ZKSQL: Verifiable and Efficient Query Evaluation with Zero-Knowledge Proofs

Database Reading Group
Jan 17, 2025

Xiling Li

Specialized on secure, private and trustworthy data management

Northwestern | McCORMICK SCHOOL OF ENGINEERING

Outline

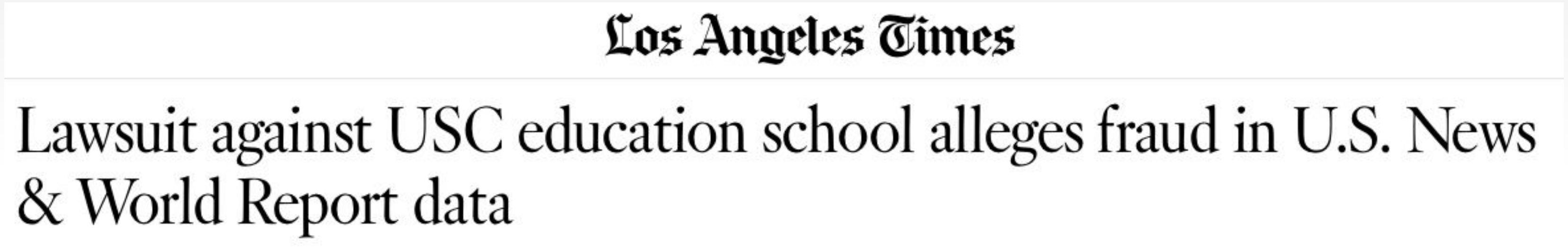
- Motivation
- Preliminaries
- System architecture
- Verifiable query evaluation
- Experimental results
- Related work

Verifiable Querying in Zero Knowledge

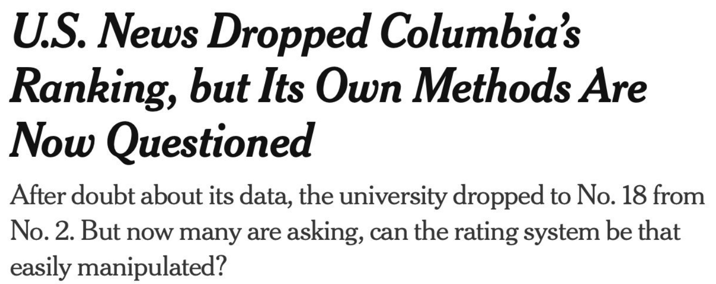
- Relational DBMSs are ubiquitous.
- Provide expressive, declarative queries with SQL.
- **Goal:** construct authenticated query answer without divulging private input data.

Running Example: College Rankings Data

- Students are increasingly using data to make decisions about when and where to attend college.
- Integrity of this data is paramount.
- We want to provide strong assurances that statistics from each school are accurate and complete.


Los Angeles Times

Lawsuit against USC education school alleges fraud in U.S. News & World Report data


U.S. News Dropped Columbia's Ranking, but Its Own Methods Are Now Questioned

After doubt about its data, the university dropped to No. 18 from No. 2. But now many are asking, can the rating system be that easily manipulated?

Forged data led to inaccurate rankings!

Data Privacy for Verifiable Queries

GDPR:

- General Data Protection Regulation
- May 25, 2018

CCPA

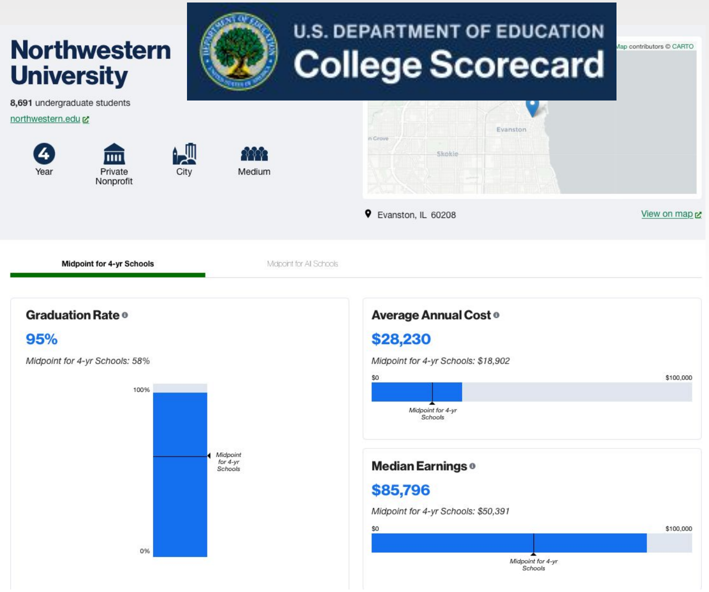
- California Consumer Privacy Act
- Jan 1, 2020

- Data owners are reluctant to accept public query over their private databases.
- Data owners don't want to reveal private information!
- Regulations on data privacy expose data owners to further liability when they answer queries.
  - Data leakage can create chilling effect.

Data breaches are also terrible!

Dilemma: Verifiable, Privacy-preserving Querying

Data Integrity ⚖ Data Privacy


U.S. DEPARTMENT OF EDUCATION
College Scorecard

Northwestern University

8,691 undergraduate students

northwestern.edu

4 Year | Private Nonprofit | City | Medium

Evanston, IL 60208 View on map

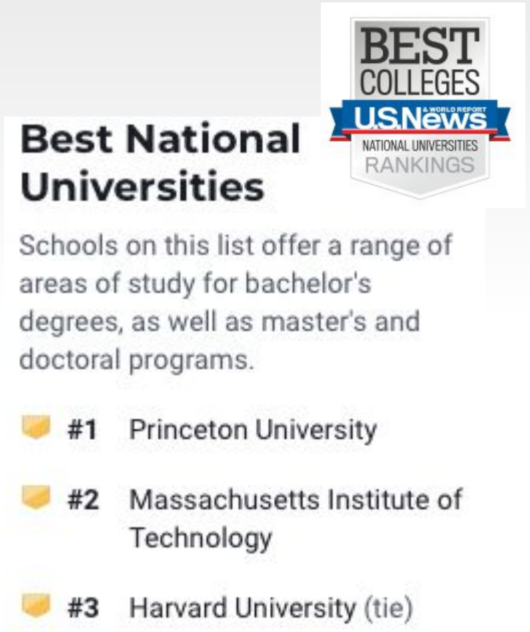
Midpoint for 4-yr Schools | Midpoint for All Schools

Graduation Rate
95%
Midpoint for 4-yr Schools: 58%

Average Annual Cost
$28,230
Midpoint for 4-yr Schools: $18,902

Median Earnings
$85,796
Midpoint for 4-yr Schools: $50,391

How can users obtain trustworthy statistics without data providers sharing their private records?


BEST COLLEGES
U.S.News & WORLD REPORT
NATIONAL UNIVERSITIES RANKINGS

Best National Universities

Schools on this list offer a range of areas of study for bachelor's degrees, as well as master's and doctoral programs.

#1 Princeton University

#2 Massachusetts Institute of Technology

#3 Harvard University (tie)

Our Solution - ZKSQL

- Contributions:
  - First work (until the paper published) for ad-hoc SQL queries with Zero-Knowledge Proofs.
  - Operator-at-a-time proofs for a given query plan.
  - Set-based protocols for proving properties about intermediate tables.
  - Experimental results on TPC-H benchmark demonstrate improvement with ZKSQL system.
    - Up to two orders of magnitude over the pure-circuit baseline.

What is a Zero Knowledge Proof?

- In a ZKP, a prover (*P*) convinces a verifier (*V*) that a statement is true without revealing its private information.
  - In running example, an university is the prover.
  - The DoE or U.S. News is the verifier.

- ZKP guarantees correctness, soundness, and zero knowledge.

ZKP in Query Evaluation

- Problem statement:
  - *P* has a private database *D*.
  - *P* and *V* wish to evaluate query *Q* to prove authenticated query answer *A*.
- ZKSQL uses state-of-the-art interactive ZK protocols with commit-and-prove paradigm.
  - Scalable to large data size due to the flexible proof size.
  - Composable for complex query logic, where future operator-level protocols can be added directly.
- ZKSQL constructs proofs for ad-hoc queries that guarantees:
  - A malicious *P* cannot sabotage integrity of query evaluation (*correctness* and *soundness* properties).
  - A malicious *V* cannot obtain any unauthorized information about *D* (*zero-knowledge* property).

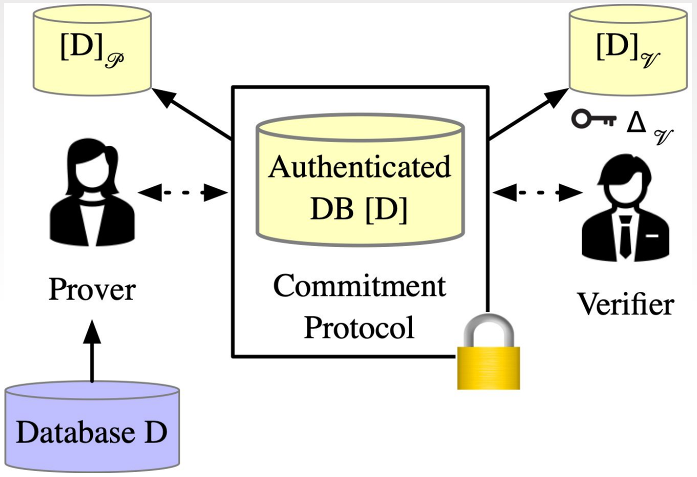
Obliviousness

- To remain zero-knowledge, proof evaluation is ***oblivious*** w.r.t. operator control flow.
  - Operation on each tuple should be same even if it is not necessary.
  - Prevents data access pattern leakage.

- Rather than removing a tuple, we keep each tuple in [D] with a *dummy* tag (0 or 1).
  - Dummy tuples do not contribute to query answer.

- V knows:
  - Schema of relations including column names, column types and constraints.
  - (Maximum possible) cardinalities of intermediate tables.
  - Operator execution order.
  - Query answer A.

Private Database Commitment



No one can deduce commitment of the other party!

- *P* commits private database *D*.

  $$D \rightarrow [D]$$

- Each Party holds its partition respectively.

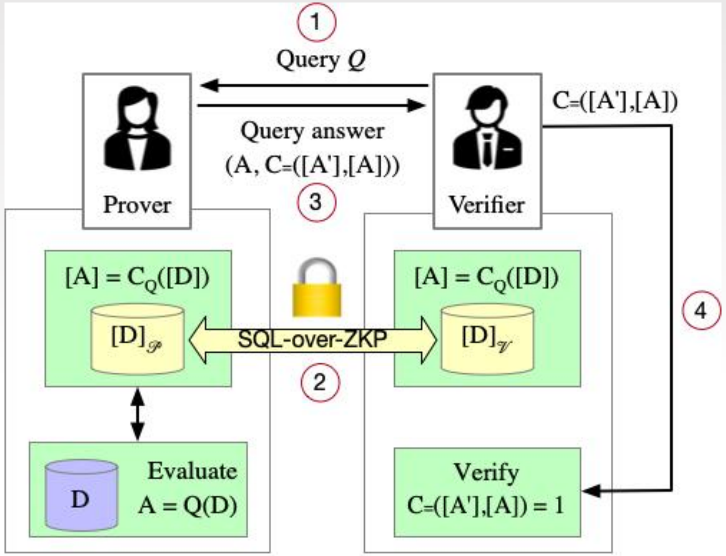
  $$[D]_P \text{ or } [D]_V.$$

- *V* alone additionally holds an authentication key, Δ_V.

  $$[D]_P = [D]_V + D * \Delta_V$$

- *P* only commits *D* once for all queries related to the same DB.

Authenticated Query Evaluation



- *V* sends query *Q* to *P*.

  P and *V* interactively evaluate the query in ZKSQL.

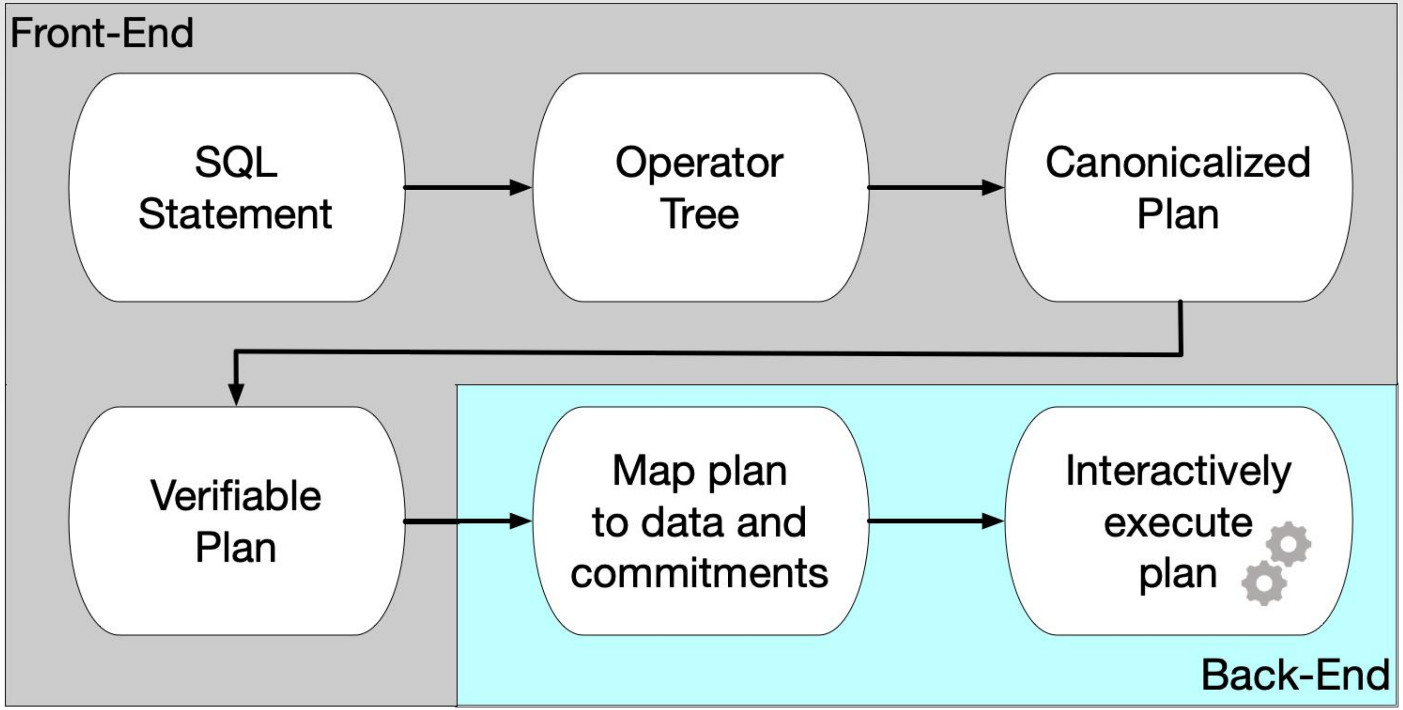
- During interactive proofs:
  - *P* has dual representation (plaintext and commitments) of query data including intermediate tables.
  - *V* works on only its commitments $[.]_V$.

- Finally *P* sends authenticated answer *A* and proof to *V*.

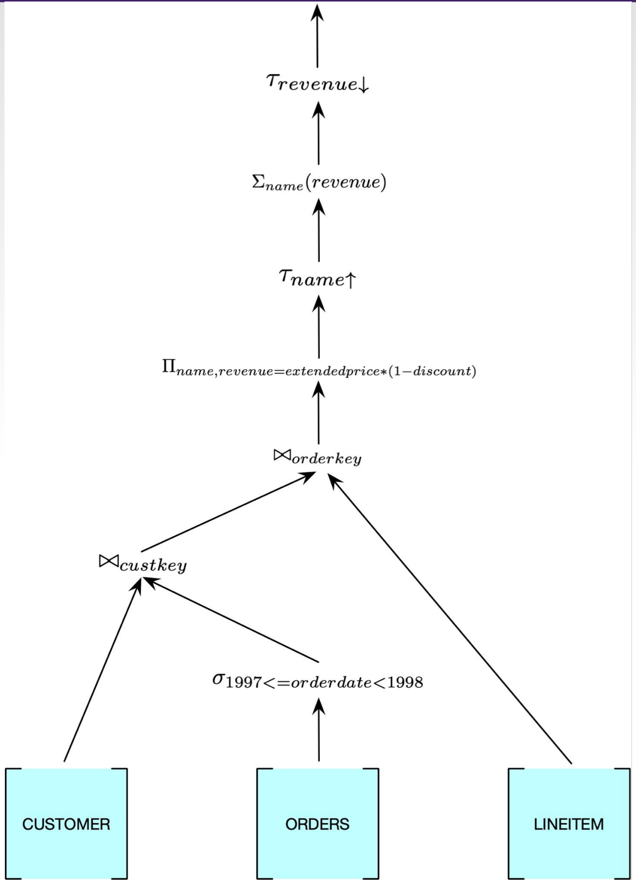
- *V* is able to verify the answer with the proof.
  - Accepts if the proof returns 1.
  - Otherwise, reject the answer.

Northwestern | ENGINEERING

ZKSQL Workflow and Roadmap



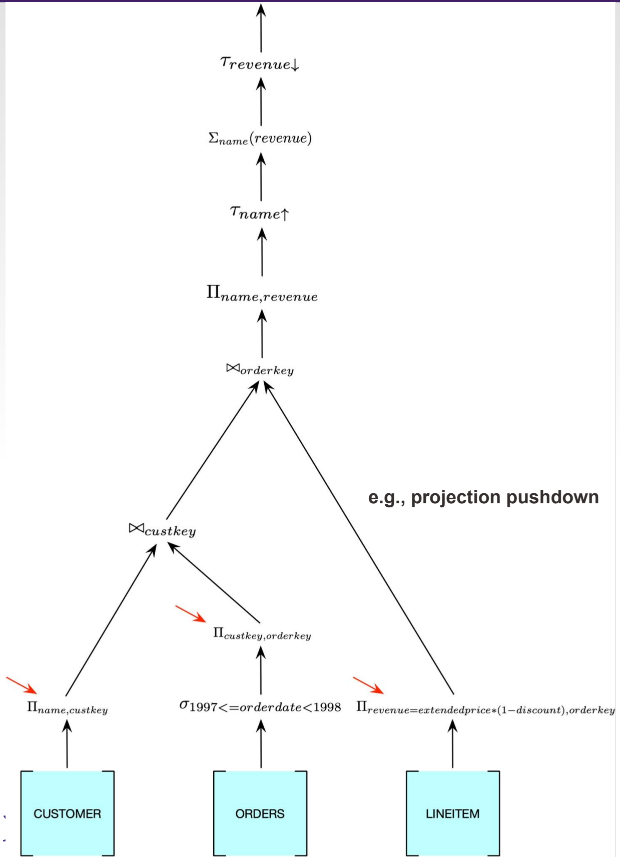
Front-End: Query Planning



- ZKSQL parses Q into a directed acyclic graph of operators
- Parsing in Apache Calcite*.
- We support:
  - Filter (σ)
  - Project (Π)
  - Join (⋈)
  - Aggregate (Group-by and Scalar)
  - Sort (τ)
- ZKSQL has a protocol template for one or more algorithms for each of these operators.

https://calcite.apache.org

Front-End: Canonicalized Plan



- ZKSQL uses Volcano-style transformations to convert the DAG into an efficient execution plan.
- Rules:
  - Projection pushdown
  - Filter merge
  - Sort for group-by aggregate
- Revised plan has more efficient execution over ZK circuits.

Verifiable Plan

- ZKSQL maps each DAG node to an operator template.
- Maps the operator with its parameters (predicates, sort direction, etc.) into operator instance.
- Compose these to form a sequential verifiable plan.
- For a leaf operator scanning table *T*, we instantiate this as a scan over the commitments [*T*].
- *P* and *V* interactively execute the verifiable plan one operator-at-a-time in the back end.

Baseline Results

Query	***Q1***	***Q3***	***Q5***	***Q8***	***Q9***	***Q18***
Plaintext	0.05s	0.02s	0.03s	0.01s	0.13s	0.05s
Circuit-Only	777s	24,130s	38,106s	32,040s	43,562s	126,168s
Slowdown	15,540X	1,206,500X	1,270,200X	3,204,000x	335,092X	2,523,360X

- Protocol of each operator can directly translate to circuits based on its standard DBMS logic.
- Circuit-only approach shows 5~6 orders of magnitude slowdown over unauthenticated query evaluation.
- Instead, ZKSQL improves performance by delegating some steps to *P*.
  - *P* and *V* interactively verify P's local computation using set operations.
  - Up to 2 OOM improvement.

ZK Set Operations

- Set operations verify intermediate tables locally computed by *P*.
  - Improve oblivious pure circuits approach.

- We support four set operations in the ZK protocols:
  - Equality
  - Disjoint
  - Intersection
  - Union

- Each set operation has complexity *O(n)*.

ZKSQL protocols

- ZKSQL instantiates two kinds of ZK protocols for operators.
- A circuit-only protocol evaluates operators in pure circuits over commitments (as baseline).
- A set-based protocol is a mix of circuits and ZK set operations.
- *T* output from each operator either an intermediate table or the authenticated answer *A*

Circuit-only protocols

- Project, Filter, and Aggregate operators run in pure ZK circuits.
- Follow logic similar to standard DBMS version.
- *P* and *V* interactively prove a tuple at a time from input *R*.
- No direct optimization benefits from set operations.

Projection

- For each e_j in the predicates ε, e_j can be:
  - a column mapping: e.g. e_0 = \$2
  - an expression on columns: e.g. e_1 = \$1+\$3.

- For each tuple r_i in R, P and V sends the commitment $[r_i]$ to the circuit C_ε and C_ε returns $[t_i]$ to each party.
  - If e_j is a column in R, ZKSQL simply copies $[r_{i,e_j}]$ to $[t_{i,j}]$ and it needs minor overhead.
  - Otherwise the protocol invokes circuit C_{e_j}, and outputs $[t_{i,j}]$: e.g. C_{e_j} evaluates $[t_{i,1}]$ with $[r_{i,1}]$ + $[r_{i,3}]$.
  - V should abort $[t_{i,j}]$ if the proof returns 0 (soundness).

Filter

- Filter on predicate *p*.

- For each tuple r_i in *R*, *P* and *V* send the commitment [r_i] to the circuit C_p; C_p returns [t_i] to each party.

- Each party also verifies predicates satisfaction.
    - C_p marks [t_i] is dummy if [r_i] satisfies *p*, but [r_i] is dummy.
    - The dummy [t_i] will not participate in the latter operators, but helps with keep obliviousness.

- If we have a specific limitation m, the protocol truncates *T* into first *m* tuples, after oblivious sort.

Aggregation

- Input*: Agg in* (*SUM*, *AVG*, *MIN*, *MAX*, COUNT).
- Obliviously sort *R* by group-by bins, *G*.
- For each tuple r_i in *R*, *P* and *V* send the commitment [r_i] to the circuit C_{Agg} and C_{Agg} replies to each with [t_i].
  - If [$r_{i,dummy}$] = 1, then C_{Agg} won't involve [r_i] into aggregated results.
- Mark [t_i] to *dummy* if [r_{i+1}] and [r_i] in same group-by bin.
  - Again, we don't explicitly delete tuples.
  - Instead, we make them as placeholders to keep obliviousness.
- Each group-bin has exactly one non-dummy tuple in *T*.

Set-based Protocols

- Baseline with *n* input tuples:
  - Oblivious (bitonic) sort takes $O(n \log^2 n)$.
  - Oblivious (nested-loop) join takes $O(n^2)$.
- We optimize protocols up to $O(n)$ with ZK set operations.
  - Implements equi-join and sort.
  - Protocols consist of both circuits and set operations.
- Verification of *P*'s result:
  - Verify set relationships of input/output tables: e.g. set equality.
  - Prove properties of tables: e.g. monotonically ordering.

Sort

- *P* locally sorts *R* to *T* w.r.t. sort definition and commits *T* into [*T*].
  - Sort algorithm could be any efficient one picked by P itself.
- Sort definition is a set of sorting order (either *increasing* or *decreasing*) for some attributes of *R*.
- **Order verification (correctness)**:
  - Check each adjacent tuples in [T] are monotonically ordered w.r.t sort definition.
- **Set equality verification (soundness)**:
  - Check if [*R*] and [*T*] have exactly the same tuples (order-agnostic).
- Zero knowledge property is still guaranteed by obliviousness.
  - Bitonic sort will do dummy swap even if the two tuples do not have to swap.

Equi-Join

- *P* locally joins *R* and *S* w.r.t. join predicates *p* and commits *T* into [*T*].
  - Again, P can use any efficient join algorithm according to the cost model.
- **Predicate verification (correctness)**: All output rows satisfy *p* by linearly scanning [T].
- **Set difference verification (soundness)**
  - Both parties generate [*U*] and [*V*], and *P* commits [Δ_R] and [Δ_S].
  - ZKSQL checks if [R] = [Δ_R] || [U] and [*S*] = [Δ_S] || [V].

No spurious tuples added by *P*!

- **Disjoint verification (completeness)**
  - Both parties generate [K_R] and [K_S] from [Δ_R] and [Δ_S].
  - ZKSQL checks if [K_R] is disjoint to [K_S].

T has all output rows!

- Zero knowledge property is guaranteed by padding the table T up to maximum possible cardinality by P.
  - If primary-key-foreign-key information is known, |[T]| = |Foreign-key table|.
  - Otherwise, |[T]| = |R|*|S|.

Experimental Setup

- We implement ZKSQL on top of EMP Toolkit* and our implementation is open-sourced#.
- We evaluate ZKSQL over TPC-H benchmark powered by PostgreSQL for plaintext execution.
- We choose queries varying degrees of complexity: *Q1*,*Q3*,*Q5*,*Q8*,*Q9*,*Q18*.
- We scale TPC-H database into 3 sizes: *60k Rows*, *120k Rows* and *240k Rows*.
- We convert all float fields into 64-bit integers although ZKSQL supports this type.
- We setup two AWS *EC2 r6i.4xlarge* instances, one for each party.
- **Metrics:** runtime, memory usage and communication cost between parties.

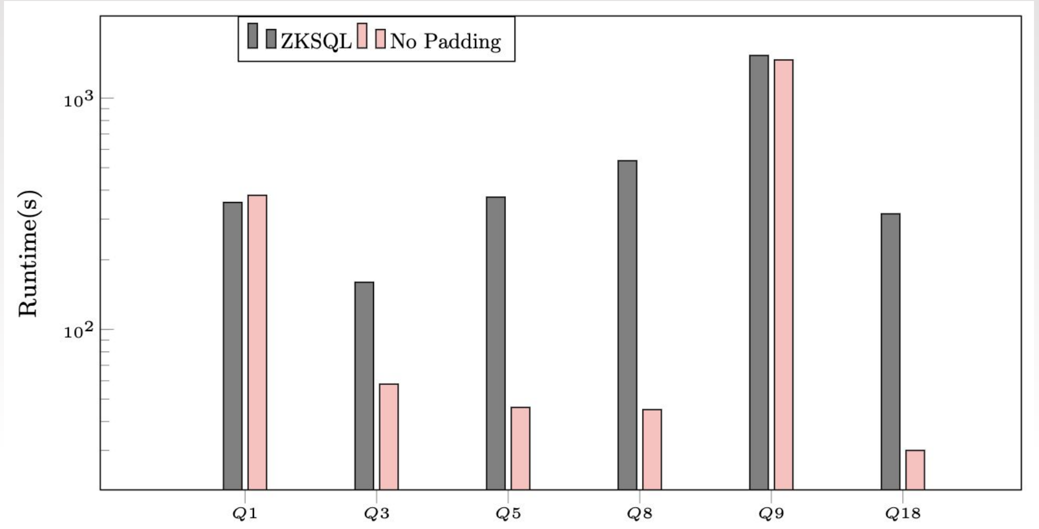
* *https://github.com/emp-toolkit*
# *https://github.com/vaultdb/zksql*

Commit Protocol Cost (TPC-H)

	Part	Partsupp	Lineitem	Orders	Supplier	Region	Nation	Customer
Runtime (s)	0.59	2.01	11.40	2.51	0.36	0.33	0.33	0.58
Memory (MB)	764	960	1,916	997	723	719	720	764
Comm. Cost (MB)	3.75	4.31	8.06	4.31	3.75	3.75	3.75	3.75

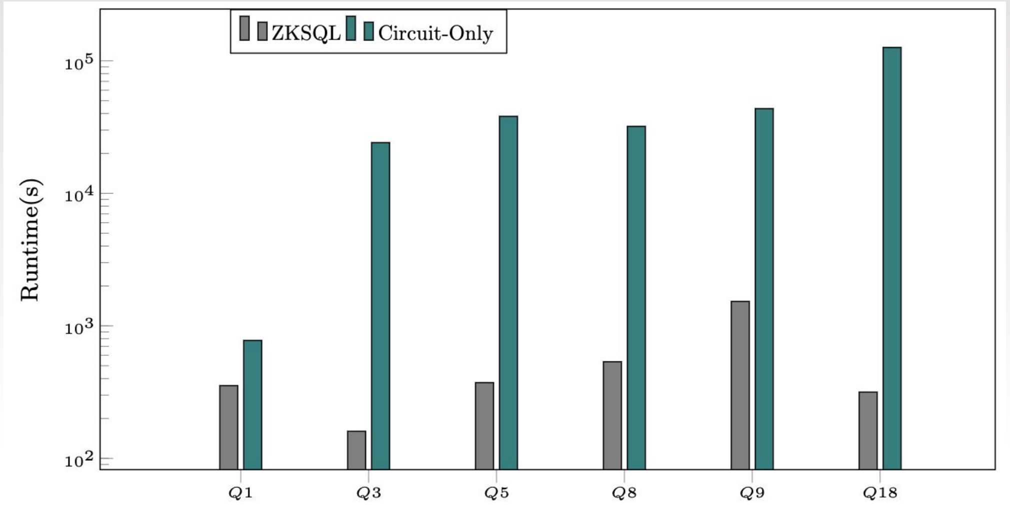
- The commitment for each table is an one-time setup for all queries.
- Compared to costs of experiments over *60k Rows*, the cost of commitment is minor part of our overall cost.

Oblivious Querying Overhead



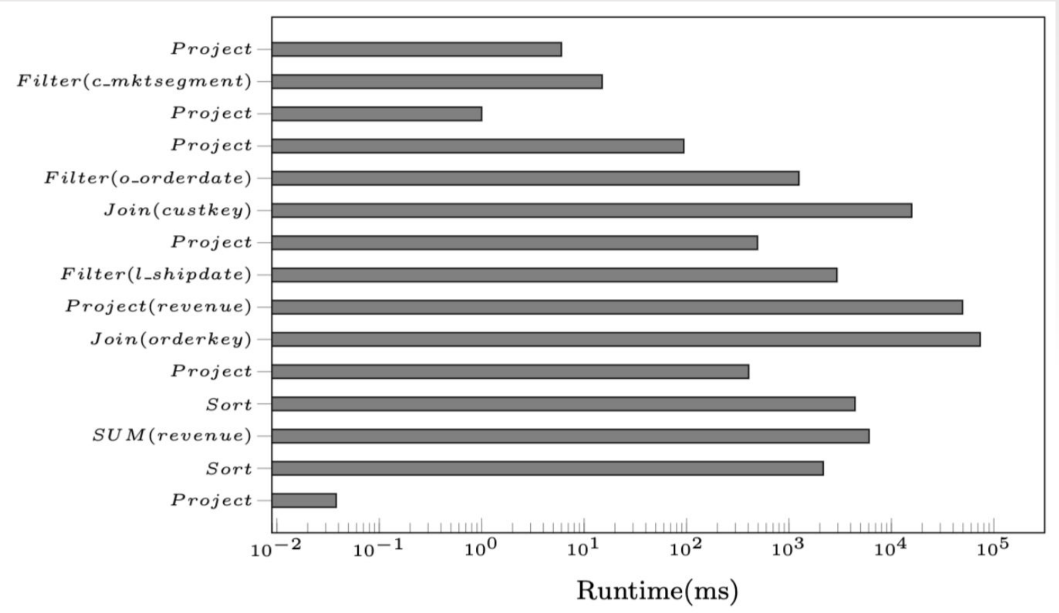
- Remove padding / dummy tags - leaks intermediate cardinalities (give up obliviousness)
  - Performance proportional to true intermediate result sizes
- Dummy rows are necessary to uphold ZK guarantee!

ZKSQL Performance vs Circuit-only Baseline



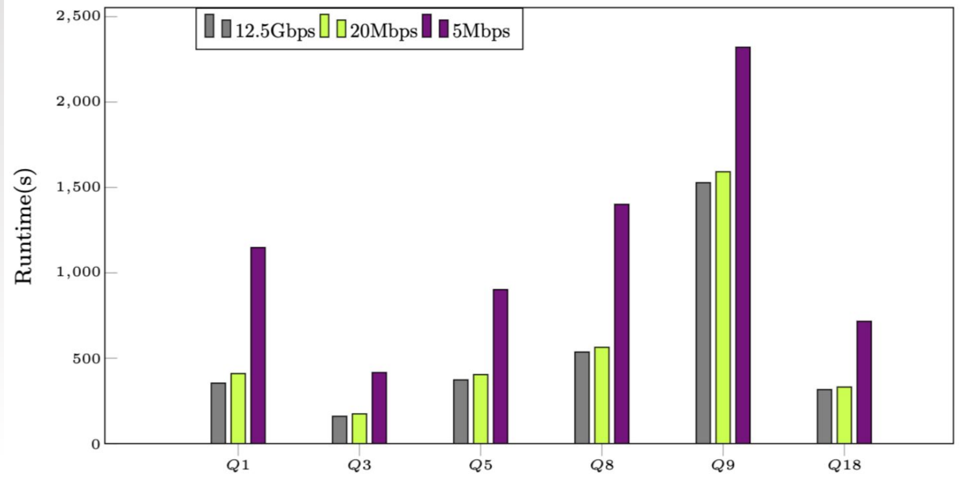
- Proving set relationships about results is more efficient than tracing the evaluation in circuits.
  - Queries with more sorts and joins benefit more from the improvement.
- TPC-H queries gain approximately average ~2 orders of magnitude improvement.

Operator-at-a-Time Performance



- Project with column reordering and filter takes minor cost.
- Math expressions are expensive in circuits.
  - Project and Aggregate.
- Sort is expensive because of oblivious C&S.
- Joins dominate runtime (~57% of *Q3* runtime).

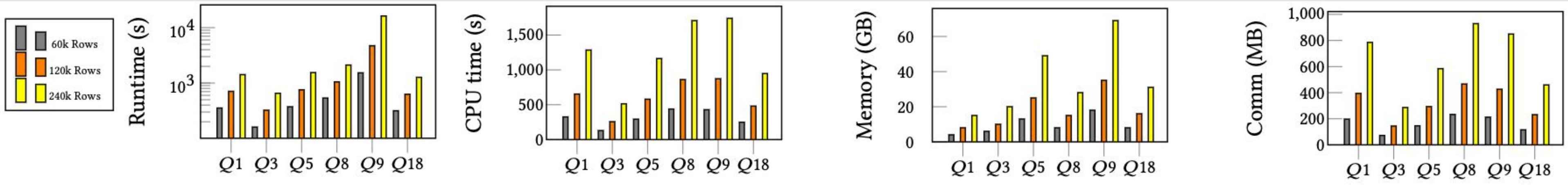
Performance with Varying Network Bandwidth



<u>**ZKSQL is insensitive to network conditions!**</u>

- By default, *r6i.4xlarge* instances have <u>*12.5Gbps* network</u> as our baseline.
- We restrict bandwidths to simulate low-bandwidth environments or geographically distributed machines.
- Throttling network to <u>*20Mbps*</u> does not affect runtime performance.
- Throttling network to <u>*5Mbps*</u> only causes <u>*2X slowdown*</u>.

Scale Up



- Except *Q9*, all other queries doubles their runtime as input sizes approximately double.
  - *Q9* has wider rows -> more time for paging data into CPU cache.
- CPU runtime is proportional to the runtime as input sizes increase.
- *Q9* occupies most memory among all queries, which ranges from 18 GB to 69 GB.
- Communication costs scale linearly with change of input size.

Financial Cost

Query	*Q1*	*Q3*	*Q5*	*Q8*	*Q9*	*Q18*	Total
60k Rows (Circuit-Only)	$0.22	$6.77	$10.69	$8.99	$12.22	$35.39	$74.28
60k Rows (ZKSQL)	$0.10	$0.04	$0.10	$0.14	$0.42	$0.09	$0.89
120k Rows (ZKSQL)	$0.20	$0.09	$0.21	$0.29	$1.27	$0.17	$2.23
240k Rows (ZKSQL)	$0.39	$0.18	$0.43	$0.57	$4.39	$0.34	$6.30

- Each *r6i.4xlarge* instance costs $1.01.
- Compared to ZKSQL, circuit-only approach takes ~1.2X-392X more monetary cost.
- In ZKSQL, costs for proving queries grow linearly as input size increases except for *Q9*.
  - 120k Rows takes 2.5X more cost than 60k Rows.
  - 240k Rows takes 2.8X more cost than 120k Rows.

Related work

- CorrectDB and VeriDB rely more on trusted hardware while leak some information on program traces.

  Sumeet Bajaj and Radu Sion. 2013. ***CorrectDB**: SQL engine with practical query authentication*. Proceedings of the VLDB endowment 6,7 (2013), 529-540.

  Wenchao Zhou, Yifan Cai, Yangqing Peng, Sheng Wang, Ke Ma, and Feifei Li. 2021. ***VeriDB**: An sgx-based verifiable database*. In Proceedings of the 2021 International Conference on Management of Data. 2182-2194.

- IntegriDB and vSQL use cryptographic verifiable computation in outsourcing setting.
  - Guarantee integrity, but *not privacy and zero knowledge* property.

  Yupeng Zhang, Jonathan Katzm and Charalampos Papamanthou. 2015. ***IntegriDB**: Verifiable SQL for Outsourcing Databases*. In ACM CCS 2015, Indrajit Ray, Ninghui Li, and Christopher Kruegel (Eds.). ACM Press, Denver, CO, USA, 1480-1491

  Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. ***vSQL**: Verifying Arbitrary SQL Queries over Dynamic Outsourced Databases*. In 2017 IEEE Symposium on Security and Privacy. IEEE Computer Society Press, San Jose, CA, USA, 863-880.

- ZK extension of vSQL provides ZK proofs, but does not support ad-hoc queries.

  Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. 2017. *A **Zero-Knowledge** Version of vSQL.* Cryptology ePrint Archive, Report 2017/1146. https://eprint.iacr.org/2017/1146.

Future work: E2E Maliciously Secure Data Federation

- Each data provider has access to partitions of private input data.
- Two-phase computation: ZK for local computation, MPC for joint computation
- Local phase:
  - Each party verifies local operators over its inputs with ZKSQL.
  - All pairs: data provider is a prover and each of others is a verifier.
- Convert commitments for each party to secret shares.
  - It is a core step needed from the cryptographic community.
- Joint phase:
  - All-but-one malicious security.
- ZKSQL + MPC is secure and efficient to query over multiple data providers!

Conclusion

- **ZKSQL** is the first work on verifiable and efficient query evaluation with zero-knowledge proofs for ad-hoc queries.
- ZKSQL authenticates its query evaluation one operator at a time.
- Our set-based protocols improves the pure circuits counterparts.
  - Up to 2 orders of magnitude speedup on TPC-H benchmark.
- Owing to universal composability, ZKSQL is extensible to new operator protocols.
- Next steps: securely and efficiently evaluate ad-hoc queries over multiple data providers using ZKSQL and MPC.

Thank You!

xiling.li@northwestern.edu

Northwestern | McCORMICK SCHOOL OF ENGINEERING