ObliDB: Oblivious Query Processing using Hardware Enclaves

Saba Eskandarian       Matei Zaharia
Private Data in the Cloud

Compromised cloud can:

- Read data
- Read queries
- Alter data
Private Data in the Cloud

NSA spying fiasco sending customers overseas

NSA spy program could lead to loss of business for some hosting vendors, experts say

Read data
Read queries
Alter data
Private Data in the Cloud

NSA spying fiasco sending customers overseas

Uber will pay $20,000 fine in settlement over 'God View' tracking
Private Data in the Cloud

Compromised cloud can:
- Read data
- Read queries
- Alter data

NSA spying fiasco sending customers overseas

Uber will pay $20,000 fine in settlement over 'God

Lyft Investigates Allegation That Employees Abused Customer Data
Private Data in the Cloud

Every single Yahoo account was hacked - 3 billion in all
Private Data in the Cloud

NSA spying fiasco sending customers overseas

Uber will pay $20,000 fine in settlement over 'God

V. Left Lane' texting allegations. The

Every single Yahoo account was hacked. 3

Atos, IT provider for Winter Olympics,
hacked months before Opening
Ceremony cyberattack
(Purely) Cryptographic Solutions

*Huge* body of work on how to protect databases with cryptography

Various tradeoffs between functionality, performance, and security, but relatively little industry adoption thus far.

(Purely) Cryptographic Solutions

**Huge** body of work on how to protect databases with cryptography

Various tradeoffs between functionality, performance, and security, but relatively little industry adoption thus far.

Minimum Requirements:

- Broad support for common workloads
- Acceptable performance
- Strong security guarantees

Outline

- Intro: Protecting Cloud Data
- **Hardware Enclaves and Obliviousness**
- ObliDB Design
  - Threat Model and Security Guarantees
  - SELECT Algorithms
  - Oblivious Indexes
- ObliDB Performance Evaluation
Hardware Enclaves

A trusted component in an untrusted system

- Uses protected memory to isolate enclave execution from compromised OS
- Proves that it is an authentic enclave running the desired code with *attestation*
- Enclaves in our implementation use Intel SGX
Azure Confidential Computing Heralds the Next Generation of Encryption in the Cloud

BY ERICA PORTNOY | SEPTEMBER 18, 2017

For years, EFF has commended companies who make cloud applications that encrypt data in transit. But soon, the new gold standard for cloud application encryption will be the cloud provider never having access to the user’s data—not even while performing computations on it.

Microsoft has become the first major cloud provider to offer developers the ability to build their applications on top of Intel’s Software Guard Extensions (SGX) technology, making Azure “the first SGX-capable servers in the public cloud.” Azure customers in Microsoft’s Early Access program can now begin to develop applications with the “confidential computing” technology.
“We are exploring future backends based on AMD Secure Encryption Virtualization (SEV) technology, Intel® Software Guard Extensions (Intel® SGX)”
Enclaves in the Cloud

Enclave space 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 Kantarcioğlu
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 an cost effective manner. Sensitive documents, however, need to be stored in encrypted format due to security concerns. 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 Engin and Computer Scien
The University of Texas a (saiful, mehmet.kuzu, muratk)

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. However, sensitive documents, however, need to be stored in encrypted format due to security concerns.

Observing and Preventing Leakage in MapReduce

Olga Ohrimenko
Microsoft Research
ochrim@microsoft.com

Christos Gkantsidis
Microsoft Research
gkantsidis@microsoft.com

Cédric Fournet
Microsoft Research
fournet@microsoft.com

Manuel Costa
Microsoft Research
manuelc@microsoft.com

Markulf Kohlweiss
Microsoft Research
markulf@microsoft.com

Divya Sharma
Carnegie Mellon University
divyasarma@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 those communications patterns may leak sensitive information.

We consider secure implementation of MapReduce jobs, and analyze their intermediate data flow between instances 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, VCS [20] recently showed that, by relying on the new Intel SGX infrastructure [19] to protect local mapper and reducer processing, one can adapt the popular Filebot 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-
Enclaves in the Cloud

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

Mohammad Saiful Islam, Mehmet Kuzu
Jonsson School of Engin
and Computer Scien

Breaking Web Applications Built On Top of Encrypted Data

Paul Grubbs
Cornell University
pag225@cornell.edu

Richard McPherson
UT Austin
richard@cs.utexas.edu

Muhammad Naveed
USC
mnaveed@usc.edu

Thomas Ristenpart
Cornell Tech
ristenpart@cornell.edu

Vitaly Shmatikov
Cornell Tech
shmat@cs.cornell.edu

ABSTRACT

We develop a systematic approach for analyzing client-server applications that aim to hide sensitive user data from untrusted servers. We then apply it to Mylar, a framework that uses multi-key searchable encryption (MKSE) to build Web applications on top of encrypted data.

We demonstrate that (1) the Popa-Zhukovich model for MKSE does not imply security against either passive or active attackers; (2) Mylar-based Web applications reveal users' activity on the server but not interfering with its operations; and active attacks involving arbitrary malicious behavior. We then work backwards from these adversarial capabilities to models. This approach uncovers significant challenges and security-critical decisions faced by the designers of BoPETs: how to partition functionality between the clients and the server, which data to encrypt, which access patterns can leak sensitive information, and more.

We then apply our methodology to a recent BoPET called FasterPET, which is a more complex version of Web applications.

"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"
Goal: Obliviousness

Leakage attacks observe *access patterns* to protected memory

Problem: Leakage of access patterns *completely compromises* security

Solution: design enclave operation to be *oblivious* of input data
Introducing ObliDB

**Functionality:**
Oblivious query processing algorithms *for both transactional and analytic queries*
Supports most SQL operations (SELECT, GROUP BY, JOIN, various aggregates)

**Security:**
Protects against powerful attacker with full control of the OS

**Performance:**
Point queries 7-22x faster than (non-enclave) prior work (Sophos, HIRB)
Analytic queries 20-330x faster than naive, 1-19x faster than prior work (Opaque)
Threat Model

ObliDB protects against an attacker with full control of the OS who can:

- Read and tamper with all of untrusted memory
- Pause and resume enclave execution
- Observe access patterns to untrusted memory
- Monitor network communications
- Know auxiliary information about data stored
Threat Model

ObliDB protects against an attacker with full control of the OS who can:

- Read \textit{and tamper with} all of untrusted memory
- Pause and resume enclave execution
- Observe access patterns to untrusted memory
- Monitor network communications
- Know auxiliary information about data stored
- \textbf{Assumption:} limited oblivious memory pool (same as Opaque)
Security Guarantees

ObliDB protects data and query parameters:

- 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
ObliDB Overview

Oblivious database engine with support for both transactional and analytic queries

Tables stored encrypted in untrusted memory but access patterns hidden

Two storage methods: linear tables and oblivious indexes

Enclave used to store keys/metadata and as working space for sensitive operations
SELECT Algorithms
Storage Methods: Linear

Access every block every time!

Good when accessing most blocks anyway

Used when we only need oblivious analytics

Decouple memory accesses from sensitive data

Point Read: $O(N)$
Large Read: $O(N)$
Insertion: $O(1)$
Deletion: $O(N)$
Storage Methods: Linear

Access every block every time!

Good when accessing most blocks anyway

Used when we only need oblivious analytics

Point Read: $O(N)$
Large Read: $O(N)$
Insertion: $O(1)$
Deletion: $O(N)$

Decouple memory accesses from sensitive data
Oblivious SELECT

First pass over data: determine size of output table, pick strategy to satisfy query
Oblivious SELECT

First pass over data: determine size of output table, pick strategy to satisfy query

Q: Why not just select at the same time as the first pass?
A: Naive SELECT is not oblivious!
Oblivious SELECT

Naive SELECT is not oblivious!

Input Table

| * |  
| * |
| * |
| * |
| * |
| * |

Enclave

“Can only fit 2 rows”

Output Table
Oblivious SELECT

Naive SELECT is not oblivious!

Input Table

*  
*  
*  
*  
*  
*  
*  

Enclave

“Can only fit 2 rows”

Output Table
Oblivious SELECT

Naive SELECT is not oblivious!
Oblivious SELECT

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>
</tbody>
</table>

Output Table

"Can only fit 2 rows"

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

Naive SELECT is not oblivious!

Input Table

*  
*  
*  
*  
*  
*  

Enclave
“Can only fit 2 rows”

Output Table

*  

*  

*  

*  

*  

*  

Oblivious SELECT

Naive SELECT is not oblivious!
Oblivious SELECT

Naive SELECT is not oblivious!
Oblivious SELECT

Naive SELECT is not oblivious!

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

“Small” SELECT algorithm

Input Table

```
* 
* 
* 
* 
* 
* 
```

Enclave

“Can only fit 2 rows”

Output Table

```
* 
* 
```
Oblivious SELECT

“Small” SELECT algorithm

Input Table

*  
*  
*  
*  
*  

Output Table

*  
*  
*  

Enclave

“Can only fit 2 rows”
Oblivious SELECT

“Small” SELECT algorithm

Input Table

* 
* 
* 
* 
* 

Enclave

“Can only fit 2 rows”

Output Table

* 
* 
* 
* 
* 
*
Oblivious SELECT

“Continuous” SELECT algorithm
Oblivious SELECT

“Continuous” SELECT algorithm
Oblivious SELECT

“Large” SELECT Algorithm

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

Enclave

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

Copy

Extra

Extra
Oblivious SELECT

“Large” SELECT Algorithm

| Input Table | * |
|            | * |
|            | * |
|            | * |
|            | * |
|            | * |
| Copy       | * |
| Enclave    | X |

Output Table

| * |
| X |
| * |
| * |
| * |
| X |

Delete

Dummy write
Oblivious Indexes
Tool: ORAM

Crypto primitive to generically hide access patterns to data

Security guarantee: two memory traces of the same length are indistinguishable

**Important:** does not automatically give obliviousness

Key question: how can we use ORAM to make indexes oblivious?

Goldreich and Ostrovsky, Software Protection and Simulation on Oblivious RAMs, 1993.
Oblivious Indexes: Considerations

- Naive composition of ORAM + Index NOT oblivious
- Generic solution: pad everything to maximum number of possible accesses
- How to do this without destroying performance?
  - Choice of index data structure (T tree, B tree, B+ tree, other?)
  - Make the worst case less bad (optimize for enclave/ORAM setting)
  - Small average-case improvements can be big worst-case improvements
Oblivious Indexes

Naive composition of ORAM and B+ Tree is not oblivious!

All data in leaves $\rightarrow$ ORAM ensures oblivious access ✓

Insert/Delete $\rightarrow$ number of operations depends on data ✗

First solution: pad all inserts/deletes to \textit{worst-case} number of ORAM accesses, but this is too slow.
Oblivious Indexes

Optimizations:

1. Cache nodes accessed during insertion/deletion inside enclave until certain they will not be accessed again

2. Remove parent pointers

3. Pad operation to worst-case number of operations, knowing we have made optimizations (1) and (2)

Point Read: $O(\log^2 N)$
Large Read: $O(N)$
Insertion: $O(\log^2 N)$
Deletion: $O(\log^2 N)$
Oblivious Indexes

Optimizations:

1. Cache nodes accessed during insertion/deletion inside enclave until certain they will not be accessed again
2. Remove parent pointers
3. Pad operation to worst-case number of operations, knowing we have made optimizations (1) and (2)

Point Read: $O(\log^2 N)$
Large Read: $O(N)$
Insertion: $O(\log^2 N)$
Deletion: $O(\log^2 N)$

Why not $O(N\log N)$?
Can do linear scan over ORAM data structure without using ORAM algorithm
Enables analytics on frequently updated table!
Performance
Design Validation

Choice of Storage Method

Effectiveness of Optimizer
## Design Validation

### Choice of Storage Method

<table>
<thead>
<tr>
<th>Workload Type</th>
<th>Best Storage Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>90% Insert, 5% Point/Large read</td>
<td>Combined</td>
</tr>
<tr>
<td>90% Small read, 9% Insert, 1% Delete</td>
<td>Index</td>
</tr>
<tr>
<td>50% Large read, 50% Point read</td>
<td>Combined</td>
</tr>
<tr>
<td>45% Point/Large read, 5% Insert/Delete</td>
<td>Combined</td>
</tr>
<tr>
<td>90% Large read, 5% Insert/Delete</td>
<td>Linear</td>
</tr>
</tbody>
</table>

Queries over 100k row table

### Effectiveness of Optimizer

<table>
<thead>
<tr>
<th>Query Selectivity</th>
<th>Alg. Choice</th>
</tr>
</thead>
<tbody>
<tr>
<td>5% of table, continuous</td>
<td>Small</td>
</tr>
<tr>
<td>5% of table, non-continuous</td>
<td>Small</td>
</tr>
<tr>
<td>95% of table, continuous</td>
<td>Continuous</td>
</tr>
<tr>
<td>95% of table, non-continuous</td>
<td>Large</td>
</tr>
</tbody>
</table>

Takeaway: Variety of storage methods and operator algorithms helpful for diverse workloads!
Comparison to Baseline

Performance vs baseline based on naive use of index/operators with ORAM

<table>
<thead>
<tr>
<th>Query Type</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range Selection (Linear)</td>
<td>29.2x</td>
</tr>
<tr>
<td>Group By Aggregate (Linear)</td>
<td>185x</td>
</tr>
<tr>
<td>Range Selection (Index)</td>
<td>1.4x</td>
</tr>
<tr>
<td>Point Selection (Index)</td>
<td>1.5x</td>
</tr>
<tr>
<td>Insert (Index)</td>
<td>64x</td>
</tr>
<tr>
<td>Delete (Index)</td>
<td>15x</td>
</tr>
</tbody>
</table>

Queries over Consumer Financial Protection Bureau dataset: ~107k rows
Comparison to HIRB + vORAM

7.6x faster for point query on 1M row table

HIRB Tree does not support range queries

Difference: enclave security guarantees

Roche et al, A Practical Oblivious Map Data Structure with Secure Deletion and History Independence, 2016.
Comparison to Opaque

Linear storage method:
comparable

Queries from Big Data Benchmark

Comparison to Opaque

Linear storage method: comparable

Combined storage method: comparable - 19x speedup

Analytics within 2.6x of Spark SQL

---

Comparison to Opaque

Linear storage method:
comparable

Combined storage method:
comparable - 19x speedup

Analytics within 2.6x of Spark SQL

Oblivious Index only:
<2x slowdown vs combined

Queries from Big Data Benchmark

Summary

**ObliDB**: Secure hardware enclave

+ new oblivious operator algorithms

+ multiple storage methods

= **fast oblivious performance on analytic AND transactional queries**


Source code available at https://github.com/SabaEskandarian/ObliDB
Extra Slides
Prior/Concurrent Oblivious Systems over SGX

**Opaque** [ZDBPGS17] (Prior): oblivious analytics, no support for indexes

**Oblix** [MPCCP18], **POSUP** [HOJY18]: oblivious indexes, but no operators over them

**StealthDB** [GVG17]: SGX database, no integrity or access pattern protection for index

**EnclaveDB** [PVC18]: SGX database, no access pattern protection (not oblivious)

**VeritasDB** [SC18]: integrity for key-value store over SGX

**ZeroTrace** [SGF17] (Prior): ORAM for oblivious key-value store over SGX

**ObliDB**: Obliviousness, Integrity, support for queries regardless of selectivity
Oblivious SELECT

“Hash” SELECT Algorithm

Goal: only one additional scan over data, regardless of query selectivity

Idea: Hash each input row to an output row

Obliviousness considerations:

- Hash based on row number, not contents

- Oblivious collision handling: average case $\rightarrow$ worst case

Asymptotically best strategy, but often outperformed by special cases
Tool: B+ Tree

Often used for indexes in databases

Generalization of binary search tree

All data in the leaves

Average-case insert/delete very fast

Worst-case insert/delete modifies tree at every level

Good for minimizing pointer traversals

Source: Wikimedia Commons
https://commons.wikimedia.org/wiki/File:Bplustree.png
Performance: Comparison to Sophos

Searchable symmetric encryption scheme without obliviousness

Supports only keyword lookups

Does not use hardware enclaves

22x speedup or more