ObliDB: Oblivious Query Processing for Secure Databases

Saba Eskandarian
Stanford University

Matei Zaharia
Stanford University
Private Data in the Cloud

Compromised cloud can:

- Read data
- Read queries
- Alter data
Hardware Enclaves

A trusted component in untrusted hardware

- Isolation through protected memory
- Authenticity through attestation

Currently available through Azure and IBM cloud, among others
Enclaves in the Cloud

Enclave memory is limited, but data is big!
Enclaves in the Cloud
Enclaves in the Cloud

Malicious attacker can observe access patterns to encrypted data!
Enclaves in the Cloud

Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation

Mohammad Saiful Islam, Mehmet Kuzu, Murat Kantarcioglu
Jonsson School of Engineering and Computer Science
The University of Texas at Dallas
{saiful, mehmet.kuzu, muratk}@utdallas.edu

Abstract

The advent of cloud computing has ushered in an era of mass data storage in remote servers. Remote data storage offers reduced data management overhead for data owners in a cost effective manner. Sensitive documents, however, need to be stored in encrypted format due to security concerns in the cloud. But, the advantage of cloud data storage is lost if the user cannot selectively retrieve segments of their data. Therefore, we need secure and efficient search schemes to selectively retrieve sensitive data from the cloud. The need for such protocols are also recognized by researchers from major IT companies such as Microsoft [14].
Enclaves in the Cloud

Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation

Mohammad Saiful Islam, Mehmet Kuzu, Jonsson School of Engineering and Computer Science
The University of Texas at Austin
{saiful, mehmet.kuzu, muratk}

Abstract

The advent of cloud computing has ushered in an era of massive data storage in remote servers. Remote data storage offers reduced data management overhead for data owners in a cost-effective manner. Sensitive documents, however, need to be stored in encrypted format due to security concerns.

Observing and Preventing Leakage in MapReduce

Olga Ohrimenko
Microsoft Research
oochrim@microsoft.com

Manuel Costa
Microsoft Research
manuelc@microsoft.com

Cédric Fournet
Microsoft Research
fournet@microsoft.com

Christos Gkantsidis
Microsoft Research
gkantsidis@microsoft.com

Markulf Kohlweiss
Microsoft Research
markulf@microsoft.com

Divya Sharma
Carnegie Mellon University
divyasharma@cmu.edu

ABSTRACT

The use of public cloud infrastructure for storing and processing large datasets raises new security concerns. Current solutions propose encrypting all data, and accessing it in plaintext only within secure hardware. Nonetheless, the distributed processing of large amounts of data still involves intensive encrypted communications between different processing and network storage units, and these communications patterns may leak sensitive information.

We consider secure implementation of MapReduce jobs, and measure their inter-process traffic between nodes and data, in particular when they involve complex, dynamic intermediate data. Conversely, limited trust assumptions on the cloud infrastructure may lead to efficient solutions, but their actual security guarantees are less clear.

As a concrete example, VCG [26] recently showed that, by relying on the new Intel SGX infrastructure [19] to protect local mapper and reduce processing, one can adapt the popular Hadoop framework [2] and achieve strong integrity and confidentiality for large MapReduce tasks with a small performance overhead. All data is systematically AES-GCM-encrypted, except when processed within hard-
A persistent passive attacker can extract even more information by observing an application’s access patterns … In our case study applications, this reveals users’ medical conditions, genomes, and contents of shopping carts.
Naive SELECT is not oblivious!
Naive SELECT is not oblivious!
Naive SELECT is not oblivious!

Input Table

Output Table
Naive SELECT is not oblivious!

Input Table

Output Table
Naive SELECT is not oblivious!
Naive SELECT is not oblivious!
Naive SELECT is not oblivious!

Input Table

<table>
<thead>
<tr>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>

Output Table

<table>
<thead>
<tr>
<th>*</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
<tr>
<td>*</td>
</tr>
</tbody>
</table>
Naive SELECT is not oblivious!

Watching when we write to the output table reveals exactly which rows of the input table we select!
Toward Obliviousness

Prior work solves pieces of the obliviousness problem very well
Toward Obliviousness

Prior work solves *pieces* of the obliviousness problem very well

Opaque provides obliviousness for analytic queries that scan entire tables, but no support for indexes
Toward Obliviousness

Prior work solves pieces of the obliviousness problem very well.

Opaque provides obliviousness for analytic queries that scan entire tables, but no support for indexes.

Oblix provides an oblivious index, but using an oblivious index to process a query obliviously is still non-trivial.
Toward Obliviousness

Prior work solves pieces of the obliviousness problem very well

Opaque provides obliviousness for analytic queries that scan entire tables, but no support for indexes

Oblix provides an oblivious index, but using an oblivious index to process a query obliviously is still non-trivial

This work: ObliDB, first system to provide obliviousness for general database read workloads over multiple access methods
ObliDB Overview

- Tables stored encrypted in unprotected memory, enclave only holds metadata
- Two oblivious storage methods: flat tables and oblivious indexes
- Supports most SQL operations
- Various algorithms for each operation - can pick best option at runtime
Security Guarantees

ObliDB protects data and query parameters against an attacker with full control of the OS and VMM

- Detects any malicious attempt to tamper with data
- Leaks only query selectivity, table sizes (including intermediate tables), and query plan
- Optional padding mode available to hide table sizes and query selectivity
- **Assumption**: limited oblivious memory pool
Oblivious Operators

- **Selection**
  - Small
  - Large
  - Continuous
  - Hash

- **Grouping and Aggregation**

- **Joins**
  - Oblivious hash join
  - Oblivious sort-merge join (from Opaque)
  - Zero oblivious memory sort-merge join
Oblivious Operators

- Selection
  - Small
  - Large
  - Continuous
  - Hash

- Grouping and Aggregation

- Joins
  - Oblivious hash join
  - Oblivious sort-merge join (from Opaque)
  - Zero oblivious memory sort-merge join

Oblivious optimizer chooses best algorithm for each query at runtime
Oblivious Operators

● Selection
  ○ Small
  ○ Large
  ○ Continuous
  ○ Hash

● Grouping and Aggregation

● Joins
  ○ Oblivious hash join
  ○ Oblivious sort-merge join (from Opaque)
  ○ Zero oblivious memory sort-merge join

Oblivious optimizer chooses best algorithm for each query at runtime
Oblivious SELECT

“Large” SELECT Algorithm: use when almost the whole table is selected
Oblivious SELECT

“Large” SELECT Algorithm: use when almost the whole table is selected
Oblivious SELECT

“Continuous” SELECT algorithm: use when a continuous range of rows is selected
Oblivious SELECT

“Continuous” SELECT algorithm: use when a continuous range of rows is selected
ObliDB

Performance highlights:

- 1.1-19x faster than Opaque (on Big Data Benchmark queries)
- Within 2.6x of Spark SQL (on Big Data Benchmark queries)

See paper for system details, more oblivious operators, and full evaluation

Source Code: https://github.com/SabaEskandarian/ObliDB
Questions/Contact: saba@cs.stanford.edu