SMCQL: Secure Querying for Federated Databases

Raluca Ada Popa

Oct 1, 2019

Some slides are adapted from Jennie Rogers, adding my views
The challenge

• Cheap computing and storage means people record and process enormous amounts of data at different data owners (DOs)

• DOs do not wish to share information with one another often owing to privacy concerns

SMCQL proposes an architecture for database federations for combining the private data of multiple parties for querying
Private Data Federations

- Querying the private records of many DOs with a unified SQL interface
- A DO will not reveal info about their sensitive data to others, but is willing to enable a client to learn certain query results over all DOs
- Client issues queries in SQL
- Built-in security policy
Threat model

- Honest-but-curious DOs

- Honest broker plans and orchestrates queries over the DOs on behalf of the client (the broker is not strictly needed)
SQL 101
Databases

- **Structured** collection of data
  - Often storing tuples/rows of related values
  - Organized in tables

<table>
<thead>
<tr>
<th>Customer</th>
<th>AcctNum</th>
<th>Username</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1199</td>
<td>zuckerberg</td>
<td>35.7</td>
</tr>
<tr>
<td></td>
<td>0501</td>
<td>bgates</td>
<td>79.2</td>
</tr>
<tr>
<td></td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
SQL

• Widely used database query language
  • (Pronounced “ess-cue-ell” or “sequel”)

• Fetch a set of rows:

  \[
  \text{SELECT column FROM table WHERE condition}
  \]

  returns the value(s) of the given column in the specified table, for all records where \textit{condition} is true.

• e.g:

  \[
  \text{SELECT Balance FROM Customer WHERE Username='bgates'}
  \]

  will return the value 79.2
SQL (cont.)

• Can add data to the table (or modify):

INSERT INTO Customer VALUES (8477, 'oski', 10.00);

<table>
<thead>
<tr>
<th>AcctNum</th>
<th>Username</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1199</td>
<td>zuckerberg</td>
<td>35.7</td>
</tr>
<tr>
<td>0501</td>
<td>bgates</td>
<td>79.2</td>
</tr>
<tr>
<td>8477</td>
<td>oski</td>
<td>10.00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
SQL (cont.)

• Can delete entire tables:
  
  \[
  \text{DROP TABLE Customer}
  \]

• Issue multiple commands, separated by semicolon:
  
  \[
  \text{INSERT INTO Customer VALUES (4433, 'vladimir', 70.0); SELECT AcctNum FROM Customer WHERE Username='vladimir'}
  \]
  
  returns 4433.
**Join tables**

```sql
SELECT Username, Car from Customer, Cars where Customer.Username = Cars.uname WHERE Balance > 70;
```

Result: (bgates, Tesla)

<table>
<thead>
<tr>
<th>AcctNum</th>
<th>Username</th>
<th>Balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>1199</td>
<td>zuckerberg</td>
<td>35.7</td>
</tr>
<tr>
<td>0501</td>
<td>bgates</td>
<td>79.2</td>
</tr>
<tr>
<td>8477</td>
<td>oski</td>
<td>10.00</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Car</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toyota</td>
</tr>
<tr>
<td>Tesla</td>
</tr>
<tr>
<td>Honda</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cars</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
</tr>
<tr>
<td>uname</td>
</tr>
<tr>
<td>zuckerberg</td>
</tr>
<tr>
<td>bgates</td>
</tr>
<tr>
<td>oski</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Customer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Car</td>
</tr>
<tr>
<td>uname</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Cars</th>
</tr>
</thead>
<tbody>
<tr>
<td>uname</td>
</tr>
<tr>
<td>zuckerberg</td>
</tr>
<tr>
<td>bgates</td>
</tr>
<tr>
<td>oski</td>
</tr>
</tbody>
</table>
Back to SMCQL
HealthLNK Use Case

A group of healthcare providers, such as HealthLNK in Chicago-area, agree to use their patient records for research.

Each hospital responsible for maintaining confidentiality of patient health records
Running Example: Electronic Health Records

Example in the paper:

<table>
<thead>
<tr>
<th>patient ID</th>
<th>gender</th>
<th>diag</th>
<th>.....</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001</td>
<td>M</td>
<td>blues</td>
<td>.....</td>
</tr>
<tr>
<td>00002</td>
<td>F</td>
<td>cdiff</td>
<td>.....</td>
</tr>
<tr>
<td>00003</td>
<td>M</td>
<td>X</td>
<td>.....</td>
</tr>
</tbody>
</table>

I have concerns about patient ID really being public, but let’s assume so for as in the paper
Clinical Data Research Network

“How many patients are there?”

SELECT COUNT(DISTINCT patient_id) FROM diagnosis;
Issues with Currently Deployed Systems

• Need to trust honest broker unconditionally
• Network traffic between honest broker and data providers leaks info on secret data to curious observers
Clinical Data Research Network

“How many patients suffer from rare disease X?”

SELECT COUNT(DISTINCT patient_id) FROM diagnosis WHERE diag=X;

I can’t share that private data!

I can’t share that private data!

I can’t share that private data!
Goal: simulate a completely trustworthy third party to query private datastores

Honest Broker

How many patients suffer from X?

Query results

Analyst
**SMCQL**

- Sensitive query evaluation carried out *in-situ* among DOs using secure multiparty computation (SMC)
- Generates hybrid SMC/plaintext query execution plans

- Differential privacy: can be used complementarily to hide any one record in the final query result
SMC Building Blocks

• Secure query execution is **oblivious** – it reveals nothing about the data to parties other than the result

• *Garbled circuits*
  • Cryptographic protocol used to securely compute a function across two parties
  • Protects a query’s program traces from snooping

• *Oblivious RAM (ORAM)*
  • Shuffles data on all reads/writes to prevent DO from learning memory traces of secure computation
  • $O(\log^2 n)$ bandwidth per I/O

• *ObliVM*
  • Converts imperative code into garbled circuits and ORAM
  • We use it to translate a query’s DB operators into SMC

There are better MPC/SMC tools these days, so consider substituting those
**SMCQL Architecture**

SMCQL is for two mutually distrustful data owners.

![Diagram showing SMCQL Architecture](image-url)
Setting and Trust Model

- Analysts alone view the output of their queries
- Data providers learn nothing about the private records of their peers
- Query results are either precise or differentially-private
- All data providers support a shared schema definition
- Column-level security policy initialized before first query
SQL Supported

- Filter
- Projection
- Join: equi-joins, theta joins
- Cross products
- Aggregates (inc. group-by)
- Limited window aggs
- Distinct
- Sort
- Limit
- Common table expressions

```
\text{COUNT}(*)
\text{DISTINCT}
\sigma_{\text{diag}=\text{hd}}
\sigma_{\text{med}=\text{aspirin}}
\text{diagnosis}
\text{medication}
```

Figure 7 displays the runtime for each query end-to-end. The first approach is a baseline of fully secure execution with no optimizations. The second approach, SMC minimization, evaluates optimizations that reduce the subtree of a query's plan. In comparison to a hypothetical federated database where the query is executed securely and the data processed therein, these results show the system performance with the previous optimizations plus sliced execution. With the split operator at the root, the system's performance has been shown to be competitive with that of federated databases.

It is clear that our baseline execution is very slow, even for modest data sizes. Leveraging the PDN's security policy is important for end-to-end performance. The samples were taken uniformly at random, with the restriction that each sample has at least one distributed slice. The results in this section are from query executions over samples of HealthLNK data with 50 tuples per data provider. We now verify that the Aspirin prescription begins in plaintext with scans on the medical attributes. The optimizer's heuristics with three tests. Aspirin count is executed securely and the data processed therein. For recurrent c. disease, this tests split operators. In comparison to a hypothetical federated database where the query is executed securely and the data processed therein, these results show the system performance with the previous optimizations plus sliced execution.

The baseline has the same configuration as the results in Section 4.1. The second approach, SMC minimization, evaluates optimizations that reduce the subtree of a query's plan. In comparison to a hypothetical federated database where the query is executed securely and the data processed therein, these results show the system performance with the previous optimizations plus sliced execution. With the split operator at the root, the system's performance has been shown to be competitive with that of federated databases.
**COMORBIDITY**
SELECT diag, COUNT(*) cnt
FROM diagnoses
WHERE patient_id IN
cdiff_cohort
GROUP BY diag
ORDER BY cnt
LIMIT 10;

**RECURRENT C. DIFF**
WITH rcd AS (  
    SELECT pid, time, row_no() OVER  
    (PARTITION BY pid ORDER BY time)  
    FROM diagnosis
    WHERE diag=cdiff)

SELECT DISTINCT pid
FROM rcd r1 JOIN rcd r2 ON r1.pid =  
r2.pid
WHERE r2.time - r1.time >= 15 DAYS  
AND r2.time - r1.time <= 56 DAYS  
AND r2.row_no = r1.row_no + 1;

**ASPIRIN COUNT**
SELECT COUNT(DISTINCT pid)
FROM diagnosis d  
    JOIN medication m ON d.pid = m.pid
WHERE d.diag = hd AND m.med = aspirin  
AND d.time <= m.time;

ASPIRIN COUNT
Secure multiparty computation is breathtakingly expensive even with small data.
Attribute-level Security Model

- Annotated table definitions—each column has an access control policy
- **Public attribute**
  - Visible to all parties
  - E.g., Lab results, anonymized IDs
- **Protected attribute**
  - Conditionally available to other parties (e.g., k-anonymous)
  - E.g., Age, gender, diagnosis codes
- **Private attribute**
  - Accessible only by originating available to DO
  - E.g., Timestamps, zip codes

K-anonymity is an obsolete and weak privacy notion. I think the protected attribute should not exist.
Generally, attribute-level security is weak because there are correlations between attributes due to their place in the same record and across foreign keys/primary keys relations.

Arrows go from primary key to foreign key. Example: Say that we keep P_ID unencrypted and treatment plans are also unencrypted (e.g., they are generic). If we know that one patient is following a certain treatment, we can infer the other treatments.
Second path analysis [Hinke’88]

Sensitivity inference rule in relational tables:

If an attribute of a table is private, the entire table is private and all tables reachable via primary-foreign key relationships

SMCQL should have used this
Which tables are sensitive here?

Patient, treatment plan and record are sensitive and should not be visible.

Disease, medication and gene can be public, and contain not information about the patients.
COMORBIDITY
SELECT diag, COUNT(*) cnt
FROM diagnoses
WHERE patient_id IN cdiff_cohort
GROUP BY diag
ORDER BY cnt
LIMIT 10;
Query optimizations

• Aim to reduce the amount of computation happening in MPC
• Important lesson when using MPC

• Need to rewrite query planners 😊
Query Optimization: Split Operators

Precompute part of the operator locally

Partial count(*) #1

Partial count(*) #2
Security Type System

- Taint analysis
- Trace the flow of sensitive attributes through the operator tree
- Identify minimal subtree that must be computed securely to uphold security policy
Example:

Recall each hospital has a horizontal partition (e.g., subset of records) of table diagnoses.

**COMORBIDITY**

```sql
SELECT diag, COUNT(*) cnt
FROM diagnoses
WHERE patient_id IN (cdiff_cohort)
GROUP BY diag
ORDER BY cnt DESC
LIMIT 10;
```

- Local filter
- Group by locally and compute local count

Pad intermediate values to public values to avoid leakage.
Query Optimization: Sliced Evaluation

Horizontally partition tuples on public attributes for secure evaluation
Query Optimization: Semi-join

Find single-party slices to eliminate unnecessary secure computation

Honest Broker

Secure Evaluation

Alice

Tuple ID ∈ ID_A ∩ ID_B

Bob

Tuple ID ∈ ID_B ∩ (ID_A ∩ ID_B)

Tuple ID ∈ ID_A ∩ ID_B

Local Evaluation

Local Evaluation

Encrypted Output

Encrypted Output
Example

Assume table diagnosis at a party and medication at another party

**ASPIRIN COUNT**

```sql
SELECT COUNT(DISTINCT pid)
FROM diagnosis d
    JOIN medication m ON d.pid = m.pid
WHERE d.diag = hd AND m.med = aspirin
    AND d.time <= m.time;
```

If pid is not sensitive, what is the split?

If pid is sensitive/encrypted (which I think it should), what is the split?
Example:

**RECURRENT C. DIFF**
WITH rcd AS (  
SELECT pid, time, row_no() OVER  
(PARTITION BY pid ORDER BY time)  
FROM diagnosis  
WHERE diag=cdiff)

SELECT DISTINCT pid  
FROM rcd r1 JOIN rcd r2 ON r1.pid = r2.pid  
WHERE r2.time - r1.time >= 15 DAYS  
AND r2.time - r1.time <= 56 DAYS  
AND r2.row_no = r1.row_no + 1;

If pid is not sensitive, what is the split?

If pid is sensitive/encrypted (which I think it should), what is the split?
**SMCQL Query Planner (at the honest broker)**

```sql
SELECT COUNT(DISTINCT pid)
FROM diagnosis d
JOIN medication m ON d.pid = m.pid
WHERE d.diag = hd AND m.med = aspirin
AND d.time <= m.time;
```

---

**Executable Plan**

```
int$dSize[m*n] join(int$sSize[m] lhs, int$rSize[n] rhs) {
    int$dSize[m*n] dst;
    int dstIdx = 0;
    for(int i = 0; i < m; i++) {
        int$sSize 1 = lhs[i];
        for(int j = 0; j < n; j++) {
            int$rSize x = rhs[j];
            if(\$filter(l, x) == 1) {
                dst[dstIdx] = $project;
                dstIdx = dstIdx + 1;
            }
        }
    }
    return dst;
}
```

---

**Query Tree**

**Secure**

COUNT(*)

**Sliced**

DISTINCT

**Diagnosis**

**Secure Plaintext**

- `\sigma_{\text{diag}=\text{hd}}`
- `\sigma_{\text{time} \leq \text{m.time}}`

---

**Figure 7**: System performance on sampled data.

![Diagram](image-url)
Performance on Sampled HealthLNK Data

Minimizing SMC: reducing secure subtree, identifying data that can be evaluated locally

Fully Optimized: using slicing often creates further speedup
System Scale Up

Minimizing the secure subtree enables us to scale to larger inputs.
Secure computation has substantial overhead, and there is fertile ground for optimization in this space.
Conclusions

• Second-path analysis for inferring sensitivity

• Perform as much computation as possible on plaintext

• Query planners need to be redesigned to reason in terms of secure and plaintext computation