Conclave
Secure Multi-Party Computation on Big Data

Nikolaj Volgushev
Andrei Lapets
Malte Schwarzkopf
Mayank Varia
Ben Getchell
Azer Bestavros
How about trips in Manhattan?
How about the morning rush hour?

How concentrated is the market airport transfers in hired vehicles in NYC?
We’re worried about information leaking.

Competitors might see our pricing.

We’re a private company — no obligation to report.
A solution: Secure MPC

How concentrated is the market airport transfers in hired vehicles in NYC?
A solution: Secure MPC

How concentrated is the market airport transfers in hired vehicles in NYC?

Secure MPC

➡ Read network messages.
➡ Inspect memory & storage contents.
➡ Observe control flow (IP + access patterns).
✔ Only local input and result are exposed.
Secure MPC

175M annual trips!
Does MPC scale?

- **Garbled circuits**: 1 wire = 1 bit = 1 label
  - Scales poorly in space: state >> input size

- **Secret sharing**: multiplication = network I/O
  - Vectorization & batching help, but only so much
  - Mostly small computations with few operations

- Lots of research on scaling to **many parties**, but little on scaling to **large input data**
Does MPC scale? — Projection

![Graph showing runtime versus input records per party. The graph compares insecure and secure (Sharemind) implementations. The runtime is plotted on a log-scale.](image)

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

![Graph showing the comparison of secure (Sharemind and Viff) and insecure (Spark) runtimes for different input sizes. The secure implementations show a much better scalability compared to the insecure one.](image)
Secure MPC
Conclave

Key insights:

For most queries, not all of the work **must** happen using cryptographic MPC techniques.

End-to-end guarantees can often be maintained even if part of the query is evaluated locally and in the clear by the parties.

We can automatically determine the MPC bounds.
Contributions

1. Conclave paradigm: run as little as possible, but as much as necessary under MPC

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

3. New “hybrid” MPC-cleartext protocols to 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 ...;

Per-party, mixed local-MPC query plan
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 ...;
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 ...;
Assumptions

• Existing local data-parallel infrastructure (e.g., Spark cluster)
• Schemas are public or common schema agreed
• Honest-but-curious adversary
• Honest majority or anytrust model
• Okay to leak intermediate relation sizes
  - MPC backend might impose additional meta-data leakage
  - Sensitive values never exposed to untrusted parties
# compute the Herfindahl–Hirschman Index (HHI)

```csharp
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()
```
import conclave as cc


# 3 parties each contribute inputs with the same schema
schema = [Column("companyID", cc.INTEGER), ...
          Column("price", cc.INTEGER)]

taxi_data = cc.concat([inputA, inputB, inputC])

# create multi-party input 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(owner=[pA])
PROJECT \rightarrow SUM \rightarrow PROJECT \rightarrow SUM \rightarrow PROJECT \rightarrow SUM

CONCATENATE

JOIN \rightarrow SUM

DIVIDE

MULTIPLY \rightarrow SUM

DIVIDE by 10k
import conclave as cc

  cc.Party("bankB")
demo_schema = [Column("ssn", cc.INTEGER), 
  Column("race", cc.INTEGER)]
demographics = cc.defineTable(demo_schema, owner=[pA])

# credit card companies trust the regulator to compute on SSNs
bank_schema = [Column("ssn", cc.INTEGER, trust=[pA]), 
  Column("score", cc.INTEGER)]
scores1 = cc.defineTable(bank_schema, owner=[pB])
...

Regulator has SSNs anyway
(ssn, race)

SHUFFLE

PROJECT ssn

ENUM

JOIN

PROJECT index

JOIN ON index

SHUFFLE

(ssn, score)

SHUFFLE

PROJ ssn

PROJ ssn

Large join outside MPC!
Implementation

- LINQ-style relational front-end

- Rewrite rules on intermediate DAG of operators

- Back-ends generate code
  - Cleartext: Spark, sequential Python
  - MPC: Sharemind

- ~5,000 lines of Python
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 MPC-cleartext 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

![Graph showing runtime versus number of input records.]

Six orders of magnitude!
Hybrid MPC-cleartext operator impact

**Join**

Runtime [sec] vs. Input records [log$_{10}$]

- Sharemind only
- Conclave hybrid

**Aggregation**

Runtime [sec] vs. Input records [log$_{10}$]

- Sharemind only
- Conclave hybrid
Credit card regulation query

Four orders of magnitude!
Related work

- 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”
• Computes as much as possible locally in the clear
• Automatically shrinks MPC step to be as small as possible
• New hybrid MPC-cleartext protocols speed up operators
• Scales up to 7 orders of magnitude better than pure MPC

https://github.com/multiparty/conclave

Ask me for our draft paper if you’re interested! :)

37