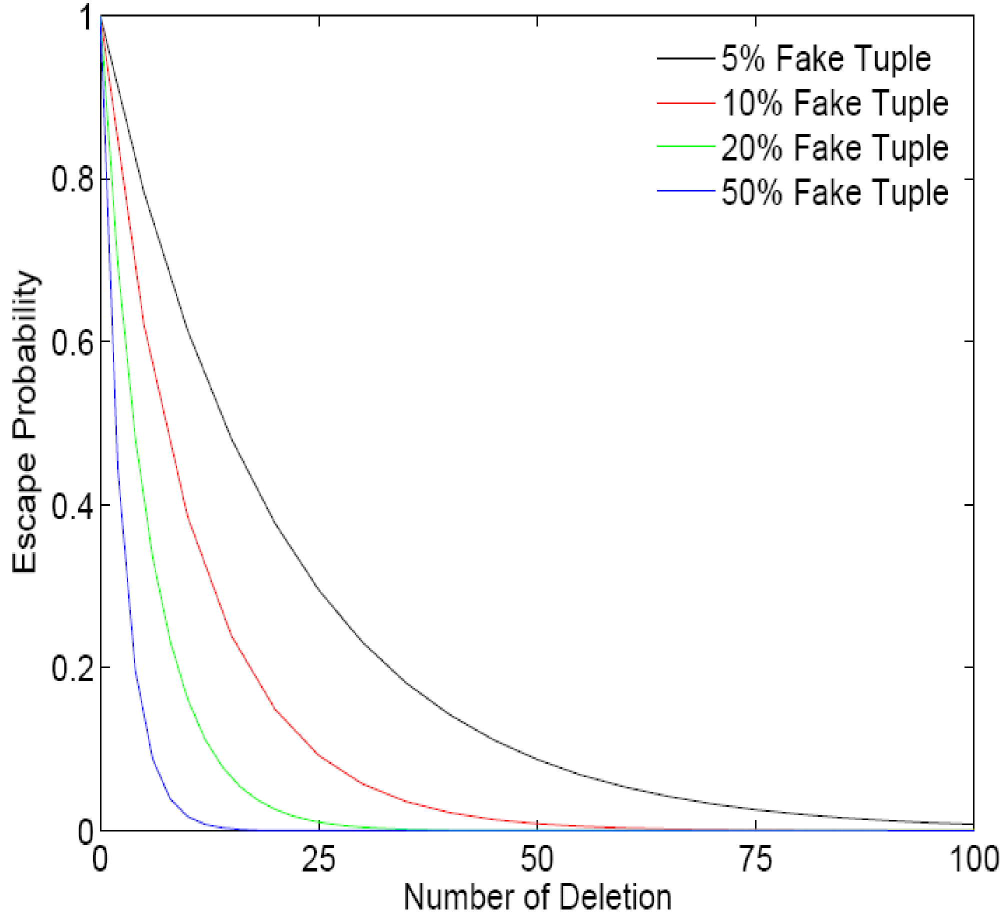
- Database has N tuples.
- We embed K “fake” tuples in the database.
- If fake tuples covered by a query do not appear in the results, we know there is an attack.
- There is a probability that the attacker can escape from being caught.

Analysis

- If a tuple is deleted by an attacker, it has the probability of $\frac{N}{N+K}$ not being caught.

- The probability of not being caught after m attacks is

$$\prod_{i=0}^{m-1} \frac{N-i}{K+(N-i)}$$



N = 1,000,000

Distinguish fake tuples from real ones

- Our scheme won't work if attackers can tell fake tuples from real ones
  - It only need to query against fake tuples

- It is easy for the client, who knows the key, to distinguish fake tuples from real ones

$$checksum = \begin{cases} H(a_0 \oplus a_1 \oplus \ldots \oplus a_n) & \text{Real tuple} \\ H(a_0 \oplus a_1 \oplus \ldots \oplus a_n) + 1 & \text{Fake tuple} \end{cases}$$

Any fake tuples missing in the query result?

- Let Q be a query.
- Let $C_s(Q)$ be the set of fake tuples in the query result sent back by the server
- Let $C_c(Q)$ be the set of fake tuples that satisfy the query
- **Integrity check:** $C_c(Q) = C_s(Q)$?
- If $|C_c(Q)| <> |C_s(Q)|$, then there is definitely a problem.
- If $|C_c(Q)| = |C_s(Q)|$, do we need to compare the two sets for equality?

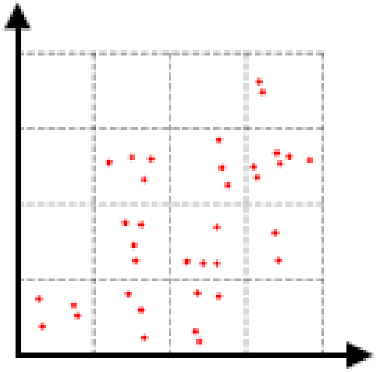
Any fake tuples missing in the query result?

THEOREM 1. *If* $|C_s(Q)| = |C_c(Q)|$, *then* $C_s(Q) = C_c(Q)$.

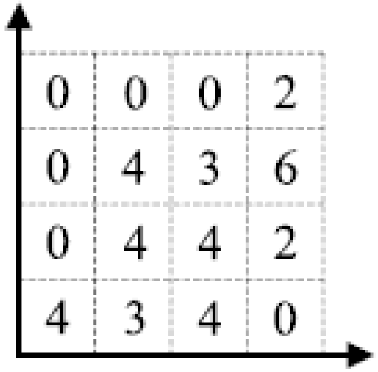
PROOF. Assume to the contrary $C_s(Q) \neq C_c(Q)$. Since $|C_s(Q)| = |C_c(Q)|$, then $\exists t \in C_s(Q)$ such that $t \notin C_c(Q)$. But $t \in C_s(Q)$ means t is a checking tuple, whose authenticity is guaranteed by the one-way hash function, and since t satisfies Q, t must appear in $C_c(Q)$. $\square$

Fake tuple distribution

- Data distribution is important to security
  - Use a multi-dimensional histogram to catch original data's distribution.
  - Match the distribution of fake tuples with that of real tuples.
- Query distribution is important to level of integrity assurance
  - Do queries follow a random distribution, or the data distribution?



(a)



(b)

How to generate fake tuples?

- A Naïve approach
  - Randomly generate fake tuples under distribution of the real data
  - Maintain a copy of fake tuples at each client
  - When a query Q is send to the server, also run Q on the client site.
  - Check whether $|C_s(Q)|$, the count of fake tuples in the query result provided by the server, is equal to $|C_c(Q)|$, the count of fake tuples the client finds out.
- Drawbacks:
  - Maintaining database locally is against the purpose of database outsourcing

Deterministic Methods

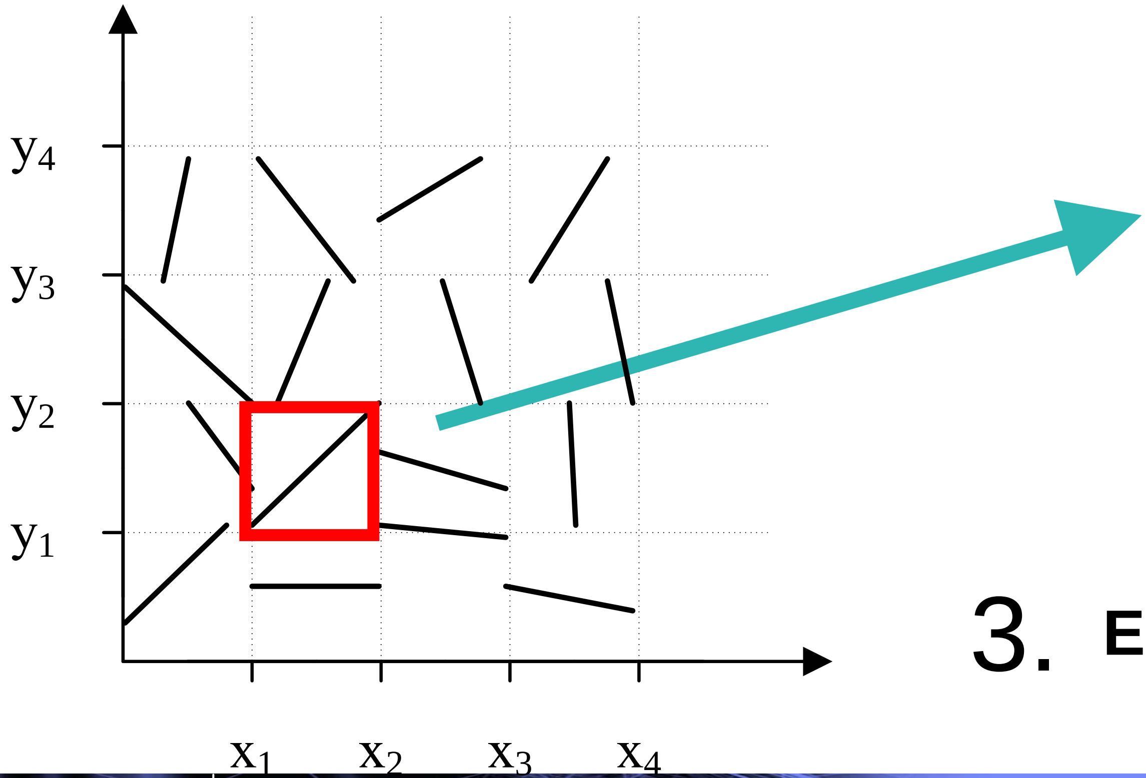
- Choose a family of functions
  - e.g., linear functions, quadratic functions
- Randomly pick a key, which determines a function in the family
  - e.g., coefficients of the linear/quadratic functions
- Each client remembers the function
  - Little storage cost
  - Efficient to find the count of fake tuples that satisfy a query

How about distribution?

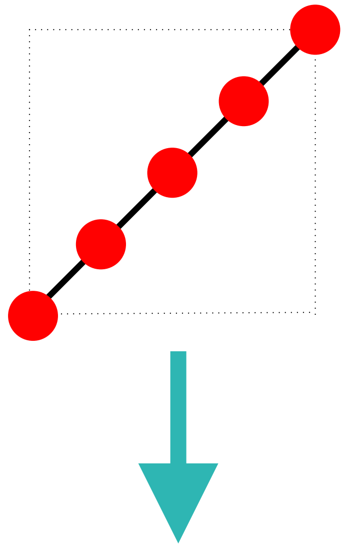
- Divide the feature space into grids
- Capture the distribution of the real data (count of tuples in each grid)
- Create a key (hence a deterministic function) for each grid
- The count decides how many tuples the function generates for that grid

Fake tuple generation

1. **Choose a Function**

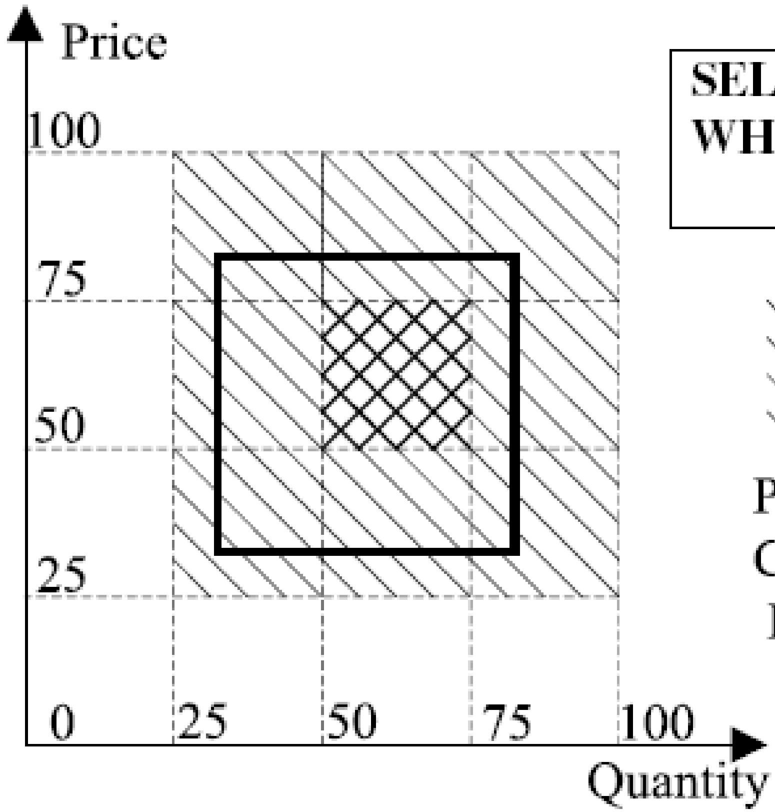


2. **Generate fake tuples**

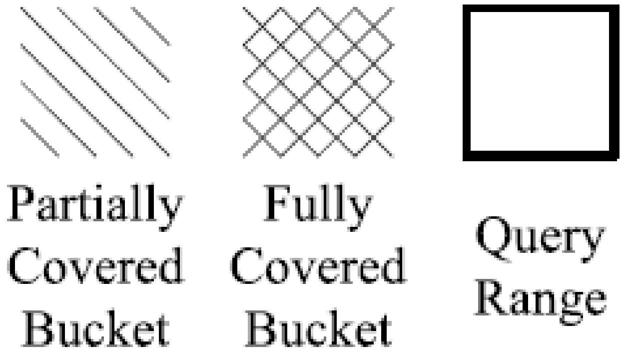


3. **Encrypt**

Checking query integrity



SELECT * **FROM** *expTable*
WHERE *Price* between 30 and 80 and
Quantity between 30 and 80



Proof of security

ε-distinguisher

Let $\varepsilon > 0$ and let f_0 and f_1 be two functions selected from two different function families F_0 and F_1 uniformly randomly.

A distinguisher A is an algorithm; given a function, A outputs 0 or 1 as it determines whether the function is from F_0 or F_1.

Let Adv_A denote A's advantage in distinguishing F_0 and F_1.

$$Adv_A = |Pr[A(f_0) = 1] - Pr[A(f_1) = 1]|$$

We say algorithm A is an ε-distinguisher of F_0 and F_1 if $Adv_A > \varepsilon$.

Proof of security

(q, t, ε)-pseudorandom

A function family F is called (q, t, $\mathcal{E}$)-pseudorandom if

there does not exist an algorithm A that can $\mathcal{E}$-distinguish F from a truly random function.

(A is allowed to use F as an oracle for q queries, and use no more than t computation time.)

Proof of security

Our approach is provable secure

Given a dataset T, we generate a dataset S, and store X=F_k(T∪S) to the server. The highest level of security is achieved if any subset from $F_k(T)$ is indistinguishable from a random subset of *X* to attackers.

We prove: there does not exist an adversary algorithm that can select l tuples from X such that all the l tuples are in T with a possibility bigger than

$$\left(\frac{|T|}{|T|+|S|}\right)^{l} + \varepsilon$$

Integrity assurance of Joins

- **Join two tables** T_1 and T_2

  SELECT *
  FROM T_1 and T_2
  WHERE $T_1.B = T_2.B$

- **We have 4 cases here:**

1. Original tuples from T_1 *join with* original tuples from T_2
2. Fake tuples from T_1 *join with* original tuples from T_2
3. Original tuples from T_1 *join with* fake tuples from T_2
4. **Fake tuples from T_1 *join with* fake tuples from T_2**

Example

T

	A	B
O	1	2
F	2	3
O	3	4
F	4	5

T

	B	C
O	2	2
O	3	3
F	4	1
F	5	10

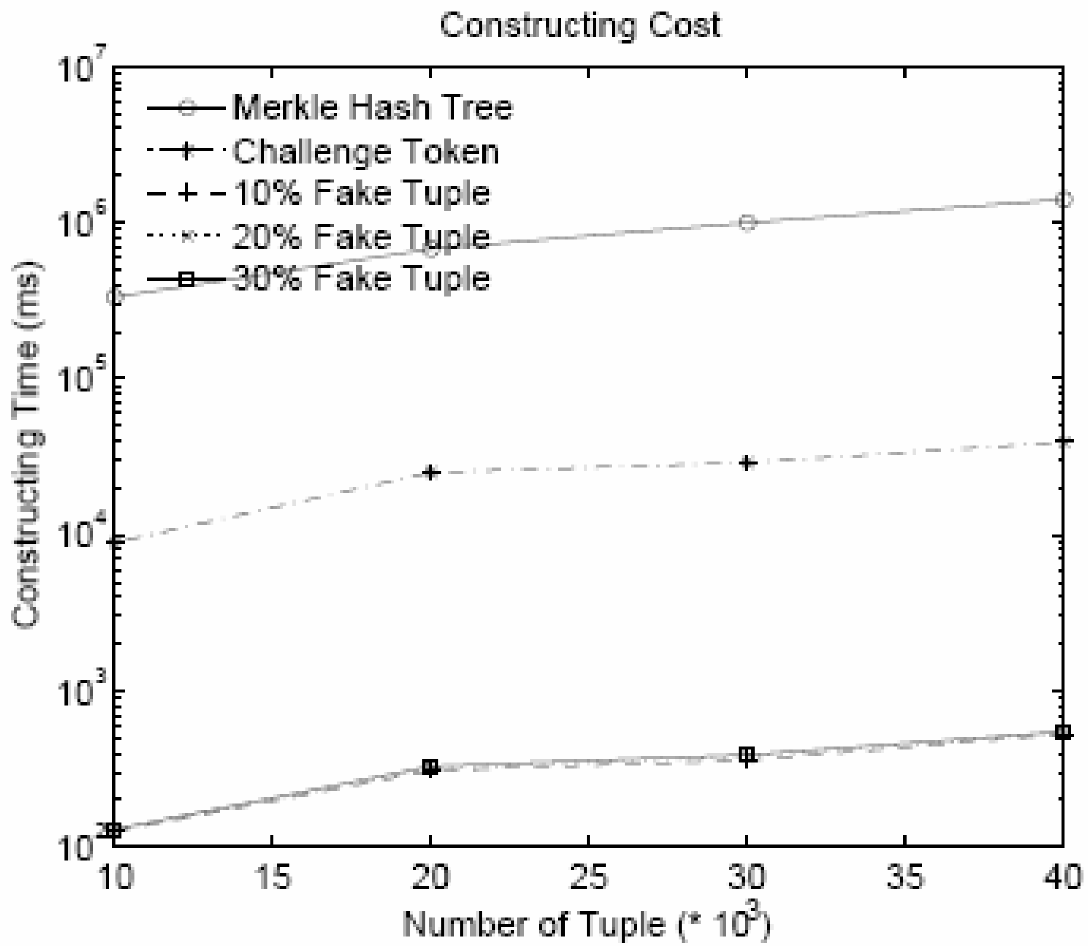
			A	B	C
Case 1	O	O	1	2	2
Case 2	F	O	2	3	3
Case 3	O	F	3	4	1
Case 4	F	F	4	5	10

T $\bowtie$ T

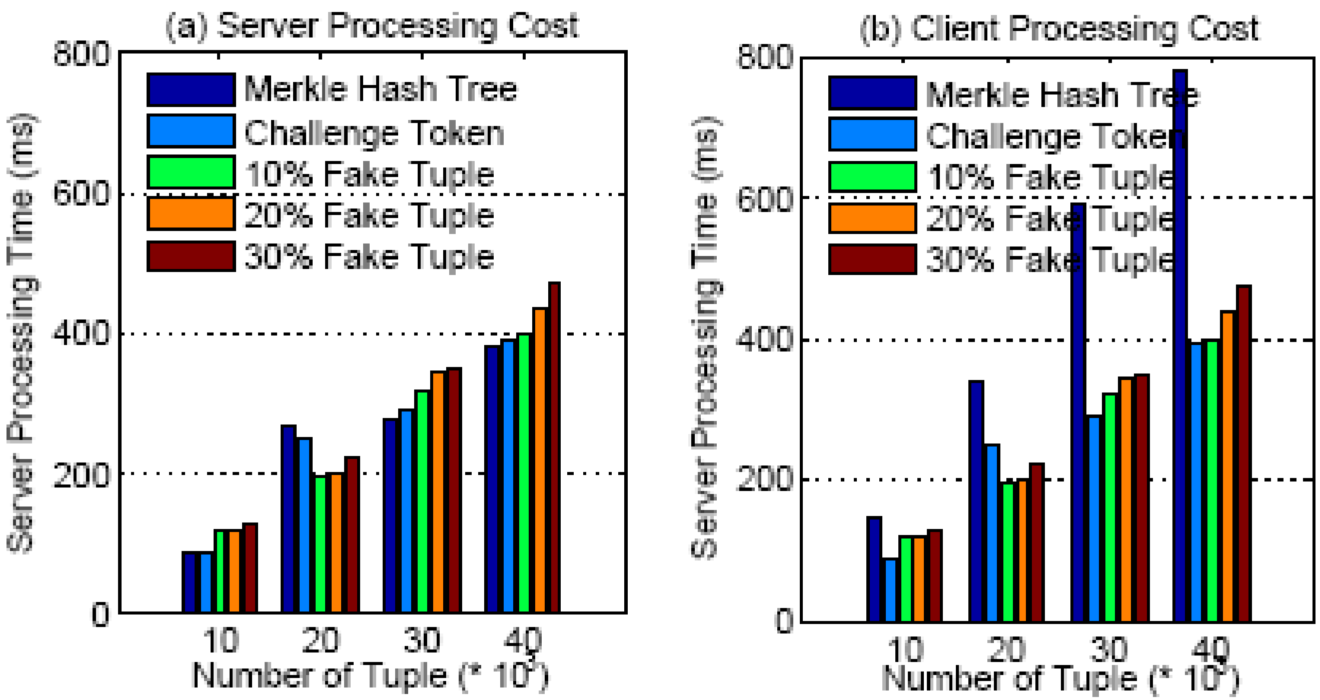
1

Header$_1$

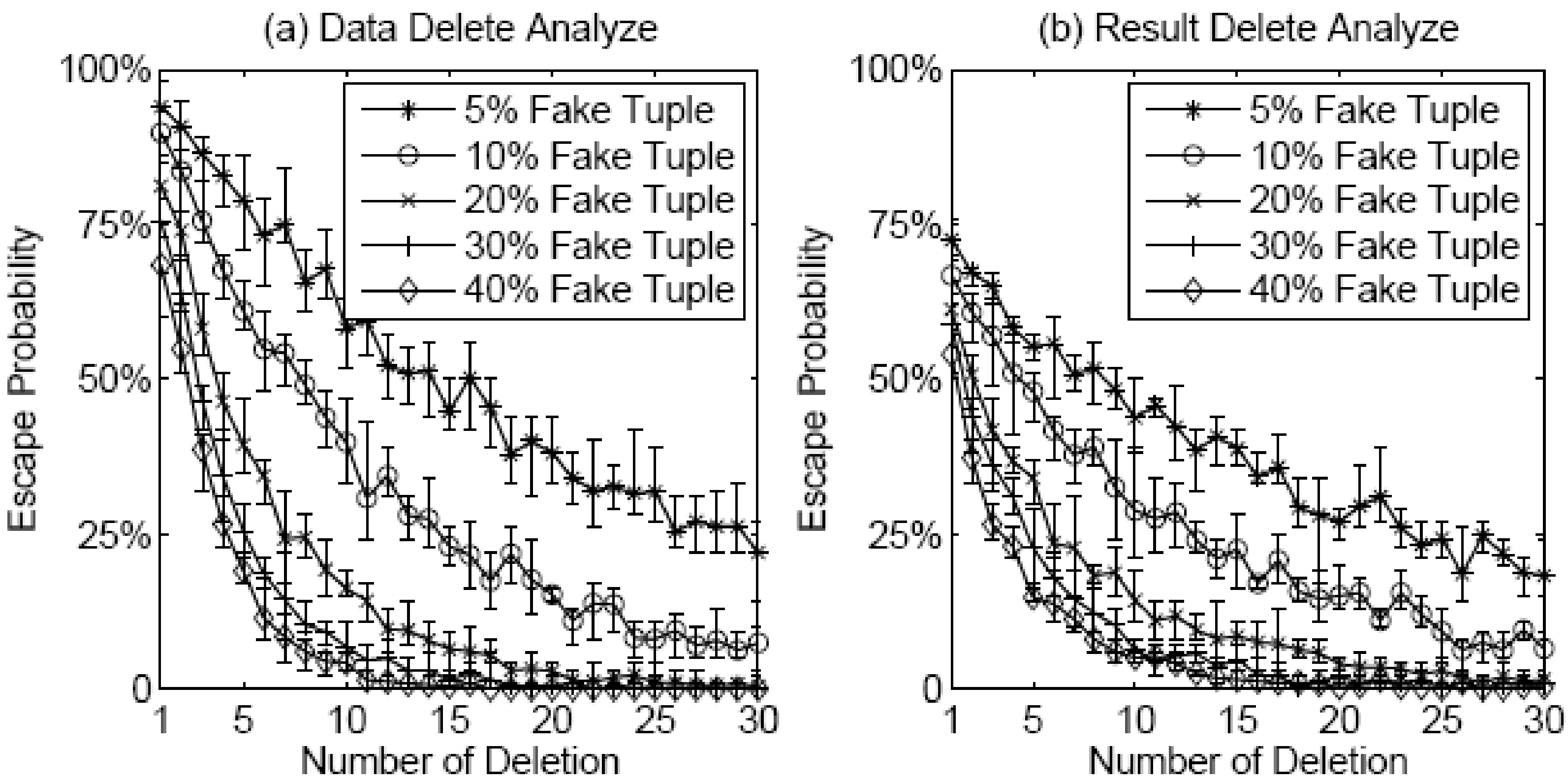
Experiment (1)



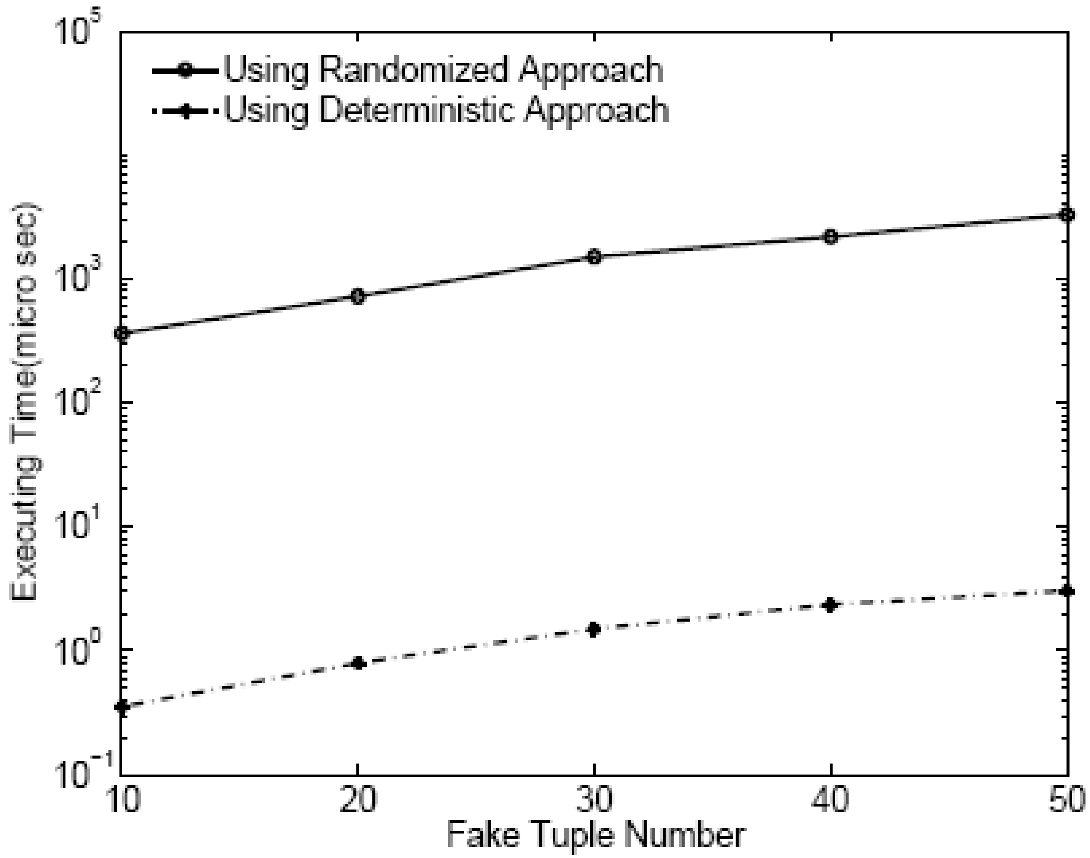
Experiment (2)



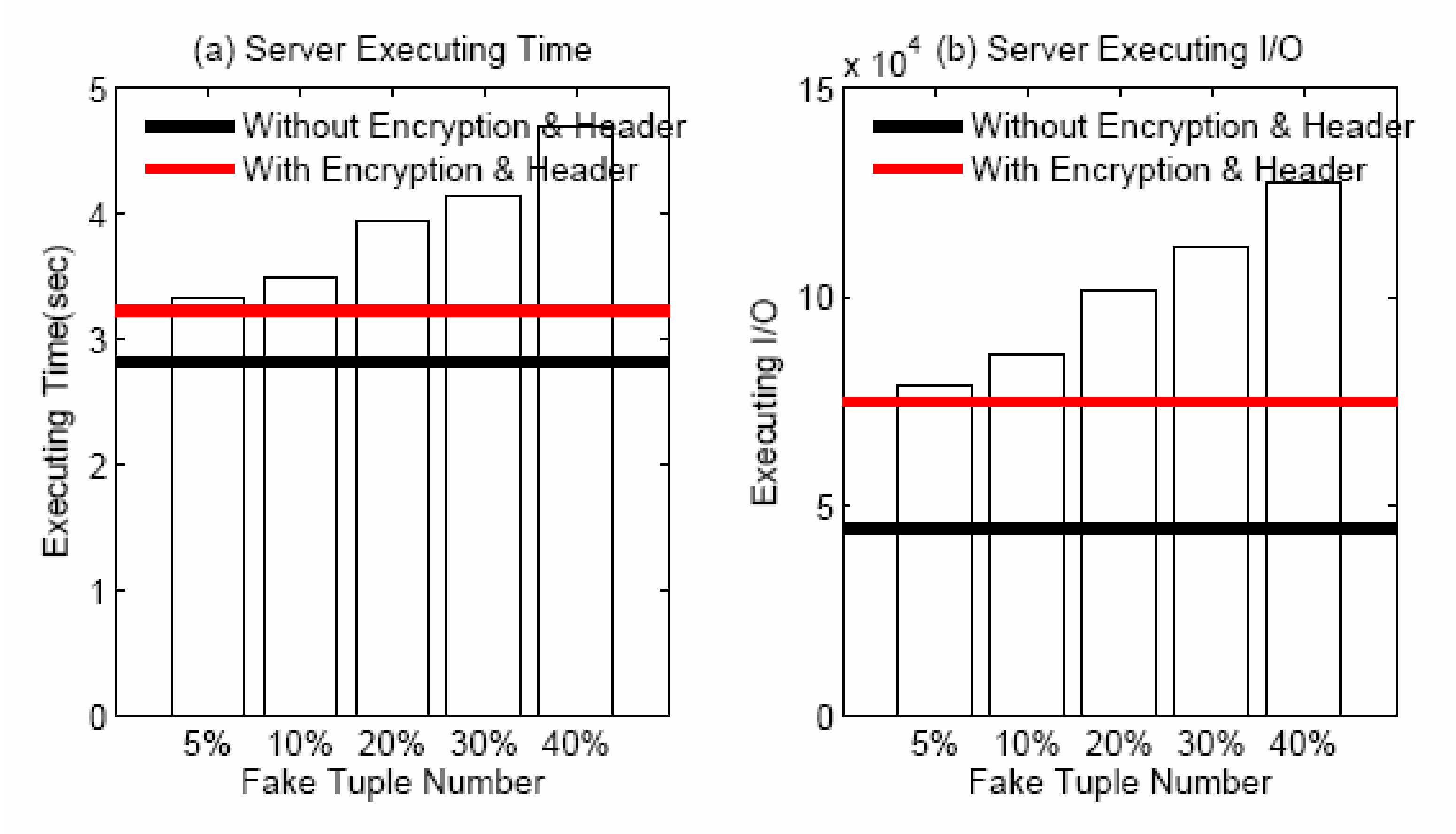
Experiment (3)

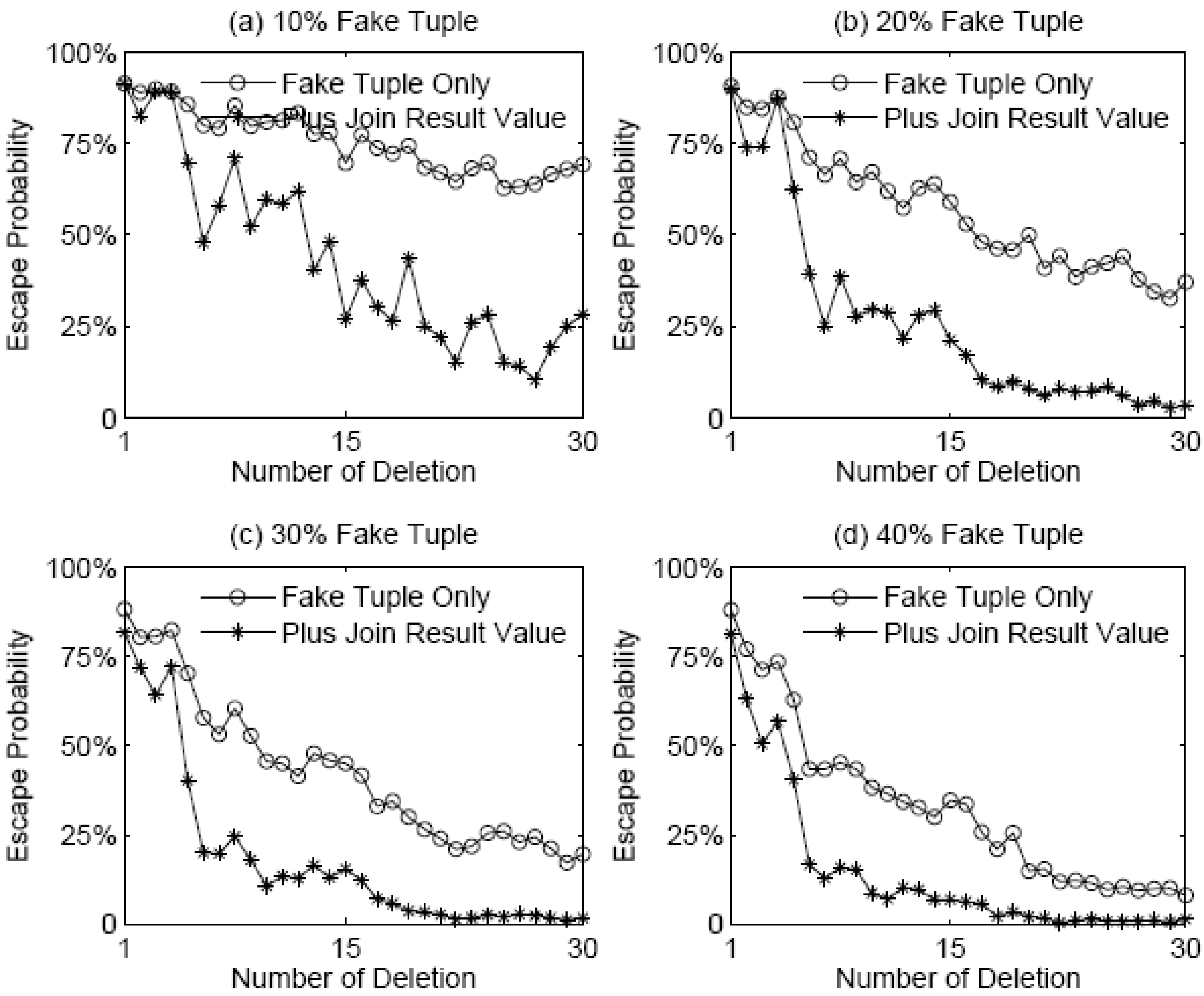


Experiment (4)



Experiment (5)





(a) 10% Fake Tuple
(b) 20% Fake Tuple
(c) 30% Fake Tuple
(d) 40% Fake Tuple
Fake Tuple Only
Plus Join Result Value
Escape Probability
Number of Deletion
100%
75%
50%
25%
0
1
15
30

Future work

- Update queries
  - Merkle tree based approaches
  - Probabilistic approaches
- Aggregate queries
  - sum and max
- Data mining queries
  - e.g., Nearest neighbor search