Conclave

Secure Multi-Party Computation on Big Data

Nikolaj Volgushev
Andrei Lapets
Malte Schwarzkopf
Mayank Varia
Ben Getchell
Azer Bestavros

SODA
Scalable Oblivious Data Analytics
Pre-presentation disclaimers/ confessions

- Less focus on developing new MPC protocols
- Instead focus on integrating MPC into big data analytics domain where the current roadblocks are:
  - Accessibility (data analysts are not MPC experts)
  - Scalability (large data volumes)
- Semi-honest adversaries throughout! 😈
How concentrated is the market for hired vehicles in NYC?

Is there healthy competition?

A dangerous monopoly forming?
A solution: Secure MPC

How concentrated is the market for hired vehicles in NYC?
DECLARE TABLE trips (start_lat int, start_lon int, ...);
SELECT SUM(mkt_share*mkt_share) AS hhi, SUM(trips.price) AS ... WHERE start_lat = ... AND ...;
Secure MPC

175M annual trips!
Does MPC scale? — Agg: SUM

N.B. log-scale!
Does MPC scale? — Join

![Graph showing runtime vs total input records for different secure and insecure methods. The graph compares Insecure (Spark), Secure (SM), and Secure (Obliv-C) methods. The secure methods show significant improvement in scalability compared to the insecure method.](image-url)
Our TODO list

• Make MPC more accessible to data analysts

• Make MPC scale for common analytics queries
Conclave

**Key insight:**
For most queries, only some of the work **must** happen under MPC.

**Automatically** rewrite query to combine scalable local computation and MPC.
Contributions

1. **MPC query compilation**: relational query compiler that optimizes for efficient MPC.

2. **Automated analysis** to minimize MPC use while maintaining required guarantees.

3. **Hybrid operators**: new MPC protocols that give the option to relax privacy requirements to further accelerate expensive operators under MPC.

4. **Prototype query compiler** implementation using Spark and Sharemind & performance evaluation.
DECLARE TABLE trips (start.lat int, start.lon int, ...);
SELECT SUM(mkt_share*mkt_share) AS hhi, SUM(trips.price) AS ...;
WHERE start.lat = ... AND ...;
Relational query specification

Parties’ data
treated as single
relation

# compute the Herfindahl-Hirschman Index (HHI)
rev = taxi_data.project(["companyID", "price"])
    .sum("local_rev", group=[“companyID"], over="price")
    .project([0, "local_rev"])
market_size = rev.sum("total_rev", over="local_rev")
share = rev.join(market_size, left=[“companyID"],
    right=[“companyID"
      ].divide("m_share", "local_rev", by="total_rev")

hhi = share.multiply(share, "ms_squared", "m_share")
    .sum("hhi", on="ms_squared")

hhi.writeToCSV()
Relational query specification

```
import conclave as cc
schema = [
    Column("companyID", cc.INTEGER), ...
    Column("price", cc.INTEGER)
]
# 3 parties each contribute inputs with the same schema
taxi_data = cc.defineTable(schema, at=[pA, pB, pC])

# compute the Herfindahl-Hirschman Index (HHI)
rev = taxi_data.project(["companyID", "price"])
    .sum("local_rev", group=[“companyID”], over="price")
    .project([0, "local_rev"])
market_size = rev.sum("total_rev", over="local_rev")
share = rev.join(market_size, left=[“companyID”],
    right=[“companyID”])
    .divide("m_share", "local_rev", by="total_rev")
hhi = share.multiply(share, "ms_squared", "m_share")
    .sum("hhi", on="ms_squared")
hhi.writeToCSV(to=[pA])
```
Contributions

1. **MPC query compilation**: relational query compiler that optimizes for efficient MPC.

2. **Automated analysis** to minimize MPC use while maintaining required guarantees.

3. **Hybrid operators**: new MPC protocols that give the option to relax privacy requirements to further accelerate expensive operators under MPC.

4. **Prototype query compiler** implementation using Spark and Sharemind & performance evaluation.
proj(concat(a, b)) = concat(proj(a), proj(b))
proj(concat(a, b)) = concat(proj(a), proj(b))
\[
\text{sum}(\text{concat}(a, b)) = \text{sum}(\text{concat}(\text{sum}(a), \text{sum}(b)))
\]
Contributions

1. **MPC query compilation**: relational query compiler that optimizes for efficient MPC.

2. **Automated analyses** to determine which parts of a query must run under MPC.

3. **Hybrid operators**: new MPC protocols that give the option to relax privacy requirements to further accelerate expensive operators under MPC.

4. **Prototype query compiler** implementation using Spark and Sharemind & performance evaluation.
| (ssn, zip) | (ssn, assets) | (ssn, assets) |
import conclave as cc
    cc.Party("mpc.b.cash")
demo_schema = [Column("ssn", cc.INTEGER), \
               Column("zip", cc.INTEGER)]
demographics = cc.defineTable(demo_schema, at=pA)
# credit card companies trust the regulator to compute on SSNs
bank_schema = [Column("ssn", cc.INTEGER, trust=[pA]), \
               Column("assets", cc.INTEGER)]
scores-1 = cc.defineTable(bank_schema, at=[pB])
...
Hybrid operators

• Two roles in hybrid operator scenario:
  - Semi-trusted party (STP) may learn a specific column in the clear; does not collude with other parties
  - Untrusted parties may not learn anything

• Goal:
  - Outsource expensive sub-steps to STP for local processing
  - Without leaking information to untrusted parties
<table>
<thead>
<tr>
<th></th>
<th>Complexity (Oblivious)</th>
<th>Complexity (Hybrid)</th>
<th>Bottle-neck operation (Oblivious)</th>
<th>Bottle-neck operation (Hybrid)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Join</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>$O(n^2)$ comparisons</td>
<td>$O(n+m \log (n+m))$ multiplications (where $m$ is size of result)</td>
<td>Pair-wise comparison between all rows</td>
<td>Batched oblivious array access</td>
</tr>
<tr>
<td><strong>Aggregation</strong></td>
<td>$O(n \log^2 n)$ comparisons</td>
<td>$O(n \log n)$ multiplications</td>
<td>Oblivious sort</td>
<td>Oblivious shuffle</td>
</tr>
</tbody>
</table>
Evaluation

1. How does Conclave scale to increasingly large inputs?

2. How much does automatic MPC frontier placement reduce query runtime?

3. What impact do hybrid operators have on query runtime?

• **Three parties**
  3 VM Spark cluster + Sharemind endpoint at each

• **Two queries**
  1. Taxi market concentration: up to 1.3B trip records
  2. Credit card regulation: up to 100k SSNs
Taxi market concentration query

Five orders of magnitude!
Credit card regulation query

Four orders of magnitude!
Related work

• Mixed-mode MPC: **Wysteria** [S&P 2014] — custom DSL

• Query rewriting for MPC

  • **SMCQL** [VLDB 2017]: binary public/private columns, no hybrid operators

  • **Opaque** [NSDI 2017]: computation under SGX, focus on reducing oblivious shuffles
Summary

- Conclave is a query compiler for efficient MPC on “big data”
- Automatically shrinks MPC step to be as small as possible
- New hybrid MPC-cleartext protocols speed up operators
- Scales up to 5 orders of magnitude better than pure MPC

https://github.com/multiparty/conclave
Conclave Implementation

• Relational front-end

• Rewrite rules on intermediate DAG of operators

• Back-ends generate code
  - Cleartext: Spark, sequential Python
  - MPC: Sharemind, Obliv-C (partial support)

• ~5,000 lines of Python
Hybrid MPC-cleartext operator impact

**Join**

- Sharemind only
- Conclave hybrid

**Aggregation**

- Sharemind only
- Conclave hybrid