SMCQL: Secure Querying for Federated Databases

Raluca Ada Popa

Oct 1, 2019

Some slides are adapted from Jennie Rogers, adding my views

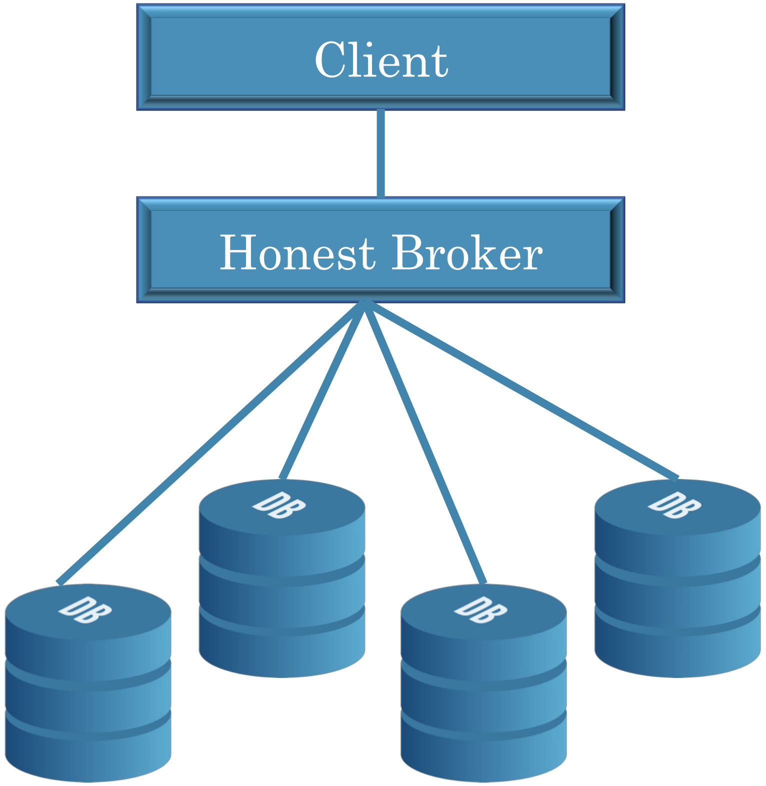
The challenge

- 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

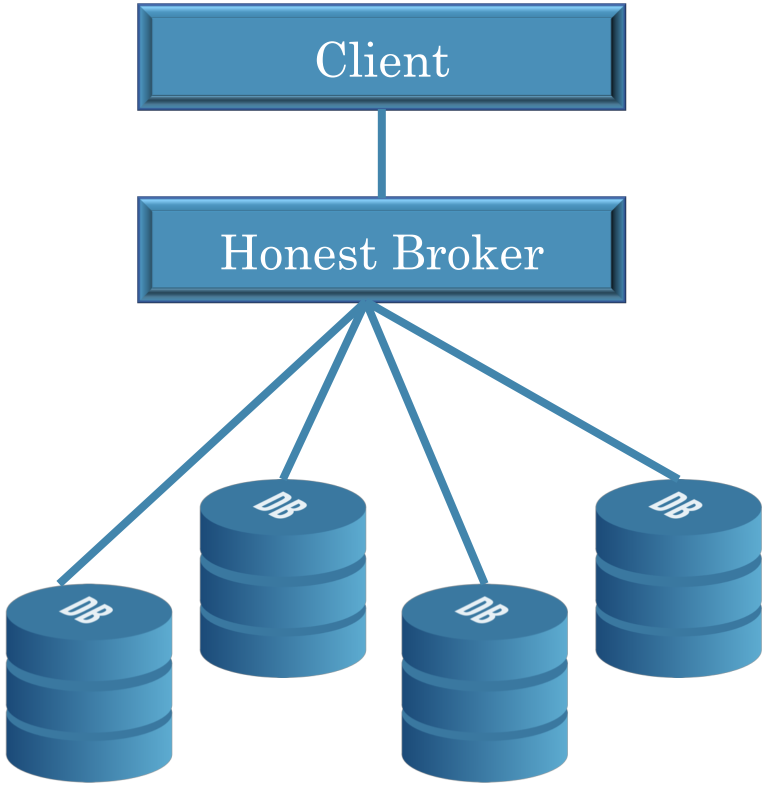


Threat model

- 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

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.7
0501	bgates	79.2
…	…	…

SQL

- Widely used database query language
  - (Pronounced “ess-cue-ell” or “sequel”)
- Fetch a set of rows:

  SELECT *column* FROM *table* WHERE *condition*

  returns the value(s) of the given column in the specified table, for all records where *condition* is true.
- e.g:

  SELECT Balance FROM Customer WHERE Username='bgates'
  will return the value 79.2

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.71
0501	bgates	79.2
…	…	…
…	…	…

SQL (cont.)

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

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

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.7
0501	bgates	79.2
8477	oski	10.00
…	…	…

SQL (cont.)

- Can delete entire tables:

  DROP TABLE Customer

- Issue multiple commands, separated by semicolon:

  INSERT INTO Customer VALUES (4433, 'vladimir', 70.0); SELECT AcctNum FROM Customer WHERE Username='vladimir'

  returns 4433.

Join tables

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

Result: (bgates, Tesla)

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.7
0501	bgates	79.2
8477	oski	10.00
…	…	…

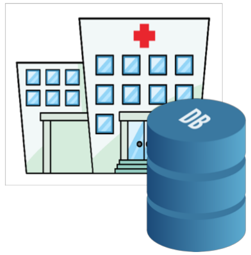
Cars	
uname	Car
zuckerberg	Toyota
bgates	Tesla
oski	Honda
…	…

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



public private private

patientID	sex	diag	.....
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

public private private

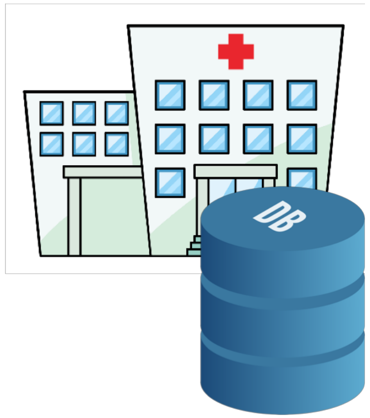
patientID	sex	diag	.....
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

public private private

patientID	sex	diag	.....
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

Running Example: Electronic Health Records

Example in the paper:



public	private	private	
patient ID	**gender**	**diag**	**.....**
00001	M	blues	…..
00002	F	cdiff	…..
00003	M	X	…..

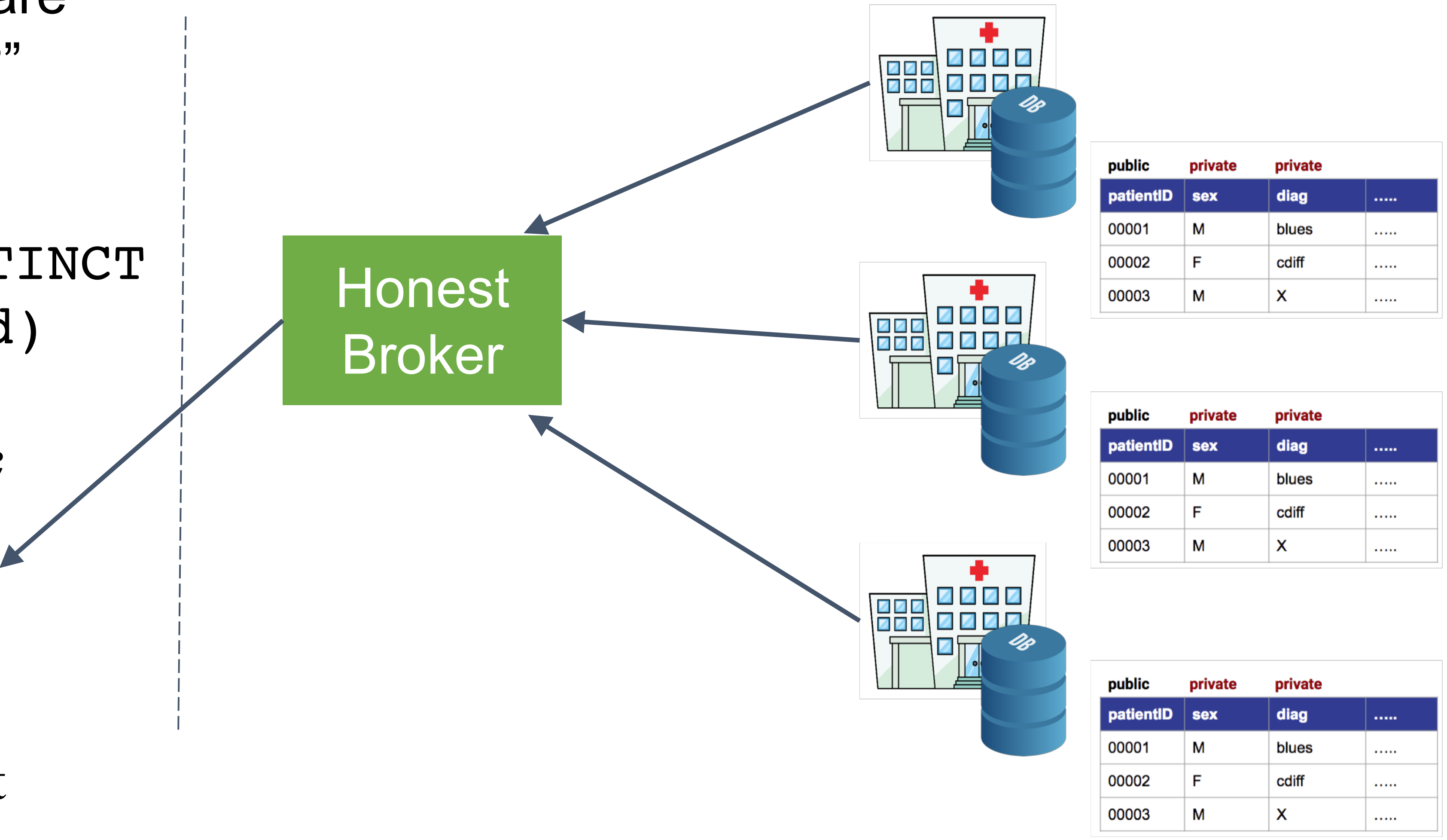
I have concerns about patient ID really being public, but let’s assume so for as in the paper

Clinical Data Research Network

“How many patients are there?”

```
SELECT
COUNT(DISTINCT
patient_id)
FROM
diagnosis;
```

Analyst



public	private	private	
patientID	sex	diag	.....
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

public	private	private	
patientID	sex	diag	.....
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

public	private	private	
patientID	sex	diag	.....
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

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;
```

Analyst

I can’t share that private data!

public	private	private	
patientID	**sex**	**diag**	**.....**
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

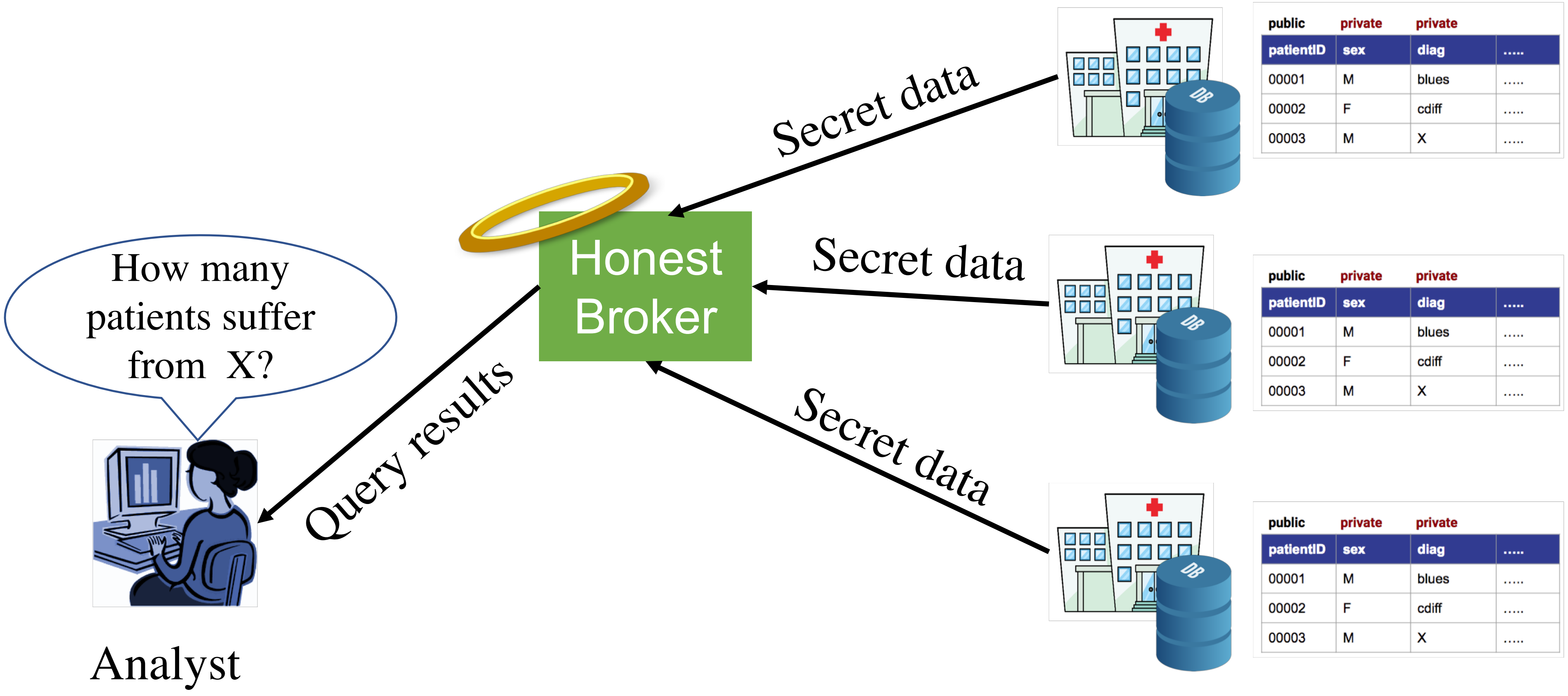
I can’t share that private data!

public	private	private	
patientID	**sex**	**diag**	**.....**
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

I can’t share that private data!

public	private	private	
patientID	**sex**	**diag**	**.....**
00001	M	blues	.....
00002	F	cdiff	.....
00003	M	X	.....

Goal: simulate a completely trustworthy third party to query private datastores



SMCQL

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

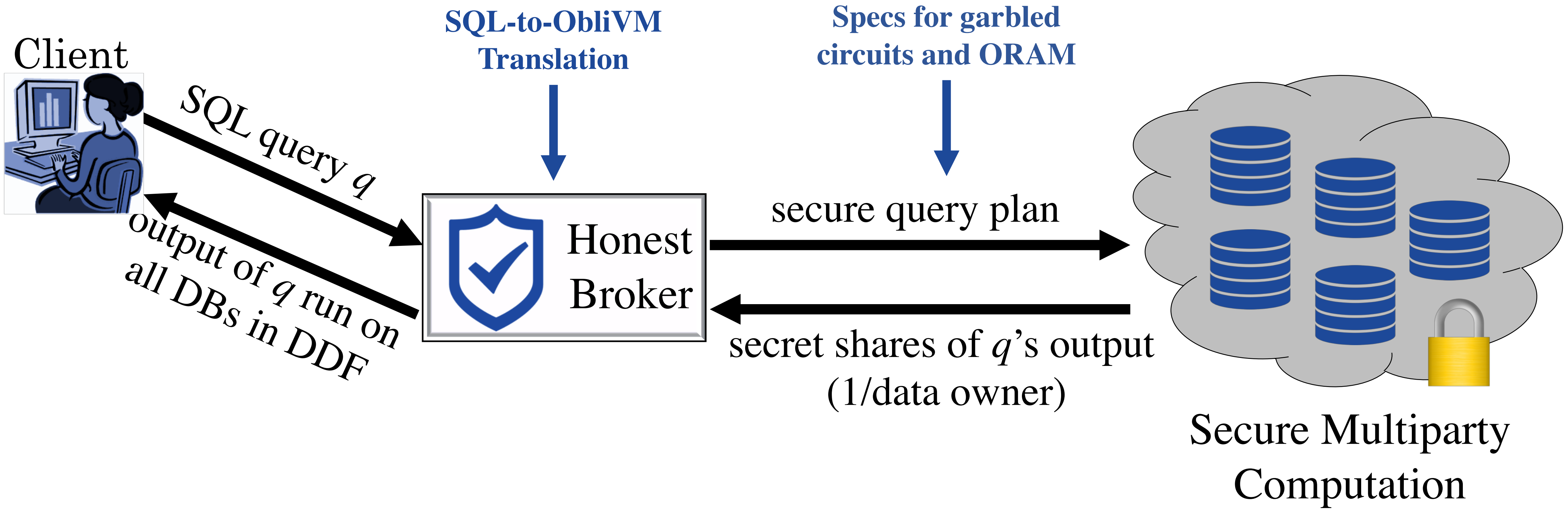
- Differential privacy: can be used complementarily to hide any one record in the final query result

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



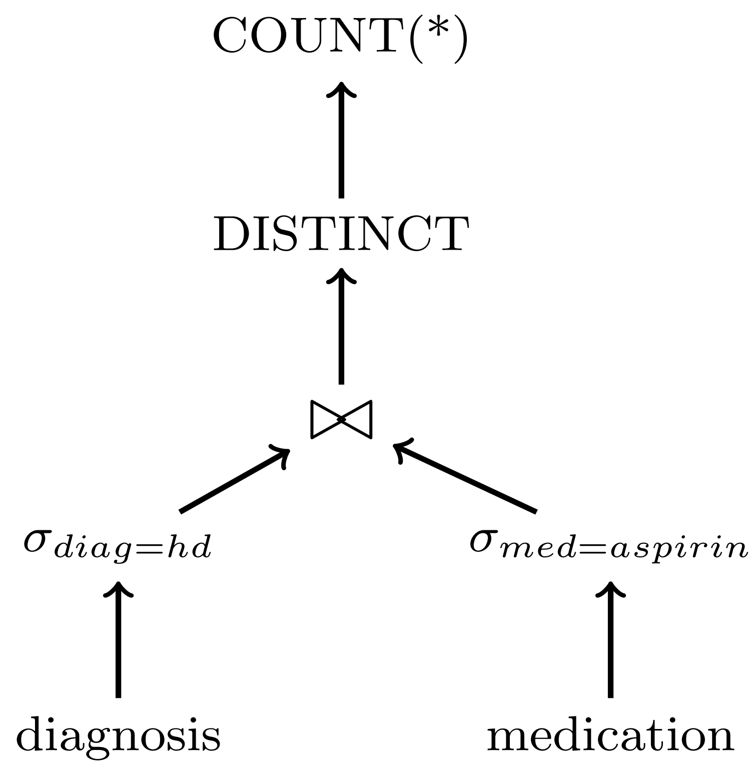
SMCQL is for two mutually distrustful data owners.

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



HealthLNK Queries

COMORBIDITY

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

ASPIRIN COUNT

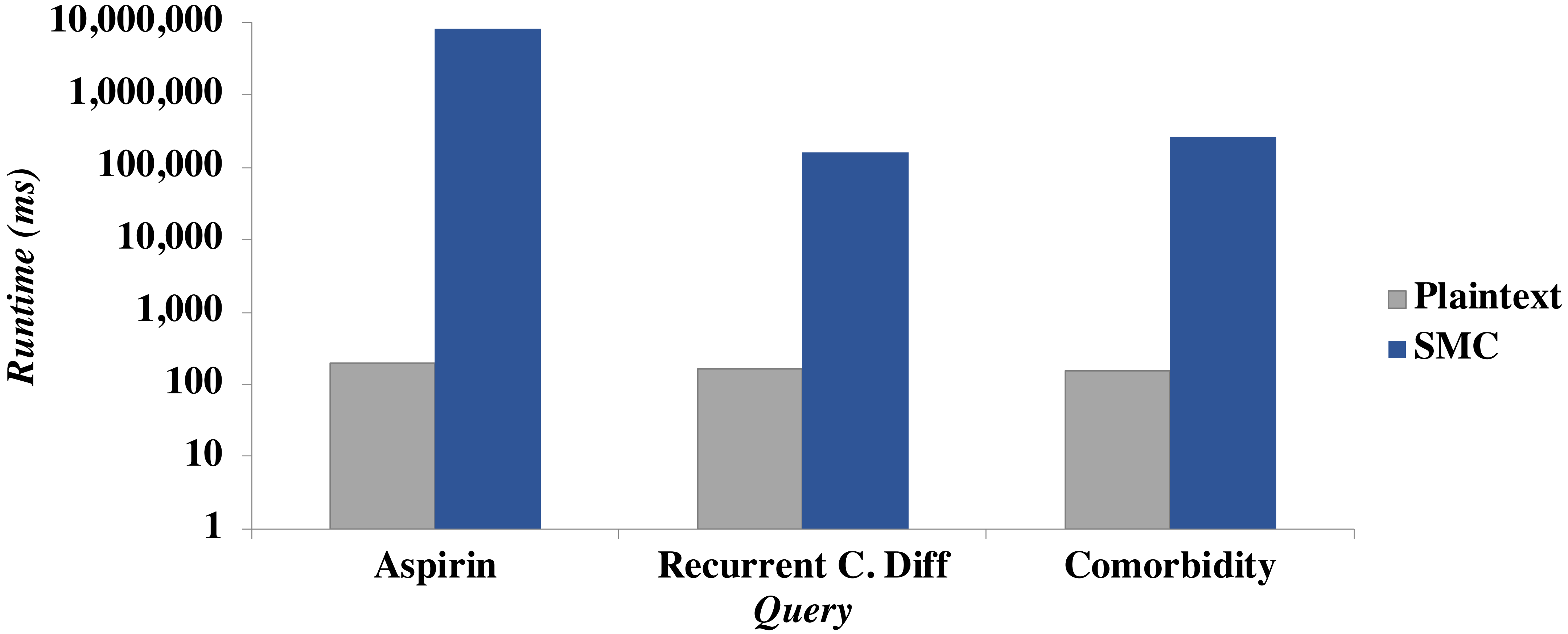
```
SELECT COUNT(DISTINCT pid)
FROM diagnosis d
   JOIN medication m ON d.pid = m.pid
WHERE d.diag = hd AND m.med = aspirin
   AND d.time <= m.time;
```

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;
```

SMC Performance

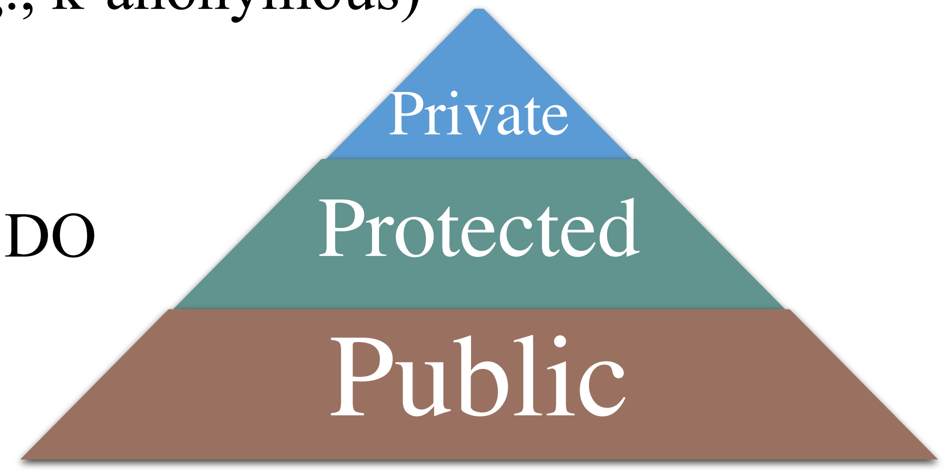


Secure multiparty computation is breathtakingly expensive even with small data.

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

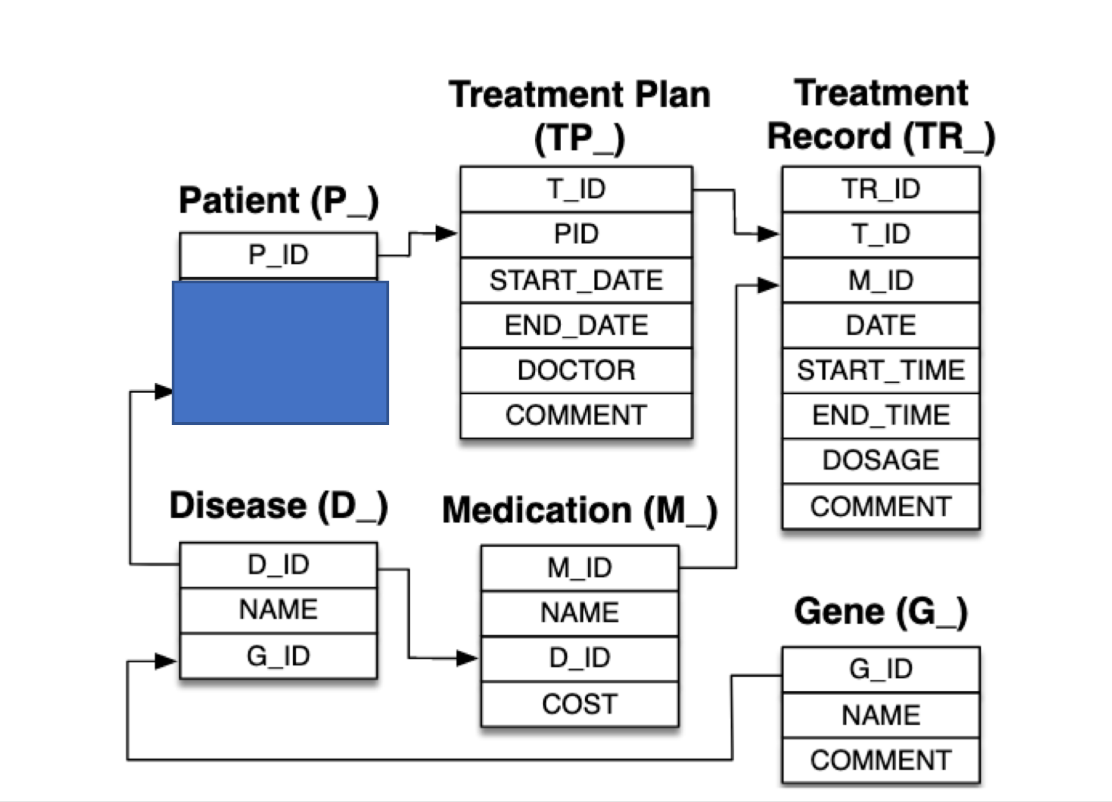
K-anonymity is an obsolete and weak privacy notion.
I think the protected attribute should not exist.



(whiteboard example of k-anonymity weakness)

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.

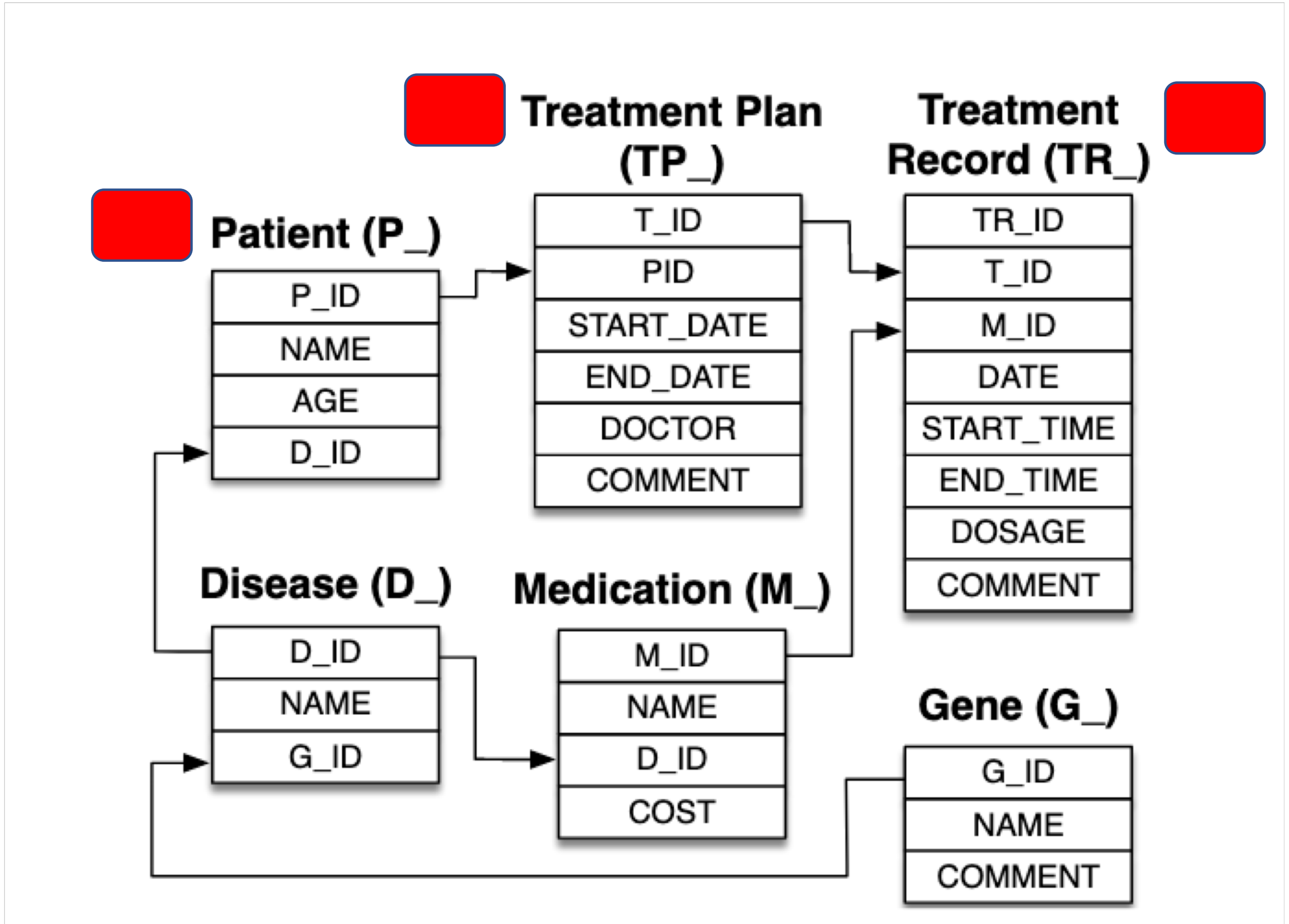
Second path analysis [Hinke'88]

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?



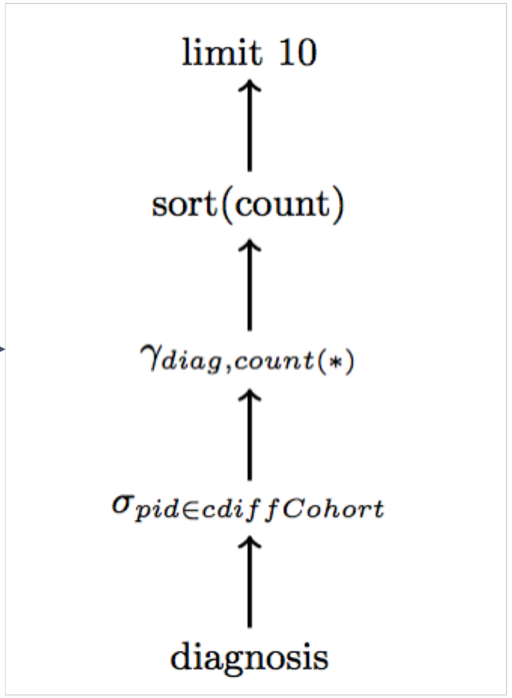
Patient, treatment plan and record are sensitive and should not be visible

Disease, medication and gene can be public, and contain not information about the patients

Operator Trees

COMORBIDITY

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



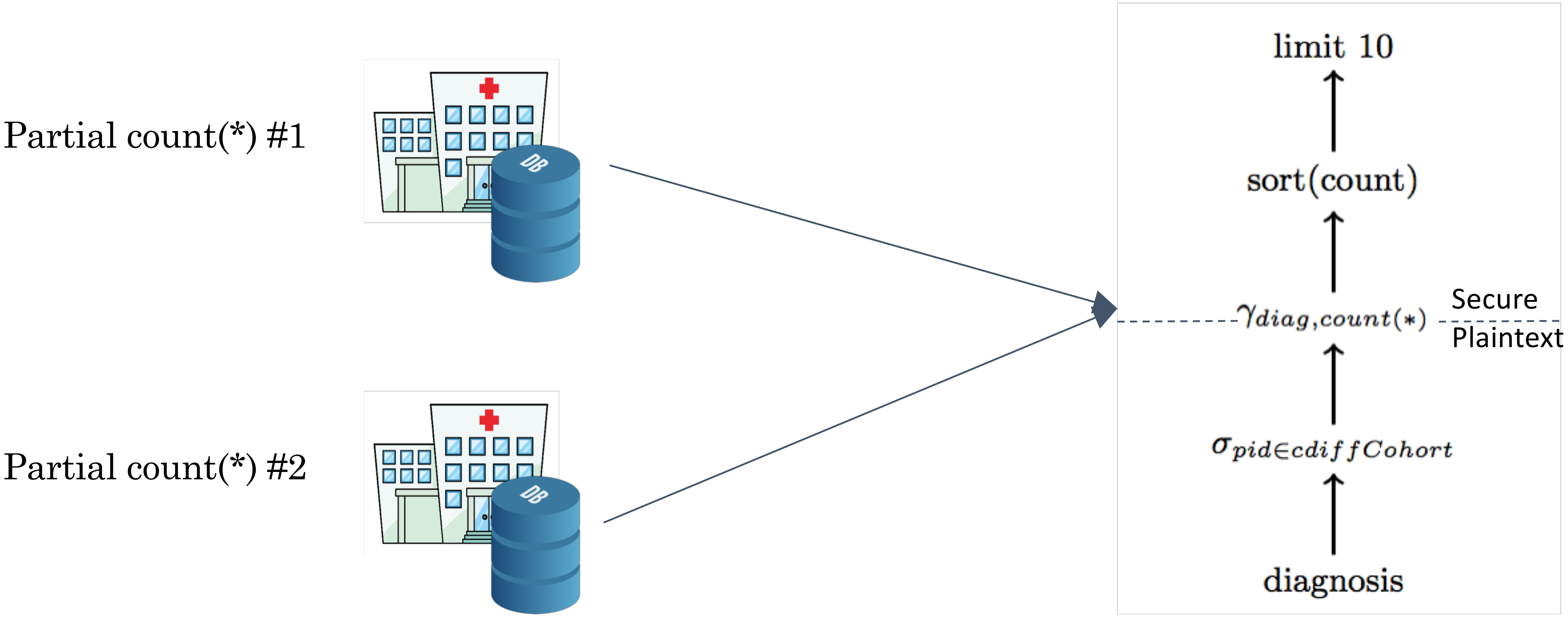
Query optimizations

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

- Need to rewrite query planners 

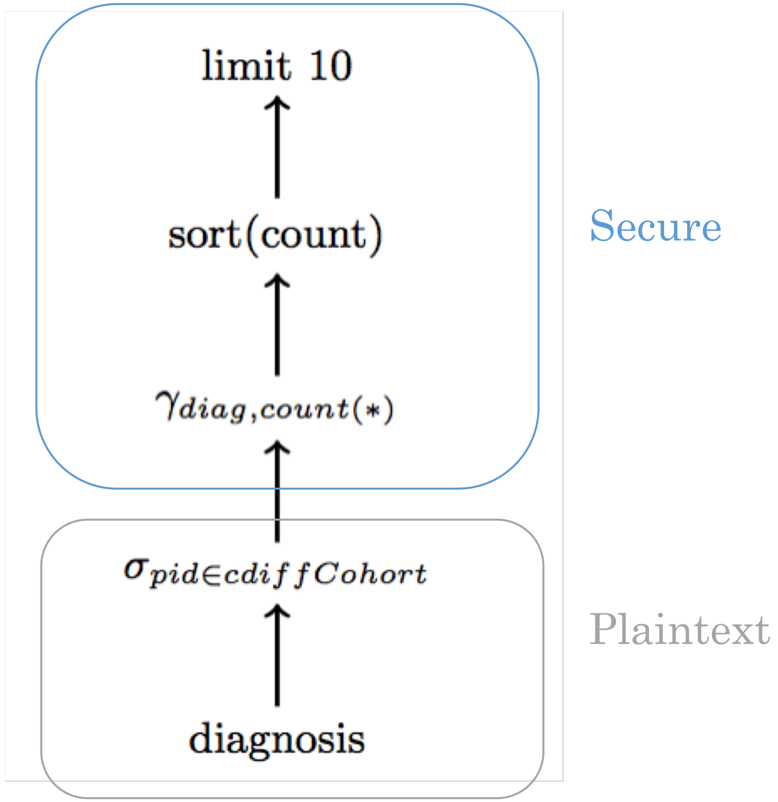
Query Optimization: Split Operators

Precompute part of the operator locally



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

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

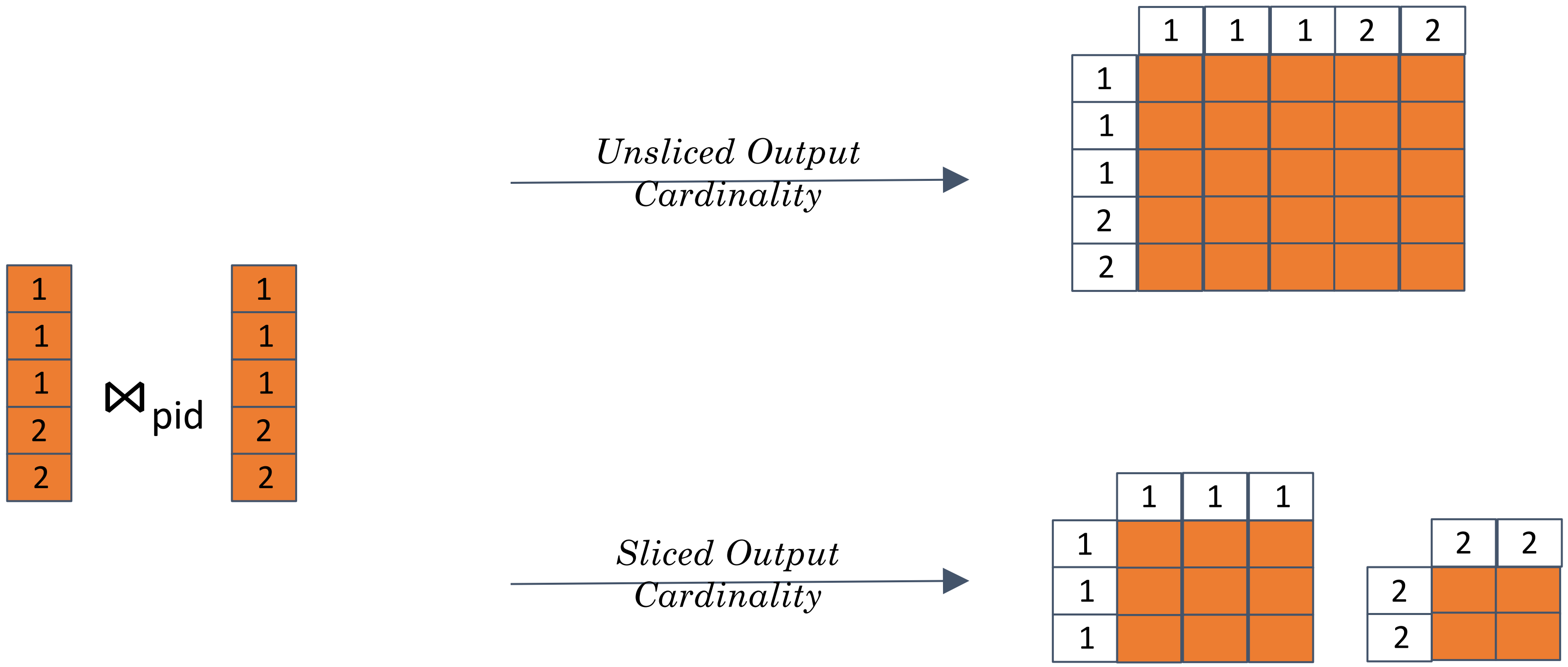
Local filter

Group by locally and compute local count

Pad intermediate values to public values to avoid leakage.

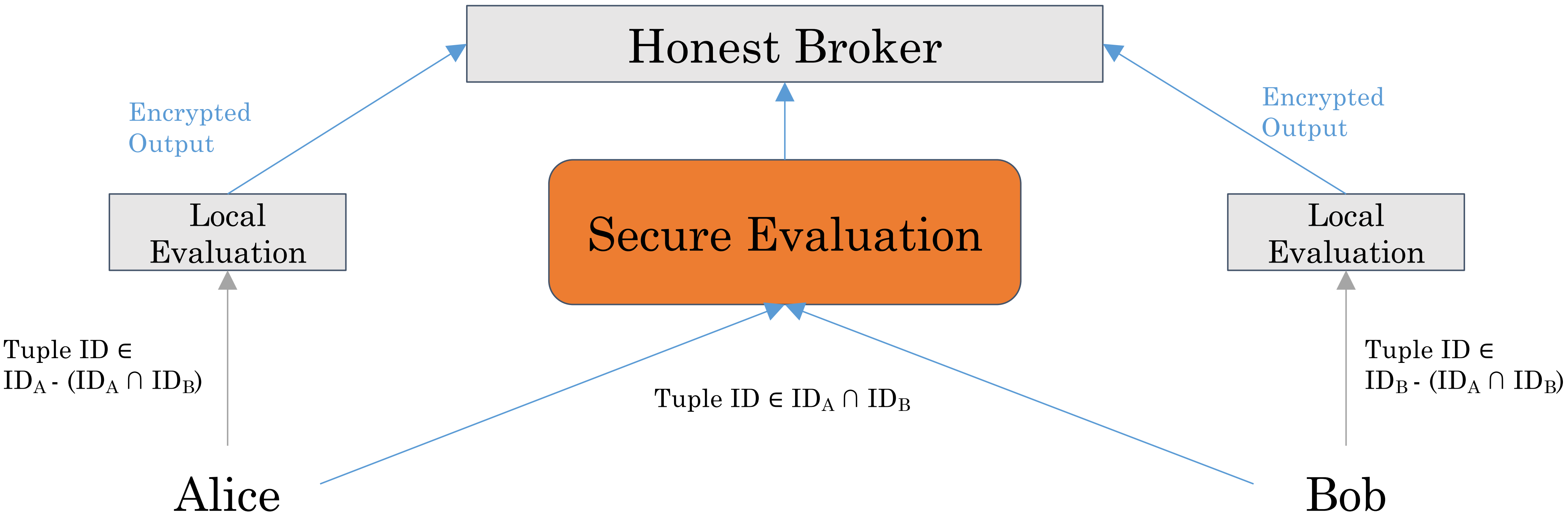
Query Optimization: Sliced Evaluation

Horizontally partition tuples on public attributes for secure evaluation



Query Optimization: Semi-join

Find single-party slices to eliminate unnecessary secure computation



Example

Assume table `diagnosis` at a party and `medication` at another party

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;
```

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;
```

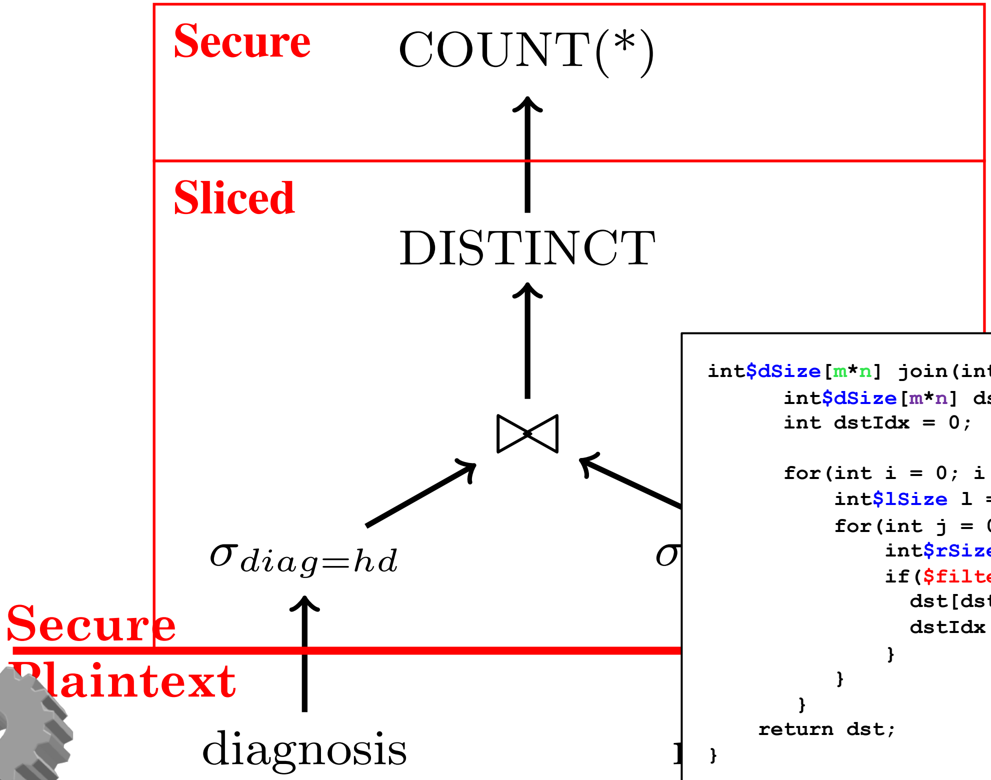
If pid is not sensitive, what is the split?

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

SMCQL Query Planner (at the honest broker)

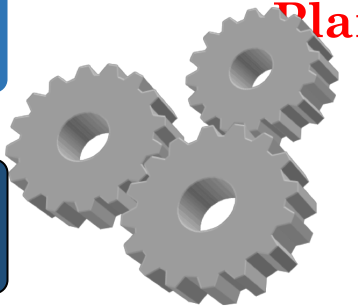
- SQL Statement
- Query Tree
- ID Secure Ops
- Optimize Secure Ops
- Generate ObliVM Code
- Executable Plan

```
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;
```

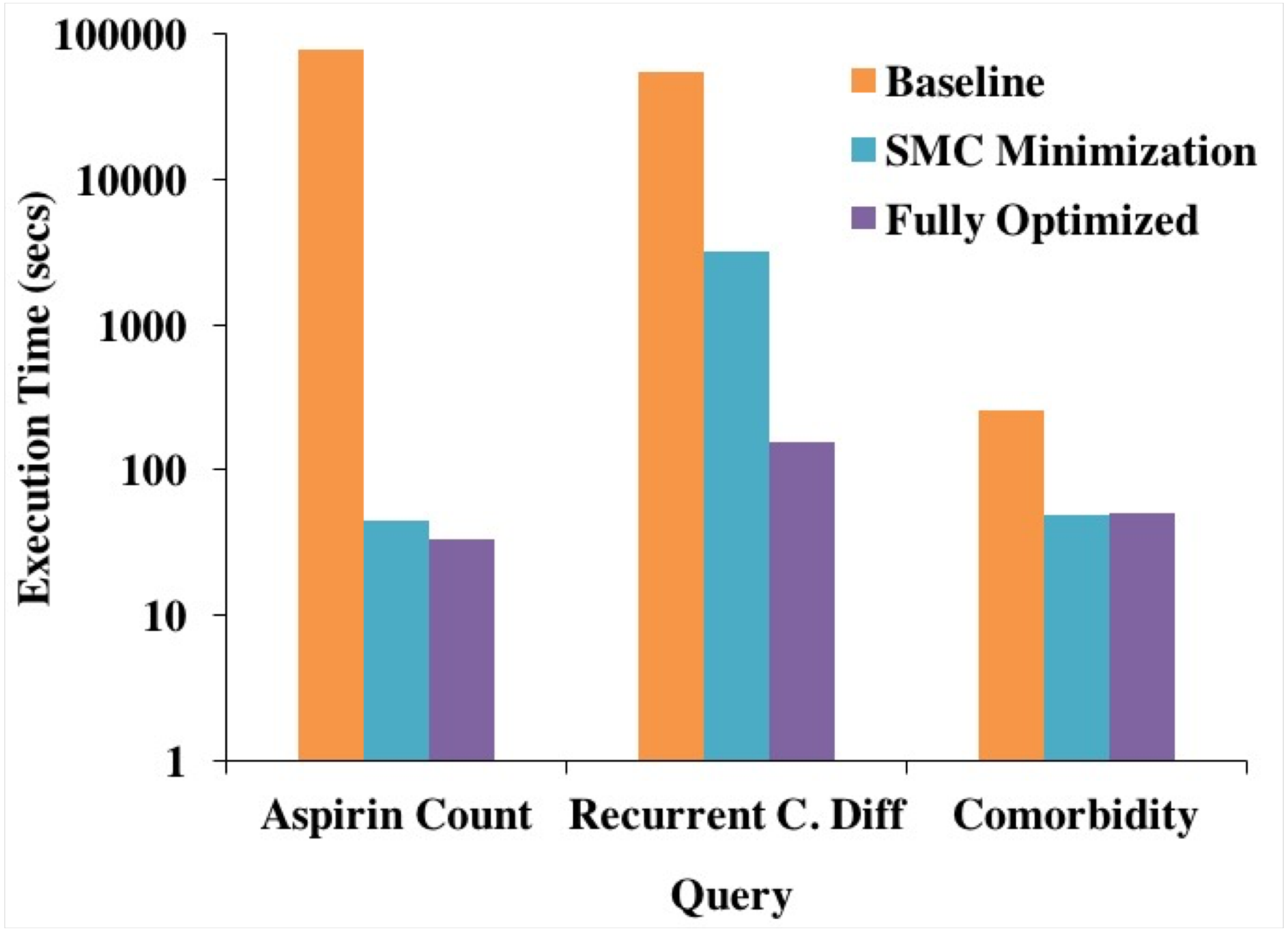



```
int$dSize[m*n] join(int$lSize[m] lhs, int$rSize[n] rhs) {
      int$dSize[m*n] dst;
      int dstIdx = 0;

      for(int i = 0; i < m; i=i+1) {
          int$lSize l = lhs[i];
          for(int j = 0; j < n; j=j+1) {
              int$rSize r = rhs[j];
              if($filter(l, r) == 1) {
                dst[dstIdx] = $project;
                dstIdx = dstIdx + 1;
              }
          }
       }
    return dst;
}
```



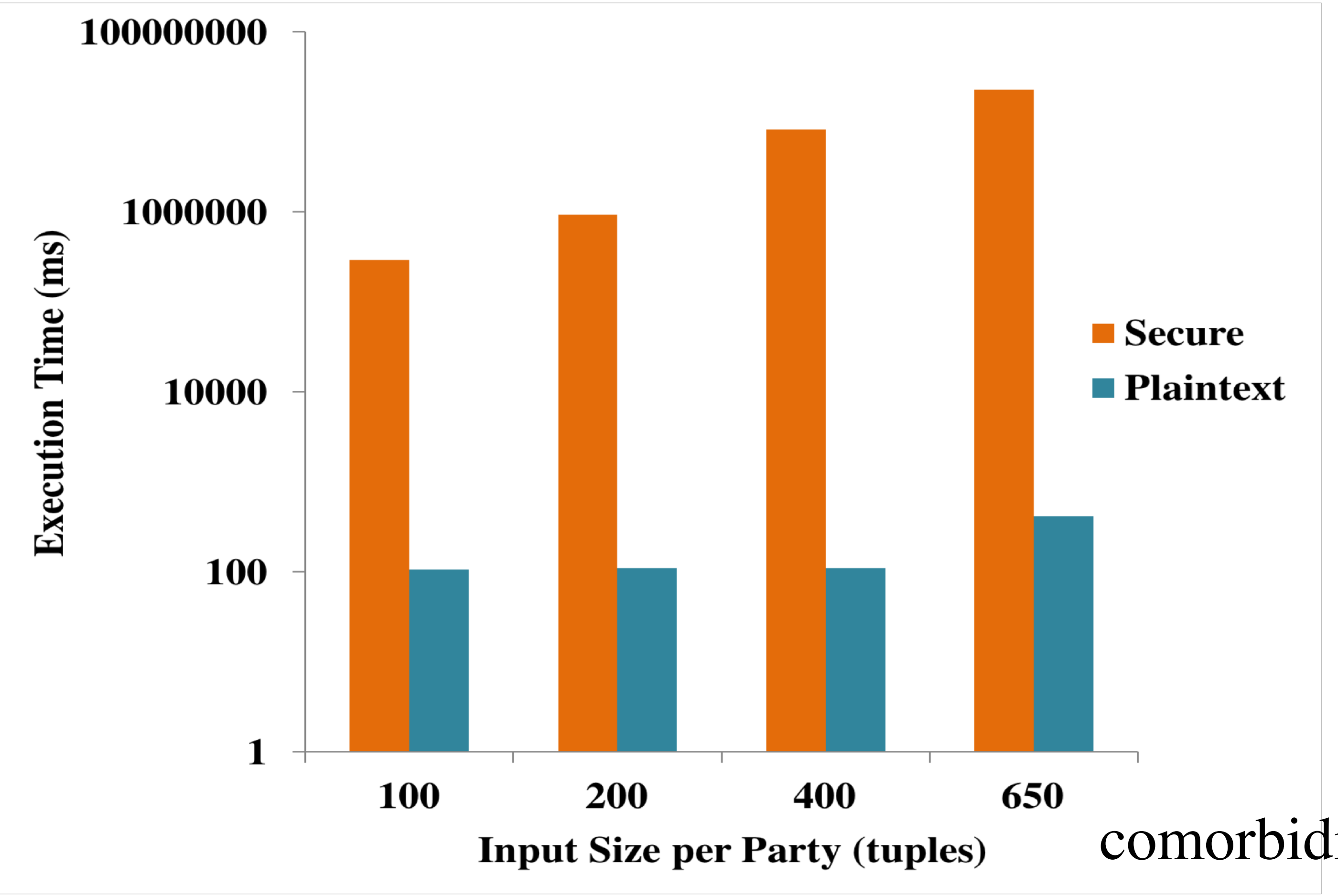
Performance on Sampled HealthLNK Data



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

Fully Optimized: using slicing often creates further speedup

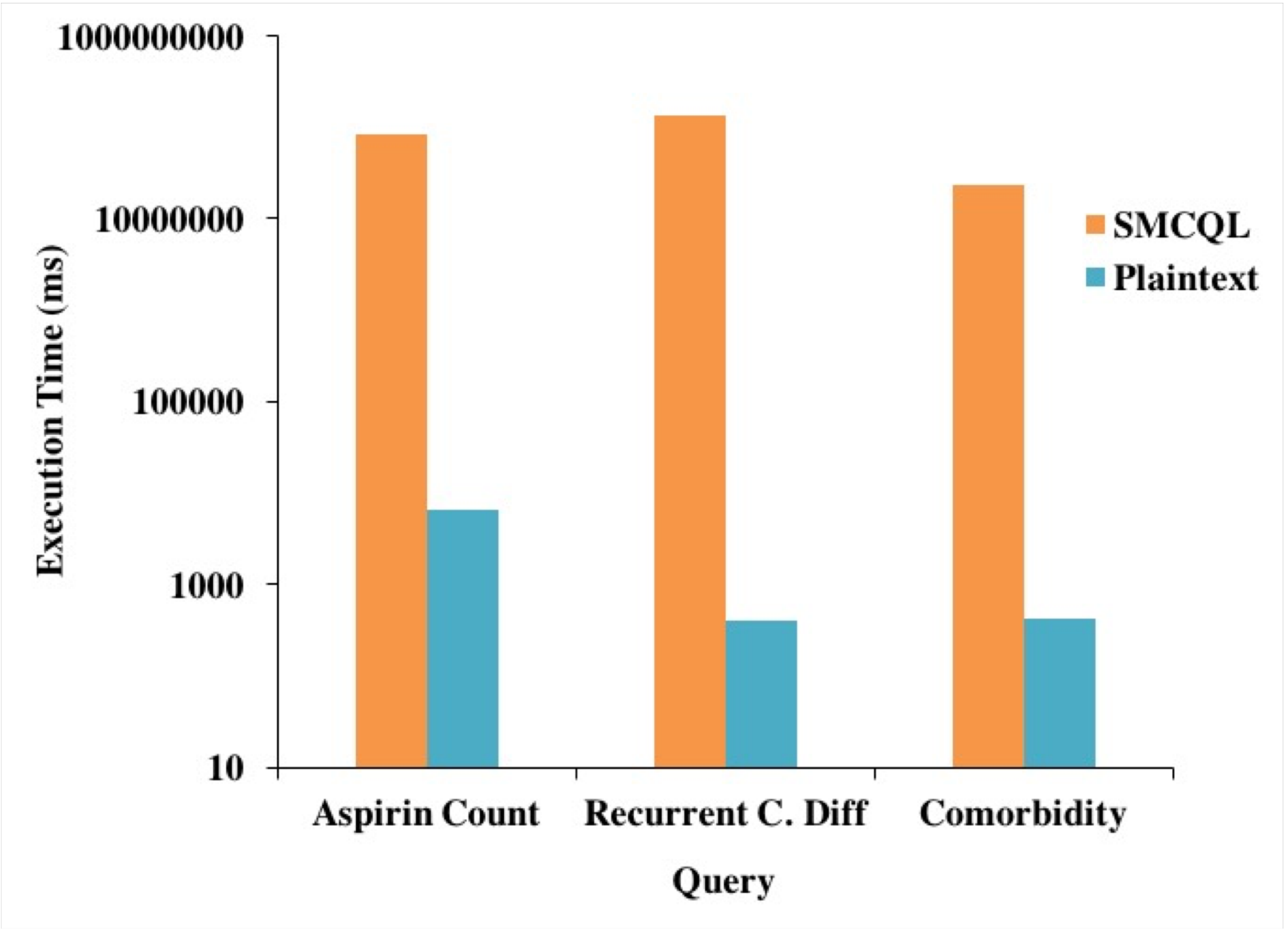
System Scale Up



comorbidity query

Minimizing the secure subtree enables us to scale to larger inputs.

SMCQL vs Plaintext



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