Parallel Oblivious Joins using Radix Partitioning

Nafis Ahmed Sujaya Maiyya

David R. Cheriton School of Computer Science
University of Waterloo
{nafis.ahmed,smaiyya}@uwaterloo.ca

1 Abstract

Due to the cloud's pay-as-you-go model, individuals and organizations are increasingly moving from managing local servers to renting storage from third-party providers. This convenience, however, risks compromising the privacy of outsourced data. Encrypting data before outsourcing preserves confidentiality, but it prevents the cloud from directly processing application queries. Hardware enclaves such as Intel SGX and Intel TDX enable cloud servers to process encrypted data without violating confidentiality guarantees, yet they remain vulnerable to side-channel leakages caused by data-dependent memory access patterns and control flow.

Oblivious query processing enables remote secure computation over encrypted data while mitigating such side-channel leakages. We present parallel oblivious algorithms for binary non-foreign key and foreign-key joins by introducing a new primitive: *oblivious radix partitioning*. Compared to the state-of-the-art, Obliviator, our join algorithms reduce the required number and size of the obliviously sorted tables, offering significant speed-ups. Beyond joins, our oblivious radix partitioning technique serves as a standalone primitive applicable to a broad class of problems such as oblivious group-by and aggregation, private set intersection and union, and private contact discovery.

2 Overview

We observe that existing oblivious join algorithms using enclaves [2, 3] are *sort-merge joins*. This research began with the question: *can we design an oblivious hash join?* At a high level, plaintext hash joins on tables R and S first compute the hash of the join keys of the smaller table R and store each key's corresponding payload in a hash table. In the second step, table S probes the hash table by hashing its keys to find matching entries.

Making this algorithm oblivious, however, raises non-trivial challenges because (i) repeated join keys in R create hash collisions and can reveal the number of unique keys in R, and/or (ii) repeated probing of the same hash table entry leaks the distribution of repeated keys in S – both leaking data dependent information to an adversary. Mitigating this leakage requires de-duplicating repeated keys from both tables, which in turn requires an oblivious sort, rendering this scheme as a sort-merge join. Therefore, while an oblivious hash join *may* be constructed when both R and S have unique keys, this fails to be secure for generalized joins. Given that our join algorithms have to be sort-merge based, we ask: *can the algorithm leverage hash-based techniques in the merge step to improve efficiency?*

We answer affirmatively by first proposing an oblivious radix partitioning approach that divides input arrays into disjoint partitions without leaking duplicate counts, unlike vanilla radix partitioning. This is especially important for join operations, where duplicate keys are common. Our join algorithms apply oblivious radix partitioning independently to each input table, enabling *tuple comparisons only within corresponding partitions*. When input tables are presorted, our oblivious join algorithm is the first to avoid re-sorting them.

We deploy our join algorithms [1] on Azure's Intel TDX machines. Our evaluation compares them with Obliviator [3] across synthetic workloads, five real-world datasets, and TPC-H queries. Results demonstrate the advantage of reducing both the number and size of obliviously sorted tables, with the difference in execution times being more pronounced for large datasets (100M to 500M tuples). When scaling from 2 to 32 threads, our approach maintains consistent speedups across dataset sizes, unlike the baseline, demonstrating the effectiveness of radix partitioning in parallel oblivious query execution.

References

[1] Code. https://github.com/dsg-uwaterloo/obl-radix.

[2] S. Krastnikov, F. Kerschbaum, and D. Stebila. Efficient oblivious database joins. *VLDB*, 2020.

[3] A. Mavrogiannakis, X. Wang, I. Demertzis, D. Papadopoulos, and M. Garofalakis. Obliviator: Oblivious parallel joins and other operators in shared memory environments. *USENIX Security*, 2025.