Improving Oblivious RAM Protocol through Novel Eviction and Access Strategies

Akiva Gordon and Krishna Suraj
Mentor: Ling Ren

Overview

1. Background
   a. Definition of ORAM
   b. Previous ORAMs
2. Path ORAM II (Ring)
3. Future Directions
   a. Onion ORAM
   b. Optimization and Improvement

What is an ORAM?

- Oblivious Random Access Memory
- Trusted client, untrusted server

Desired Specifications:

- All accesses must be hidden
- Ideally a usable product with reasonable runtimes

Why is access pattern important?

- Information can be gained from data access pattern
    - frequently accessed files are considered more important
    - financial data, medical information

Background

a	b	c	d	e	f

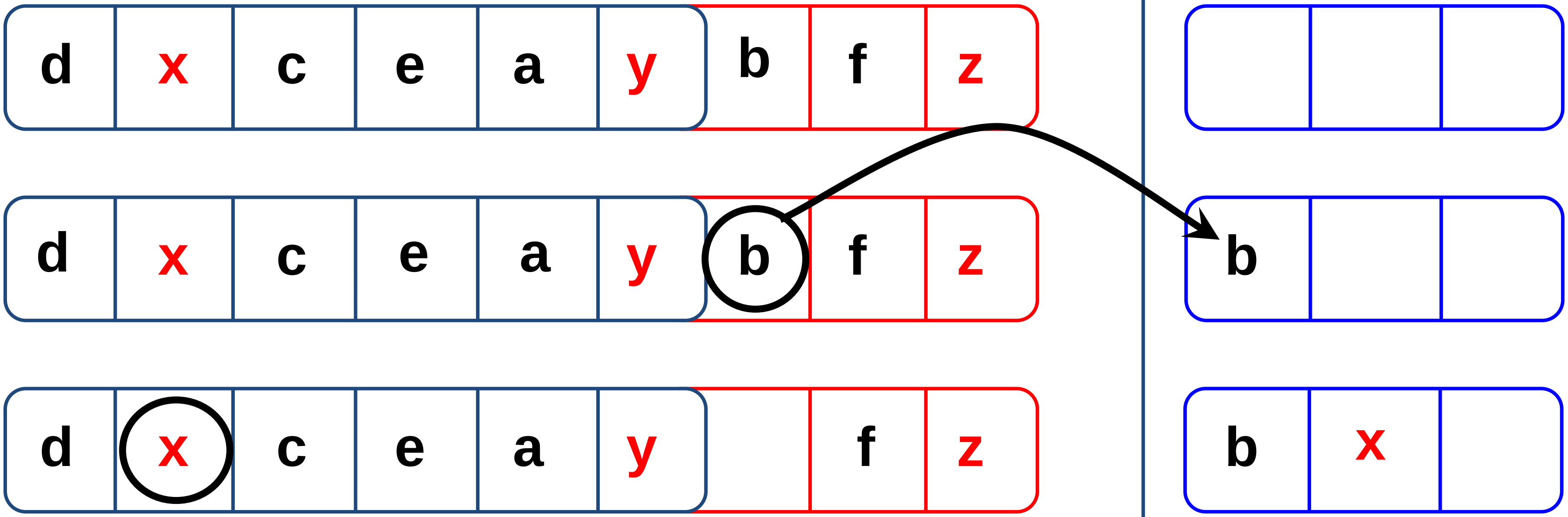
Encryption:

ɐ	q	ɔ	p	ǝ	ɟ

Goldreich 1987 ORAM

Server

Client

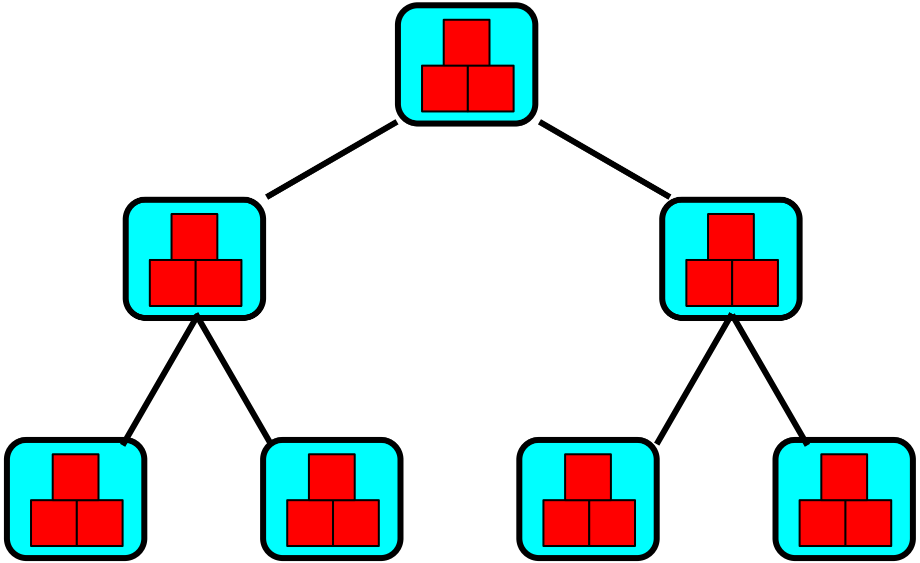


d	x	c	e	a	y	b	f	z			
d	x	c	e	a	y	b	f	z	b		
d	x	c	e	a	y		f	z	b	x	

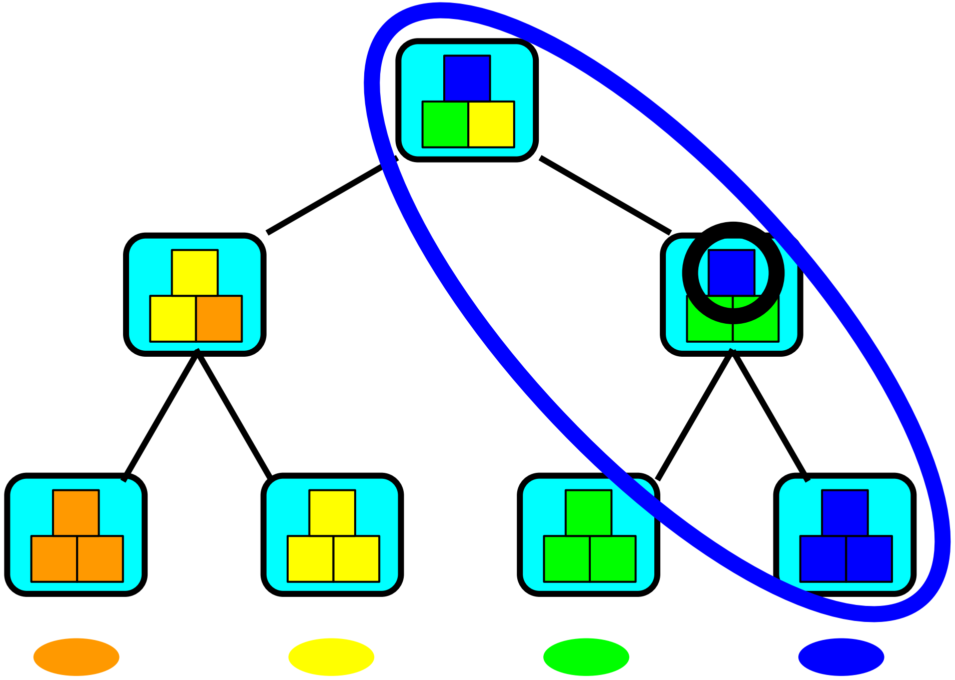
Problems with Goldreich Approach

- It’s still very inefficient - complexity $O(\sqrt{N})$
- Shuffling is also inefficient
- With large amounts of data, it’s virtually unusable

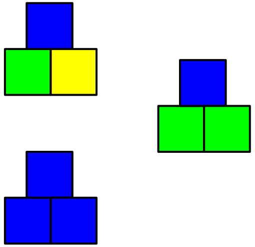
Path ORAM Overview



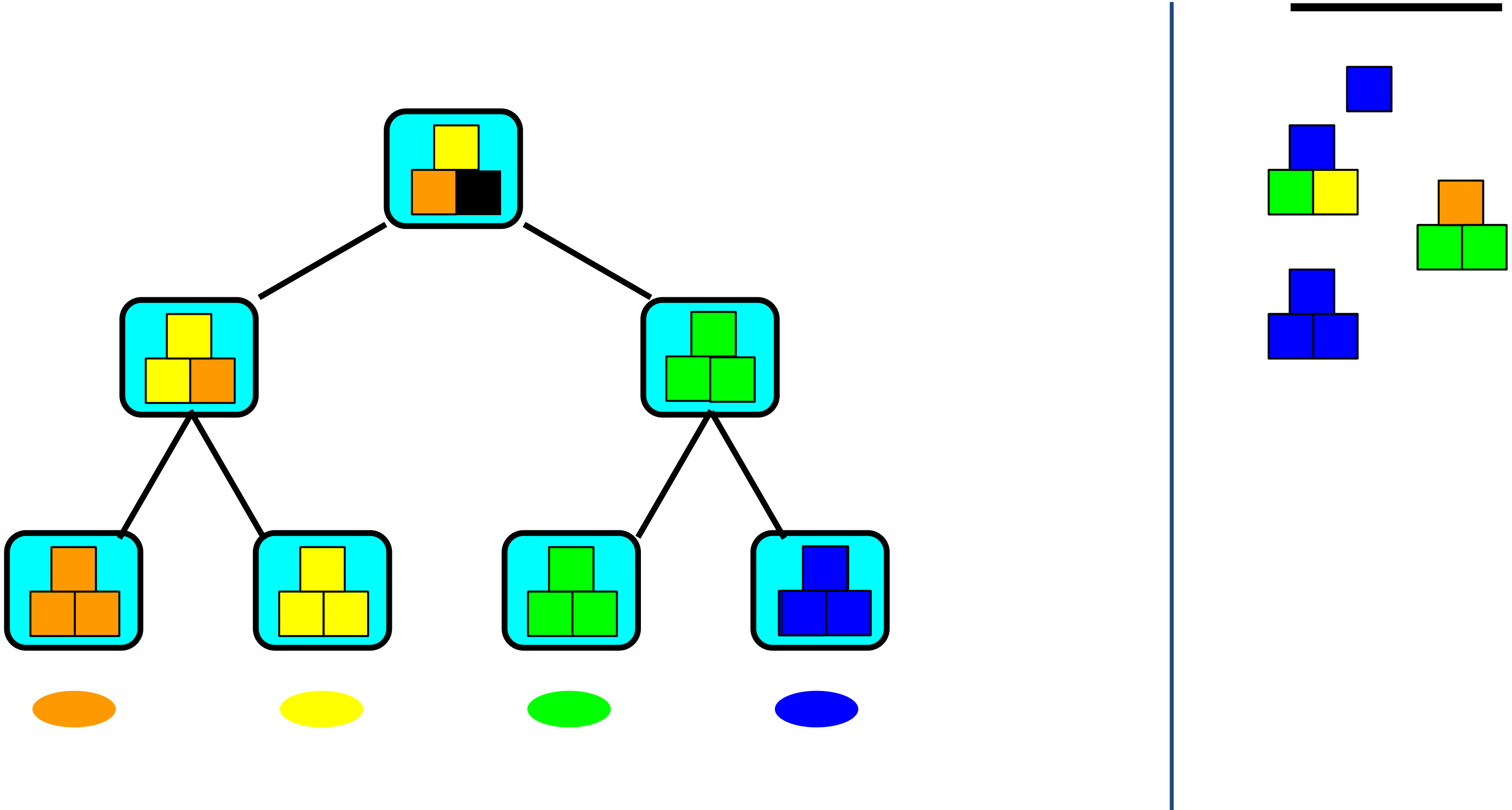
Path ORAM: Access



Stash



Path ORAM: Eviction



Path ORAM: Overall

- Much more efficient: O(log N)
- Still can be improved...

Path ORAM II: Ring ORAM

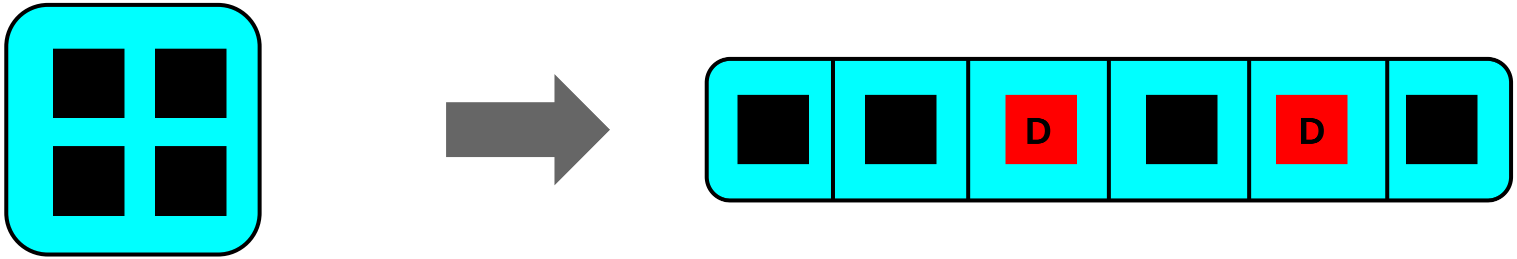
Ring ORAM: Overview

- Improvement on Path ORAM

- Improves by:
    - Decreasing bandwidth
    - Improve eviction quality

Ring ORAM: Buckets

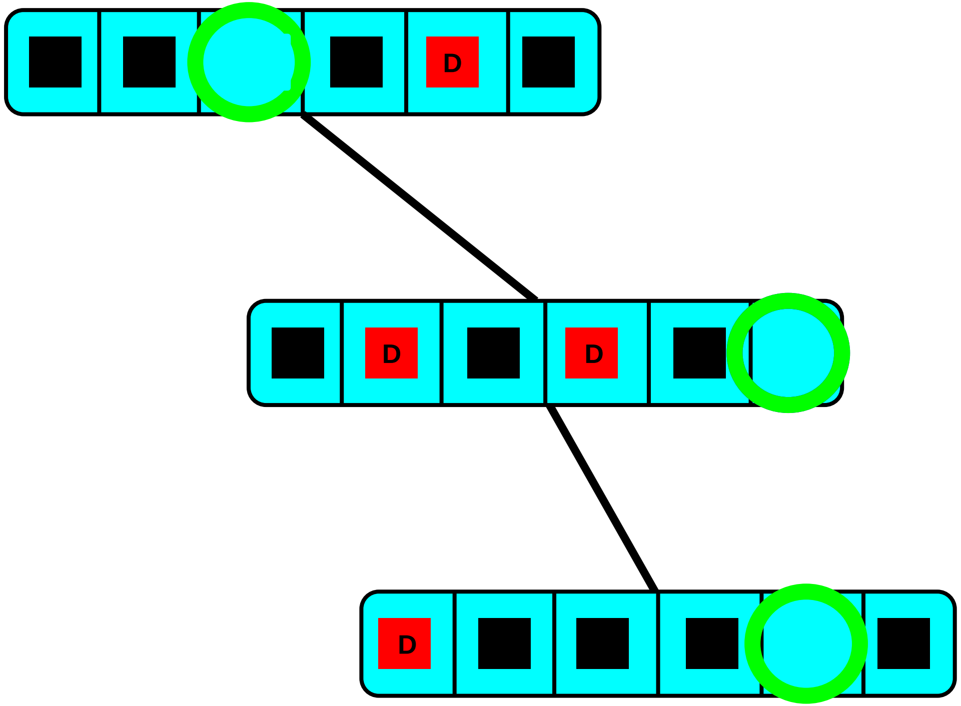
- Use Goldreich Approach:



Path ORAM Bucket

Ring ORAM Bucket

Ring ORAM: Access



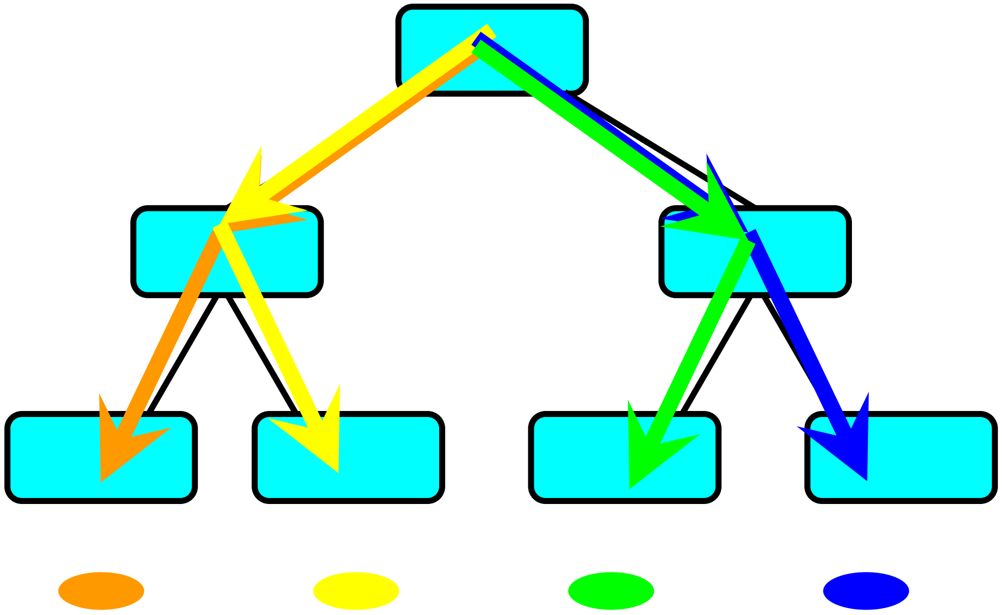
Stash

Ring ORAM: Eviction

Two Changes from Path ORAM:

- Only evict every Ath Access
- Evict along more efficient path

Optimized Eviction Paths



Our Ring ORAM Results

Z-value: 5

ORAM size: 127

Ring ORAM speed: 0.021916

Final Stash Size: 4

Table of Efficiencies

ORAM Protocol	**Bandwidth Efficiency**
Naive Linear Scan	O(N)
Goldreich (1987)	O(√N)
Path (2013)	O(lg N) (~8 lg N)
Ring (2014)	O(lg N) (~3 lg N)
?????	O(1)

FUTURE WORK

Onion Oram

Onion ORAM Details

- Breaks log N bound
- Server computation

Onion ORAM: Overview

- Server computes on encrypted data

- How?
  - Additive Homomorphic Encryption
  - Guaranteed progress of blocks

Onion ORAM protocol

E(0 1 0 0 0 0)

Data!

Onion ORAM layers

- Many layers of encryption
- Bounding layers is key
- Eviction - move all blocks to leaf

Onion ORAM efficiency

- Bandwidth cost: Constant order - O(b)

- Server Computation: O(B λ log N)

- Very Costly!

Optimizations and Improvements

- Onion ORAM multi-eviction
- Skipping layers in eviction phase
- NTRU vs Damgård-Jurik

Acknowledgements

- Our mentor, Ling Ren for his continuous help and guidance throughout the course of our research
- Professor Srini Devadas for his suggestion of our project and his assistance with our presentation
- Ethan Zou and Nathan Wolfe for Path ORAM code
- Everyone at MIT PRIMES for the opportunity to conduct world-class research
- Our parents for their support throughout the entire research process

Table of Efficiencies

ORAM Protocol	**Bandwidth Efficiency**
Naive Linear Scan	O(N)
Goldreich (1987)	O(√N)
Path (2013)	O(lg N) (~8 lg N)
Ring (2014)	O(lg N) (~3 lg N)
Onion (2015)	O(1) constant