RESCU-SQL: Oblivious Querying for the Zero Trust Cloud

Xiling Li
Northwestern University
xiling.li@northwestern.edu

Gefei Tan
Northwestern University
gefeitan@u.northwestern.edu

Xiao Wang
Northwestern University
wangxiao@northwestern.edu

Jennie Rogers
Northwestern University
jennie@northwestern.edu

Soamar Homsi
Air Force Research Laboratory
soamar.homsi@us.af.mil

ABSTRACT

Cloud service providers offer robust infrastructure for rent to organizations of all kinds. High stakes applications, such as the ones in defense and healthcare, are turning to the public cloud for a cost-effective, geographically distributed, always available solution to their hosting needs. Many such users are unwilling or unable to delegate their data to this third-party infrastructure.

In this demonstration, we introduce RESCU-SQL, a zero-trust platform for resilient and secure SQL querying outsourced to one or more cloud service providers. RESCU-SQL users can query their DBMS using cloud infrastructure alone without revealing their private records to anyone. It does so by executing the query over secure multiparty computation. We call this system zero trust because it can tolerate any number of malicious servers provided one of them remains honest. Our demo will offer an interactive dashboard with which attendees can observe the performance of RESCU-SQL deployed on several in-cloud nodes for the TPC-H benchmark. Attendees can select a computing party and inject messages from it to explore how quickly it detects and reacts to a malicious party. This is the first SQL system to support all-but-one maliciously secure querying over a semi-honest coordinator for efficiency.

PVLDB Reference Format:
Xiling Li, Gefei Tan, Xiao Wang, Jennie Rogers, and Soamar Homsi. RESCU-SQL: Oblivious Querying for the Zero Trust Cloud . PVLDB, 16(12): 4086 - 4089, 2023.
doi:10.14778/3611540.3611627

1 INTRODUCTION

Users are increasingly turning to a cloud-first deployment for their DBMS applications. Here their database runs on infrastructure provided by one or more third-party cloud service providers (CSPs), such as Amazon AWS or Microsoft. This is an attractive option for running a globally available application because it provides high availabilty with pay-as-you-go pricing. On the other hand, this setup implicitly calls for entrusting the database's records to the CSP. Some high-stakes DBMS applications, such as defense and healthcare, would like to reap the benefits of a cloud-first deployment but are unable to do so owing to confidentiality or regulatory requirements. They cannot delegate responsibility for their records to a third party and need to work within a *zero trust cloud*. We focus on OLAP querying in this work.

For example, consider someone wearing a smartwatch to monitor their vital signs. As a wearable edge device, the watch has a limited ability to maintain a large volume of data for the person such as heart rate, blood oxygen and so on. Moreover, it is too computationally weak and low energy to perform analytics locally. Hence, the watch is connected to healthcare providers supported by their insurance. These providers maintain databases of similar data for other patients and offer individualized feedback to the watch user. However, those servers are usually operated by public CSPs and are vulnerable to data breaches. To mitigate this risk–while maintaining a lightweight resilient edge presence–the patient considers the cloud servers untrusted and the watch works with this outsourced DBMS via a slim coordinating interface.

We propose RESCU-SQL a platform for resilient and secure computing of SQL queries on the untrusted cloud. We offer a zero-trust cloud security model. Our setup consists of n *cloud servers* and a *trusted coordinator* or client. Although trusted coordinator is the data owner, they would like to store the data on the cloud for better resilience and improved availability to other users authorized to query the data. The n cloud servers store and query confidential data and they are secure against malicious adversaries. This means RESCU-SQL is secure even when $n-1$ of them are corrupted. These servers are hosted by one or more CSPs. In our example, the trusted coordinator is the watch user. This party is *semi-honest* in that we trust them to submit queries and also to generate randomness for the system's underlying cryptographic protocols.

Our system evaluates its queries using *secure multi-party computation (MPC)*, or cryptographic protocols that enable a group of mutually distrustful parties to jointly compute a function over its private inputs without revealing anything except (optionally) the output of the function. Our MPC protocol relies on the authenticated garbling approach proposed by Wang et al. [9] to achieve all-but-one malicious security, protecting against unauthorized data exposure even if $n-1$ out of n servers are corrupted.

In this work, we extend the authenticated garbling protocol by Wang et al. to the outsourced setting. Specifically, we leverage the trusted coordinator to create and distribute random authenticated secret shares, which otherwise would require expensive multi-round communication between all parties. This strategy significantly reduces the communication and memory cost of our protocol. For example, we measured the communication complexity by evaluating a standard MPC benchmarking circuit that computes the SHA-256 compression function: in our protocol, the maximum

Proceedings of the VLDB Endowment, Vol. 16, No. 12 ISSN 2150-8097.
doi:10.14778/3611540.3611627
First two authors contributed equally to this work

data a server needs to send is 4.28 MB, which is a 20x improvement over the result reported in [9] for the corresponding three-party case. More importantly, we achieve optimal asymptotic improvement in memory usage. Their protocol needs to store the whole circuit in memory, requiring $O(|C|)$ memory for a circuit size of $|C|$. In contrast, with the help of a trusted coordinator, evaluating the authenticated garbled circuit in our protocol requires the same memory asymptotically as evaluating the circuit in the clear.

With our all-but-one maliciously secure MPC protocol, we ensure that client–the trusted coordinator in our setting or smartwatch in our running example–stays secure in the presence of up to $n-1$ corrupted parties among n cloud servers. To our best knowledge, there is limited work for query evaluation on zero-trust cloud in the presence of any malicious dishonest majority. For example, similar to RESCU-SQL, Senate [5] supported analytical queries over maliciously secure MPC protocols, but it is not as scalable as ours although it also decomposes its queries into operator-at-a-time secure evaluation. Our MPC protocol may be of independent interest for other outsourced tasks. Our work is implemented atop EMP-toolkit [8], although our protocols are previously unpublished.

Our contributions are:

- We propose RESCU-SQL, the first pragmatic OLAP system with all-but-one malicious security for ad-hoc SQL queries.
- We extend the state-of-the-art malicious MPC protocol to the outsourced setting, leveraging a lightweight trusted coordinator to obtain *optimal* memory usage and significantly reduced communication overhead.
- We verify the performance of this system on the TPC-H benchmark and analyze the cost of each operator.
- We propose a demonstration of a user-friendly dashboard that monitors query evaluation using TPC-H and invites users to explore its detection of malicious cloud servers.

We organize the rest of the paper as follows. First, we describe the system architecture, zero-trusting setting, and threat model in Section 2; we evaluate our prototype in Section 3. Next, we will outline our proposed demo experience in Section 4. We describe related work in Section 5 and conclude.

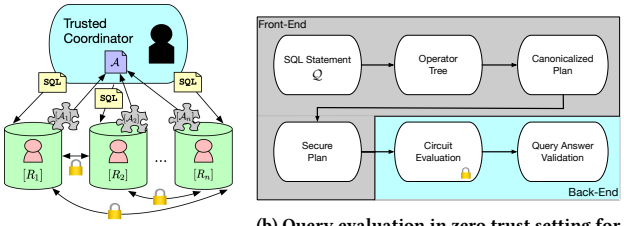
2 RESCU-SQL OVERVIEW

We now describe the setting of outsourced query evaluation in the zero trust cloud. We then walk through the steps the engine will take in a SQL query and execute it on the compute nodes.

2.1 Zero Trust Cloud

Figure 1(a) shows the workflow of RESCU-SQL during its outsourced query evaluation. In this setting, we have two types of participants: n cloud servers, shown as cylinders, and one trusted coordinator, displayed as the rounded rectangle.

Before it runs its first query, RESCU-SQL has a one-time setup phase (not shown) where the data owner secret shares their records to distribute them over our n untrusted nodes for querying. We call the private input data R and we show its corresponding secret shares as $[R]$. Server i is responsible for the shares $[R_i]$. This is equivalent to each host receiving an encrypted copy of the private input records for query evaluation but not having the key with which to decrypt it. The secret sharing protocol is maliciously secure. As with our computation, when we deploy over n hosts the



(a) System Architecture

(b) Query evaluation in zero trust setting for each cloud server.

Figure 1: System Overview

data will remain confidential for up to $n-1$ corruptions. Section 2.3 has more details on our security model.

The database schema is known to all parties, as is the number of rows in each table. RESCU-SQL's query evaluation is oblivious, meaning its instruction traces are data-independent. We do this using well-known operator algorithms similar to [2, 5, 7]. The trusted coordinator alone learns the query answers.

Our client is a lightweight trusted coordinator that is semi-honest. This party will not deviate from our MPC protocol but may try to deduce information about the private records from participating in a query evaluation. This client is the data owner. The trusted coordinator generates randomness that it sends to the cloud servers that they use for secure query evaluation. Our query lifecycle starts when the trusted coordinator sends its SQL statement, $\mathcal{Q}$, to all nodes. $\mathcal{Q}$'s execution plan is known to all parties.

The n nodes shown in Figure 1(a) jointly compute the answer to the SQL query provided by the trusted coordinator. They evaluate each gate by passing authenticated messages amongst themselves, shown with padlocks. The output of each gate is itself a secret share. We call the query answer $\mathcal{A}$.

In order to make progress on computing a query, all n servers must remain good faith participants throughout the protocol. If one node drops out or injects faulty messages into this exchange, then the remaining ones will detect this deviation and abort thereby ensuring that no unauthorized information is recovered by anyone. This distributes the trust among more parties. If RESCU-SQL is deployed over multiple CSPs, an attacker has to compromise all of them before there is any risk of unauthorized data access.

When the nodes are done evaluating the query's circuits, they each hold secret shares for $\mathcal{A}$, i.e. node i holds $[\mathcal{A}_i]$. Each host sends its shares of $\mathcal{A}$ to the trusted coordinator over an encrypted link to prevent eavesdropping. The trusted coordinator assembles the shares thereby revealing $\mathcal{A}$.

2.2 Outsourced Query Lifecycle

When the trusted coordinator submits a query to RESCU-SQL, it first sends its SQL statement to each of the n cloud servers. Figure 1(b) has the steps each server takes to convert the query into an executable plan and run it. This workflow consists of two parts. First, the Java front-end converts $\mathcal{Q}$ into a directed acyclic graph (DAG) of database operators and organizes it to run efficiently in the outsourced MPC setting. After that, the back-end runs the garbled circuits in C++ and returns $\mathcal{A}$ to the client (trusted coordinator).

The front-end is similar to a conventional SQL parser and it readies the plan for execution in MPC. First, the system receives a SQL statement from the client and verifies that it is syntactically correct with respect to the schema. Then it parses the query into

Table 1: Runtime in seconds for plaintext vs RESCU-SQL

	$Q1$	$Q3$	$Q5$	$Q8$	$Q9$	$Q18$
Plaintext	0.004	0.001	0.004	0.003	0.005	0.003
RESCU-SQL	341.764	507.914	802.051	1,373.1	8,487.79	1,827.52
Slowdown	8,544×	507,914×	200,513×	457,700×	1,697,558×	609,173×

a DAG. RESCU-SQL then applies heuristics to minimize the size of our garbled circuits by eagerly projecting out attributes and combining operators to minimize our passes over the data. Our front-end builds atop Apache Calcite [3] and it outputs the query tree in JSON for use in the back-end.

The back-end then parses the JSON plan and maps each node in the query tree to a cryptographic protocol. RESCU-SQL supports the following operators: Select, Project, Join, Sort, and Aggregate. When the system executes a query, it first does a sequential scan over the secret shares for each leaf in the query tree. Then it executes the plan bottom-up one operator at a time. It executes the operators' circuits depth-first as implemented in EMP Toolkit [8]. Finally, all parties jointly validate the query answer.

We protect the data access patterns of each operator by injecting *dummy tuples* or placeholders when a tuple is conditionally emitted from an operator. For example, when Filter removes a tuple, we replace it with a tombstone row so that an adversary cannot determine the positions of removed tuples. To do so, we add a *dummy tag* at the end of each row and set it to true when we disregard a tuple for all subsequent query evaluations. The system thus hides the true cardinality of each intermediate result. Once the root node of Q is finished, all parties return their secret shares of $\mathcal{A}$ to the trusted coordinator. The client reconstructs the query results using shares from all parties. Table 1 demonstrates RESCU-SQL runtime using the experimental configuration in Section 3. We see it has a slowdown of $4-7$ orders of magnitude compared to plaintext runtime over the TPC-H benchmark, which is a price we pay for outsourcing MPC in a zero-trust setting.

2.3 Threat Model

In our work, we adopt a threat model where a malicious adversary can corrupt any number of servers provided at least one remains honest. Specifically, cloud servers are modeled as malicious parties that could deviate from the protocol arbitrarily, while the trusted coordinator is modeled as a semi-honest party that follows the protocol but may attempt to learn as much information about the data as possible while doing so. Under this threat model, our protocol can tolerate up to $n-1$ corrupted cloud servers while also ensuring the trusted coordinator learns nothing but the query answer. To achieve this goal, We design our system on a maliciously secure MPC protocol We then build the SQL processing engine on top of our MPC protocol. Then, the security of our MPC protocol naturally translates to the all-but-one malicious security of RESCU-SQL.

We remark that viewing the trusted coordinator (client) as semi-honest in our threat model may be seen as somewhat restrictive, as the client is usually the data owner in our setting, and thus data privacy is inherently guaranteed. We believe, however, this definition aligns with the practical demands of our outsourced setting, where the client's aim is to utilize high-performance computing resources that are not available locally. Consequently, our system should not rely on the client's resources: the client should not maintain local copies of the data or receive any intermediate query results; rather, its role should be limited to providing the data at setup time, submitting queries, and assembling their answers.

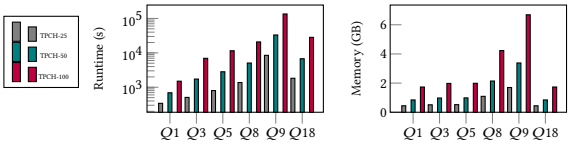


Figure 2: Performance on the trusted coordinator over data of increasing sizes.

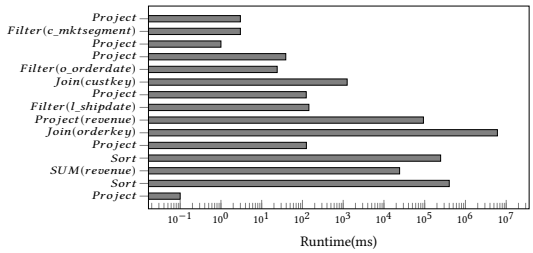


Figure 3: Operator-level performance on TPC-H Q3.

3 EXPERIMENTAL RESULTS

In this section, we first describe system implementation and experimental setup. Then, we discuss the cost of outsourcing MPC based on the system and operator-level cost analysis.

Setup We examine our system over a subset of queries in TPC-H benchmark [6] stored in the PostgreSQL engine, namely Q1, Q3, Q5, Q8, Q9 and Q18. They vary in their level of complexity with respect to the number of operators. For our experiments, we select $n = 3$ and allow up to two corrupted parties (see Section 2.1), so we use 4 Linux servers including one trusted coordinator. In addition, we create 3 different sizes of data from TPC-H database, named *TPCH-25*, *TPCH-50* and *TPCH-100*. To do so, we truncate customer and supplier tables to 25, 50, or 100 tuples and prune the other tables using their primary key-foreign key relationships.

3.1 End-to-End Performance

Recall that Table 1 shows RESCU-SQL takes several orders of magnitude more time than conventional plaintext execution on TPCH-25. Our strong security guarantees come at a substantial slowdown. Our results reveal that our performance naturally gets slower when we tackle more complex queries such as Q8 and Q9 with their cascades of joins and computationally-intense aggregation. To address this, in future work we may investigate parallelizing this query processing. We may also benchmark how our performance improves with access to more hardware resources.

Figure 2 shows the scalability of RESCU-SQL when we extended our evaluation to TPCH-50 and TPCH-100. We show the runtimes for the trusted coordinator because it has the longest execution time. As the data size doubles, runtime grows at the about same rate. Memory footprint grows modestly as we scale up; at TPC-H 25 we use up to 1.7 GB and at TPC-H 100 we reach 6.7 GB. This growth rate is more gradual as we amortize our setup phase with the trusted coordinator for the protocols in each query evaluation.

3.2 Operator-Level Results

Now, we granulate our analysis into operator level. For example, Figure 3 illustrates runtime per operator in $Q3$ over TPCH-25. Join operator, simultaneously evaluating over two tables, dominates–a total of 89%. Sort operators cost 9% runtime due to intricate Bitonic sorting network [1]. Aggregate and Project(revenue) take 0.3% and 1% respectively, mainly for arithmetic calculation for *revenue*. Filter and other Project operators take minor overall time, merely 0.006%.

4 DEMONSTRATION

Our demonstration will greet participants with a dashboard for the trusted coordinator with which participants launch a query, monitor its resource utilization, and inject corruptions into computing parties. Figure 4 provides an overview of the interface, where the lightweight trusted coordinator will run locally and the outsourced computing servers will be deployed on one or more CPSs.

First, attendees will select one of the queries from our subset of TPC-H. They click the "Run" button to launch the query. Then, the coordinator generates randomness for underlying MPC protocols and sets up its connection with each server. The coordinator sends the query to each server and launches its worker process. This starts the timer in the upper right corner of each server's panel.

Each computing party has a panel to provide real-time updates on its resource utilization and overall health. If a server is healthy, it is shown in white. If it is corrupted or the trusted coordinator aborts then it becomes shaded in red (e.g. Server B). The bottom of each panel displays query usage statistics for memory, CPU, and kilobits per second of data for communication among the nodes.

While a query is running, attendees may elect to corrupt one of the servers. They will select a server and send a message. The server will broadcast the message to its $n-1$ peers, thereby deviating from the protocol. We will show in real time how each computing node will detect this defection and abort. Each one will incrementally turn red until all nodes abort. Recall that we need all n hosts to remain uncorrupted to uphold our security guarantees. Otherwise, we would implicitly be trusting these outsourced servers.

5 RELATED WORK

We now briefly survey related research. There is significant research interest in computing SQL queries obliviously using MPC [2, 7] and secure enclaves [4, 10]. The MPC-based solutions work in a semi-honest setting. The secure enclave setting offers stronger guarantees but requires specialized hardware. In contrast, we address a single data owner outsourcing to the untrusted cloud.

Senate [5] is a SQL platform for collaborative analytics that achieves malicious security. This work most closely resembles our own. It is possible to adapt this platform for outsourcing but the protocol would not be as efficient as ours since Senate is not designed for the outsourcing setting. In particular, with the help of the data owner (i.e, trusted coordinator), we are able to achieve much higher scalability and efficiency. Instead of executing a single costly query between all parties, Senate breaks it into smaller queries between some parties. The final result is obtained through a verified joint computation. In certain cases, this decomposition allows for parallelization, which may improve runtime. Note that this optimization relies on assumptions about the collaborating

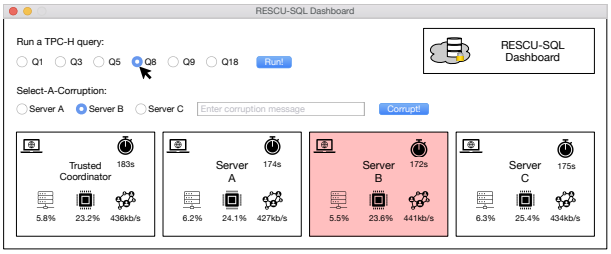


Figure 4: RESCU-SQL Dashboard

data owners being able to access their own data in the clear which is not applicable in the zero trust setting.

6 CONCLUSIONS

We propose to demo RESCU-SQL, a SQL-over-MPC framework for the zero trust cloud. This system makes it possible for a data owner to outsource their operations in a geographically distributed deployment that will detect and prevent an attacker from gaining unauthorized access to its data if at least one compute node out of n remains uncorrupted. Demo goers will get hands-on experience with launching TPC-H queries in this setting and injecting corruptions into its query evaluation.

ACKNOWLEDGMENTS

This work is supported by Air Force Research Lab grant #FA8750-22-2-0156. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the Air Force Research Laboratory or the U.S. Government.

—

Approved for Public Release on 26 July 2023. Distribution Is Unlimited. Case Number: AFRL-2023-3625.

REFERENCES

[1] K. E. Batcher. 1968. Sorting Networks and Their Applications. In Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference (Atlantic City, New Jersey) (AFIPS '68 (Spring)). Association for Computing Machinery, New York, NY, USA, 307–314. https://doi.org/10.1145/1468075.1468121

[2] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel N Kho, and Jennie Rogers. 2017. SMCQL: Secure Query Processing for Private Data Networks. Proc. VLDB Endow. 10, 6 (2017), 673–684.

[3] Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J. Mior, and Daniel Lemire. 2018. Apache Calcite: A Foundational Framework for Optimized Query Processing Over Heterogeneous Data Sources. In Proceedings of the 2018 International Conference on Management of Data (Houston, TX, USA) (SIGMOD '18). Association for Computing Machinery, New York, NY, USA, 221–230. https://doi.org/10.1145/3183713.3190662

[4] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2020. Oblivious coopetitive analytics using hardware enclaves. In Proceedings of the Fifteenth European Conference on Computer Systems. 1–17.

[5] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M Hellerstein. 2021. Senate: A Maliciously-Secure MPC Platform for Collaborative Analytics.. In USENIX Security Symposium. 2129–2146.

[6] Transaction Processing Council. 2023. TPC-H Benchmark. http://www.tpc.org/tpch/

[7] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In Proceedings of the Fourteenth EuroSys Conference 2019. 1–18.

[8] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.

[9] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. 39–56. https://doi.org/10.1145/3133956.3133979

[10] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform.. In NSDI, Vol. 17. 283–298.