SecureDB

A Secure Query Processing System in the Cloud

Group Member: Haibin LIN, Eric
Supervisor: Prof Benjamin Kao
Department of Computer Science, University of Hong Kong
Overview

1. The Problem
2. Related Work
3. Theoretical Background
4. System Architecture
5. Component Implementation
6. Experiment Result
Background

Cloud Service Provider (Server)
**Background**

Data Owner (Client) <-> Client App <-> Cloud Service Provider (Server)

**Results**

<table>
<thead>
<tr>
<th>Name</th>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>20000</td>
</tr>
<tr>
<td>Bob</td>
<td>50000</td>
</tr>
</tbody>
</table>
The Problem

Data Owner (Client)

Client App

Query processing is NOT SECURE!

Cloud Service Provider (Server)

Query

Results

Hacker

Administrator

<table>
<thead>
<tr>
<th>Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>20000</td>
</tr>
<tr>
<td>50000</td>
</tr>
</tbody>
</table>
I have to process query myself!
Overview

1. The Problem
2. Related Work
3. Theoretical Background
4. System Architecture
5. Component Implementation
6. Experiment Result
1. Hardware Approach

TrustedDB(2011)[1]
- Based on trusted **secure co-processor**
- Dedicated **hardware** for cryptographic operation
Related Work

Data Owner (Client)

Client App

Key

Query

Encrypted Results

Trusted Hardware

Key

Query

Encrypted Data

Cloud Service Provider (Server)

Untrusted

Salary (Encrypted)

$Aa%df244

F@3dewqD
## Related Work

1. Hardware Approach

   TrustedDB (2011)

<table>
<thead>
<tr>
<th>Advantage</th>
<th>Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Security</td>
<td>Expensive Hardware</td>
</tr>
<tr>
<td>Accepts any kind of query</td>
<td>$$$$$$$$$$$</td>
</tr>
</tbody>
</table>
Related Work

2. Software Approach
   a. Fully Homomorphic Encryption
      - Allows **arbitrary computation** on ciphertext without knowing the key, including +, -, *, /, >, =, √ …
      - Limitation: **Computationally Expensive**
        e.g. 30 minutes per bit operation(2011)[2]
2. Software Approach

b. CryptDB(2012)[3]

- Multiple layers of partially homomorphic encryptions

<table>
<thead>
<tr>
<th>Encryption Layer</th>
<th>E1</th>
<th>E2</th>
<th>E3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operations Supported</td>
<td>None</td>
<td>Equality check</td>
<td>Equality check Ordering comparison</td>
</tr>
<tr>
<td>Security Level</td>
<td>Strongest</td>
<td>Strong</td>
<td>Not secure against CPA</td>
</tr>
</tbody>
</table>
2. Software Approach

b. CryptDB(2012)

- Limitation: supports **limited types of queries**

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Example</th>
<th>Supported?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation</td>
<td>SELECT a * b FROM T</td>
<td>☺</td>
</tr>
<tr>
<td>Comparison</td>
<td>SELECT a, b FROM T WHERE a &gt; b</td>
<td>☺</td>
</tr>
<tr>
<td>Computation &amp; Comparison</td>
<td>SELECT a, b FROM T WHERE a * b &gt; c</td>
<td>😞</td>
</tr>
</tbody>
</table>
What is SecureDB?

- SDB is a secure query processing system based on secret sharing

- Motivation
  1. Runs on commodity hardware
  2. Accepts a wide range of queries
  3. Both efficient and secure!
  4. Less effort for the client
What is SecureDB?

Client

SDB Proxy

Server

Salary (Encrypted)

- $Aa%df244
- F@3dewqD
Overview

1. The Problem
2. Related Work
3. Theoretical Background
4. System Architecture
5. Component Implementation
6. Experiment Result
Secret Sharing

- Secret Sharing Scheme
  - For a sensitive value $V$, we split it into two shares: the \textbf{encrypted value} $V_e$ and the \textbf{item key} $V_k$
  - One needs both $V_e$ and $V_k$ to recover the value of $V$
    \[ V = \text{Decrypt}(V_e, V_k) \]
Secret Sharing

- Secret Sharing in SDB
  - Encrypt sensitive values on a column basis
  - Add helper column $r$ so that client can compute item keys on the fly

$$V_k = \text{genItemKey}(r, <m, x>)$$
Secure Computation Protocol

- For any operation on $V$ (+, -, *, <, >, =), the server can complete the operation without knowing column keys.
- Includes client protocol and server protocol.

Client:
1. Query

SDB Proxy
Key

2. Client Protocol Execution

6. Decrypt Results

Server:

3. Query

4. Server Protocol Execution

5. Encrypted Results

DBMS
Computation Protocol

- Example: Secure protocol for multiplication

1. Client computes a new column key. \[ C_{kc} = \langle m_A \cdot m_B, x_A + x_B \rangle \]
2. Server computes on the bulk encrypted data. \[ C_e = A_e \cdot B_e \mod n \]
3. Finally, client decrypts the encrypted result with Ckc
Challenge

- Every basic operator (e.g. *, +, >) has a unique protocol
- How to automate the execution process?
  1. Build a new DBMS from scratch? Or
  2. Incorporate these protocols with an existing database system?
Overview

1. The Problem
2. Related Work
3. Theoretical Background
4. System Architecture
5. Component Implementation
6. Experiment Result
System Architecture

- SparkSQL: a cluster computing engine that supports SQL
- User Defined Function (UDF) & Query Rewrite

1. select A * B from T
3. select `sdb_mul(A,B, ...), row_id` from T

Figure 5: Architecture of SDB
Why Query Rewrite & UDF?

1. Performance wise
   ● User Defined Function executed in the same address space of SparkSQL
     => Little memory copy, little network transfer and no IPC

2. Engineering wise
   ● Normal operators provided by SparkSQL
   ● Server side queries optimized by SparkSQL
   ● Machine failures, disk-based processing and parallelism handled by SparkSQL
Overview

1. The Problem
2. Related Work
3. Theoretical Background
4. System Architecture
5. Component Implementation
6. Experiment Result
SDB Proxy Components

Components of SDB Proxy

- Connector
- Key Store
- Query Processor

Currently supports +, -, *, >, =, <, count().
~18000 lines of Java code
Query Parser

- Parse query **strings** into **abstract syntax trees**

SELECT quantity * price
FROM product
Semantic Analyser

- Transform abstract syntax trees into logical plan trees, access key store to
  1. Verify if column is valid / sensitive
  2. Annotate sensitive columns with column keys
Query Rewriter

1. Identify and rewrite secure operators
2. Transform logical plan trees into physical plan trees
1. Submit rewritten queries to SparkSQL
2. Decrypt encrypted results
3. Return plaintext results via connector
Overview

1. The Problem
2. Related Work
3. Theoretical Background
4. System Architecture
5. Component Implementation
6. Experiment Result
Security Analysis

Security threats

- **Database (DB) Knowledge** – See encrypted values stored on servers’ disks
- **Chosen Plaintext Attack (CPA) Knowledge** – Select plaintext values and observe encrypted values
- **Query Result (QR) Knowledge** – See queries submitted and the encrypted results
Security Analysis

Security Level in SDB

• SDB generates 2048-bit column keys similar to RSA
• SDB is secure against DB + CPA threat and DB + QR threat
• Limitation: secret sharing doesn’t support floating point numbers
Decrypt-Before-Query Approach

Data Owner (Client)

Query processing is NOT FAST!

Client App → Query → Query Processor → Encrypted Data → Query Results → Client App

Cloud Service Provider (Server)

Salary (Encrypted)

$Aa%df244
F@3dewqD
Importance of Secret Sharing

- Compare with Decrypt-before-query (DBQ)
- Experiment Environment
  - Client: 1 CPU
  - Server: 8 CPU X 10 Machines
- Result
  a. Total Cost: SDB < DBQ
  b. Client Cost: SDB << DBQ

SELECT A, B FROM T WHERE A < p, 1% selectivity
Query Cost Breakdown

- Server cost >> client cost
- Decrypt cost >> other client cost
- Future work: Encryption/Decryption optimization

SELECT A, B from T WHERE A < q
Overview of Secure Operators

- **Compare with SparkSQL**
  - Execute on plaintext, bypassing all secure operators
  - Three types of queries
    - EC Range: SELECT A, B FROM T WHERE A < 100
    - EE Range: SELECT A, B FROM T WHERE A < B
    - Count: SELECT count(A) FROM T WHERE A < 100

- **Result**
  - ~180 times slower
  - Computation cost of modular exponential is high
  - Future work: UDF optimization

![Graph showing comparison between TDB and TSpark](image)
SQL Editor

Query:

select a, b, c, d from demo where a < 1000

Execute SQL Query

Running Time: 7269 ms

Query Result

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>b</td>
<td>c</td>
<td>d</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>628</td>
<td>7551</td>
<td>1148</td>
<td>VFVP7627</td>
</tr>
<tr>
<td>800</td>
<td>4890</td>
<td>8919</td>
<td>JHAF5328</td>
</tr>
<tr>
<td>689</td>
<td>3602</td>
<td>3678</td>
<td>OMZZ1789</td>
</tr>
<tr>
<td>571</td>
<td>6634</td>
<td>9396</td>
<td>CIWD3437</td>
</tr>
<tr>
<td>976</td>
<td>5028</td>
<td>8003</td>
<td>PSZK2748</td>
</tr>
<tr>
<td>460</td>
<td>6222</td>
<td>8787</td>
<td>RAMD4384</td>
</tr>
</tbody>
</table>
SQL Executed on remote cluster server:
```
select demo.a, demo.b, demo.c, demo.d, demo.row_id from demo where
  sdb_compare(sdb_keyupsdb_mulldemo.r, sdb_adddemo.a, demo.s, demo.s, "hh1lccfurrj8bzursh3yyk80ktcve5588zl80pj5r3xix8w8tn7snn6v
  "gm2ulme39lzl0ap8q2zffwshbolhdry79ge2ofv6dorwnjetceppfib16s5w9iizl0nvsp1zyjfczzj047l99g9w2tex33ufaedtv3v017wkj6o6zw5ziif
  < 0
```
Total number of results: 486

**Server Execution Time (%): 93**

- **Client Parse Time:** 5 ms
- **Client Analyse Time:** 21 ms
- **Client Rewrite Time:** 34 ms
- **Client Execute Time:** 517 ms
- **Server Execute Time:** 6692 ms
Future Work

- **Query expressiveness extension**
  - Join, Cartesian product, SUM(), AVG()
  - GroupBy, Having Clause

- **Crypto optimization**
  - Encryption/Decryption optimization
  - UDF optimization
Query Cost vs. Data Size

- SELECT A, B from T WHERE A < q
- SELECT A, B from T WHERE A < B
- SELECT COUNT(A) from T WHERE A < q
More on Query Rewrite

- What if multiple secure operators are involved?

\[ R \ast (A - B) > 0 \]
More on Query Rewrite

- What if multiple secure operators are involved?

```
sdb_compare(sdb_keyup(sdb_mul(r, sdb_add(a, b, ..), ..), ..), ..)
```
Demo Video
Reference

