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:

```
  t e d c b a
```
Goldreich 1987 ORAM

Server

Client
Problems with Goldreich Approach

● It’s still very inefficient - complexity O(√N)

● Shuffling is also inefficient

● With large amounts of data, it’s virtually unusable
Path ORAM Overview
Path ORAM: Access
Path ORAM: Eviction

Stash
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:
  o Decreasing bandwidth
  o Improve eviction quality
Ring ORAM: Buckets

- Use Goldreich Approach:
Ring ORAM: Access
Ring ORAM: Eviction

Two Changes from Path ORAM:

● Only evict every $A^{th}$ 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

<table>
<thead>
<tr>
<th>ORAM Protocol</th>
<th>Bandwidth Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Linear Scan</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Goldreich (1987)</td>
<td>$O(\sqrt{N})$</td>
</tr>
<tr>
<td>Path (2013)</td>
<td>$O(lg N) (~8 lg N)$</td>
</tr>
<tr>
<td>Ring (2014)</td>
<td>$O(lg N) (~3 lg N)$</td>
</tr>
<tr>
<td>????</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
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: $O(B \lambda \log N)$
- 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>
<thead>
<tr>
<th>ORAM Protocol</th>
<th>Bandwidth Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Linear Scan</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Goldreich (1987)</td>
<td>$O(\sqrt{N})$</td>
</tr>
<tr>
<td>Path (2013)</td>
<td>$O(lg N)$ ($\sim 8 lg N$)</td>
</tr>
<tr>
<td>Ring (2014)</td>
<td>$O(lg N)$ ($\sim 3 lg N$)</td>
</tr>
<tr>
<td>Onion (2015)</td>
<td>$O(1)$ constant</td>
</tr>
</tbody>
</table>