Foundations of Differentially Oblivious Algorithms

Elaine Shi

Based on [CCMS’18] and [LSX’18]
Access patterns to even encrypted data leak sensitive information.
Access Pattern Attack: Computing on JPEG Image

Original

Recovered

Controlled-Channel Attacks [XCP’15]
Secure multi-party computation

Access Pattern Leakage in MPC

[Yao'82, GMW'87]
Oblivious RAM

An algorithmic approach that provably obfuscates access patterns
Oblivious RAM

Oblivious algorithm

→

Real-world addresses

→

Simulator

→

Simulated addresses
Oblivious RAM

“Encrypting the access patterns”
Oblivious RAM

“Encrypting the access patterns”

- Permute data in memory
- Shuffle data upon accesses
Any program can be made oblivious with $O(\log N)$ to $O(\log^2 N)$ overhead

[Optoroma, Circuit ORAM, ...]
ORAM State of the Art

Any program can be made oblivious with $O(\log N)$ to $O(\log^2 N)$ overhead

$\Omega(\log N)$ is necessary

[GO’96, LN’18]
ORAM State of the Art

Any program can be made oblivious with $O(\log N)$ to $O(\log^2 N)$ overhead.

Implicit assumption:

Runtime is fixed and known
Implicit assumption:

Runtime is fixed and known
• Must pad to worst-case runtime
• Can incur even *linear* overhead

Implicit assumption:

*Runtime is fixed and known*
Relax the obliviousness notion?

- Still provide meaningful privacy
- Significantly improve efficiency
Differential Obliviousness

Inspired by differential privacy [Dwork et al. 05]
Database

Algorithm (e.g., compaction, sorting)

randomized

Memory
Memory

Algorithm
(e.g., data analytics)

Database

Neighboring input DBs
Access patterns on neighboring DBs must be close

Memory

Algorithm (e.g., data analytics)

Database
Access patterns on neighboring DBs must be close

\[ \Pr[\downarrow \in S] \leq e^{\epsilon} \Pr[\downarrow \downarrow \in S] + \delta \]

This must hold for any S
(\epsilon, \delta)\text{-Differential Obliviousness}

Pr[\ Diamond \in S] \leq e^\epsilon Pr[\ Diamond \in S] + \delta

This must hold for any S
(\(\epsilon, \delta\))-Differential Obliviousness

1. What is being relaxed?
2. Still provide meaningful privacy?
3. Overcome obliviousness barriers?
(ε, δ)-Differential Obliviousness

1. What is being relaxed?
2. Still provide meaningful privacy?
3. Overcome obliviousness barriers?
What is being relaxed?

Closeness needs to hold only for neighboring DBs

\[ \Pr[\forall \in S] \leq e^\epsilon \Pr[\exists \in S] + \delta \]

This must hold for any S
What is being relaxed?

Closeness needs to hold only for neighboring DBs

Allow multiplicative, non-negligible loss

\[ \Pr[\forall \in S] \leq e^\epsilon \Pr[\exists \in S] + \delta \]

This must hold for any S
Does not require padding to worst-case runtime

\[ \Pr[\forall \in S] \leq e^\varepsilon \Pr[\exists \in S] + \delta \]

This must hold for any S
(\(\epsilon, \delta\))-Differential Obliviousness

1. What is being relaxed?
2. Still provide meaningful privacy?
3. Overcome obliviousness barriers?
Bad idea if you are protecting your Bitcoin signing key!

Still provide meaningful privacy?
When does DO make sense?

Secure CPU

Distributed data analytics
Typical parameters

\[ \Pr[\forall \in S] \leq e^\epsilon \Pr[\forall \in S] + \delta \]

This must hold for any \( S \)
(\(\epsilon, \delta\))-Differential Obliviousness

1. What is being relaxed?
2. Still provide meaningful privacy?
3. Overcome obliviousness barriers?
Stable Compaction

Obliviousness
\( \Omega(\mathbf{N} \log \mathbf{N}) \) necessary

Differential Obliviousness
\( O(\mathbf{N} \log \log \log \mathbf{N}) \)
Stable Compaction: Why do we care?

- Simple yet non-trivial
- Frequent algorithmic building block
- Warmup scheme in paper
Stable Compaction: **insecure** algorithm
Stable Compaction: insecure algorithm
Stable Compaction: insecure algorithm
Stable Compaction: **insecure** algorithm
Stable Compaction: insecure algorithm
Stable Compaction: **insecure** algorithm
Stable Compaction: insecure algorithm
Stable Compaction: insecure algorithm
Stable Compaction: insecure algorithm

Completes in $O(N)$ time

Leaks exact progress
Stable Compaction: oblivious algorithm
Stable Compaction: *oblivious* algorithm
Stable Compaction:

Takes $N \log N$ time

Sorting network
N \log N \text{ time is necessary for obliviousness}

Assumption: algorithm does not perform encoding on the kitties

Sorting network
Stable Compaction

Obliviousness

$\Omega(N \log N)$ necessary

Differential Obliviousness

$O(N \log \log \log N)$
Obliviousness

Cannot leak progress

Differential Obliviousness

Leak rough notion of progress
polylog(N) batch

2~5 kitties so far

DP oracle
polylog(N) batch

O-sort

DP oracle

2~5 kitties so far
polylog(N) batch

O-sort

polylog(N) error, DP estimate
5~8 kitties so far
polylog(N) error, DP estimate
Completes in $O(N \log \log \log N)$ time
Need:

**Oblivious and DP alg. that estimates all prefix sums, with polylog error**
Naive algorithm:

- Compute all $N$ prefix sums
- Add independent noise to each

All prefix sums -- DP and Oblivious
Naive algorithm:

- Compute all $N$ prefix sums
- Add independent noise to each

Incurs $\Theta(N)$ error
All prefix sums -- DP and Oblivious

[Dwork et al. 10, CSS’10]
Every node in the tree represents a range
- Every node in the tree represents a range
- Compute DP estimate for every node in the tree
• Every input appears in only $\log N$ nodes!
• Achieve only $\Theta(\log N)$ error per node!
Every prefix sum is the sum of $\log N$ nodes

Achieve poly $\log N$ error for each prefix sum
Summary: Leak rough notion of progress

Non-trivial combination of DP and oblivious algorithms

- Apply oblivious alg to small bins
- Make DP mechanisms oblivious
Putting it altogether

There exists an $O(N \log \log \log N)$ time, $(\Theta(1), \text{negl}(N))$-DO algorithm that realizes stable compaction.
There exists an $O(N \log \log \log N)$ time, $(\Theta(1), \text{negl}(N))$-DO algorithm that realizes stable compaction. Is this necessary?
\((\epsilon, 0)\) - Differentially Oblivious Stable Compaction:

\[ N \log N \] is necessary

even when \(\epsilon\) is arbitrarily large!
Other Results in Our Paper

- **lower bounds for obliviousness**
- **merging, range query DB**
  - Differentially oblivious algorithms with $O(\log \log N)$ blowup.
  - $\Omega(\log N)$ blowup necessary for full obliviousness.
Closely Related Works

[Wagh et al.] DP-ORAM, achieve $O(1)$ gain

[Kellaris et al.] DP for length, otherwise fully oblivious

[Mazloom et al.] DP access patterns for MPC
This is just a beginning.

Differential obliviousness for generic programs?
Composition?
Alternative notions?
Practical performance?
Differential obliviousness for generic programs?  
Composition?  
Alternative notions?  
Practical performance?  

Thank you!

elaine@cs.cornell.edu