Onion ORAM: Constant Bandwidth ORAM Using Additively Homomorphic Encryption

Ling Ren

Joint work with: Chris Fletcher, Srini Devadas, Marten van Dijk, Elaine Shi, Daniel Wichs
Oblivious RAM (ORAM)

**Client**
limited storage, trusted

**Server**
ample storage, untrusted

\[ A = \{ \text{Read}(a_1), \text{Write}(a_2, d'), \ldots \} \]

\[ \text{ORAM}(A) = \{ \text{obfuscated } A \} \]

- **Read(a):** returns \( D[a] \);  \( \text{Write}(a, d') \): set \( D[a] = d' \)

- For client request sequences \( A \) and \( A' \) where \( |A| = |A'| \), \( \text{ORAM}(A) \) and \( \text{ORAM}(A') \) are indistinguishable
ORAM Applications

- Software protection [G’87]
- Secure processors
- Outsourced storage
- Secure computation [GKKMRV’12, LWNHS’15]
- Garbled RAM [LO’12, GHRW’14]
- Proofs of retrievability [CKW’13]
Current Art and Where to Go Next

• State of the art schemes
  – Bandwidth blowup: \(O(\log N)\) \(N = \#\) blocks
  – Client storage: \(O(1)\)
  – Server storage: \(O(N)\)

• Can we do better in bandwidth blowup?
  – Goldreich-Ostrovsky \(\Omega(\log N)\) lower bound [GO96]
  – Bound doesn’t hold if server can do computation!

• Lots of ORAMs have already used server computation
  – [SSS’12, WS’12, DSS’14, MBC’14, AKST’14]

Still \(\Omega(\log N)\) bandwidth (except using FHE [AKST’14])
Our Result: Onion ORAM

ORAM with O(1) bandwidth blowup, O(1) client storage, O(N) server storage

poly-log server computation, of additive-HE

This talk: Semi-honest can be extended by malicious
Mini Roadmap

1. Start with a basic ORAM

2. Add server computation to improve BW

3. Run into problems

4. Solve the problems
Tree-based ORAM

- N blocks, logN levels, each node has Z slots

**Client**

- Will come for free

  - Table: map each block to a path
  - red block $\rightarrow$ path 1

**Server**

- Invariant:
  - if a block is mapped to a path, it must be on that path
1. Lookup map, red block → path 1
2. Read all blocks on path 1 (in blue)
3. Remap red block to a new path (remove from path 1)
4. Add red block back to root
Tree-based ORAM: Eviction

- Read a random path (say path 2)
- Push each block as far down as possible on that path
- Can prove negligible overflow if $Z > 4$

Client

Map:
- red block $\rightarrow$ path 3
- purple block $\rightarrow$ path 2

Server

Recurse
Towards O(1) Bandwidth - Access

- **Access:** want one block from a path

- **Private Information Retrieval**
  - Select $y_i$ from $Y \in \{y_1, y_2, \ldots, y_m\}$ without revealing $i$
  - Client sends: $\{E(x_1), E(x_2), \ldots, E(x_m)\}$ under additive-HE
    
    \[
    x_j = 1 \text{ if } j = i, \quad x_j = 0 \text{ otherwise}
    \]
  - Server computes:
    
    \[
    y_1 E(x_1) + y_2 E(x_2) + \cdots + y_m E(x_m) = E(y_1 x_1 + y_2 x_2 + \cdots + y_m x_m) = E(y_i)
    \]
    
    \[
    \bigoplus_{j \in [Y]} E(x_j) \otimes y_j = E(y_i)
    \]
    
    \[
    \bigoplus_{j \in [Y]} E'(x_j) \otimes E(y_j) = E'(E(y_i))
    \]
Block Size Independence

- Select between $Y$ and $Y'$: $E(0) \otimes Y \oplus E(1) \otimes Y'$
- Problem: $\sim E(0), E(1)$ are as big as $Y, Y'$
- Break $Y, Y'$ into *chunks*
- Apply same select bit to each chunk

- Bandwidth: $B + o(B)$
**ORAM Access + Additive-HE**

Assume each block encrypted with 1 layer

**Client**

- Decrypt metadata
- Generate \( \pi = E(0), E(0), \ldots E(1), \ldots E(0) \)
- \( \pi, E(\text{updated metadata on path}) \)

**Server**

- Compute \( E(E(d)) = \text{Select}(\pi, \text{blocks\_on\_path}) \)
- \( E(E(d)) \)
- \( E(\text{updated metadata}) \)

- Decrypt \( d = D(D(E(E(d)))) \)
- Update \( d \rightarrow d' \)
- \( E(d') \)

- Append(root, \( E(d') \))
Assume each block encrypted with 1 layer

ORAM Access + Additive-HE

Client

Server

path

 Decrypt metadata

E(metadata on path)

O(1) bandwidth on accesses

when block size \geq \text{select vector size} 
\geq L \times Z \text{ chunks}

Decrypt d = D(D(E(E(d))))

Update d \rightarrow d'

E(d')

Append(root, E(d'))
Towards O(1) Bandwidth - Eviction

Blocks may get stuck where they are and keep acquiring layers in evictions!
Towards $O(1)$ Bandwidth - Eviction

Problem with all Tree-based ORAMs!

Blocks may get stuck where they are and keep acquiring layers in evictions!
Design our ORAM eviction algorithm such that all blocks involved are guaranteed to move down

Goal for Eviction

Select($\pi_C$, B, A, C)
Design our ORAM eviction algorithm such that all blocks involved are guaranteed to move down

1. Evict down path, one bucket triplet at a time
2. push all blocks from the parent to its 2 children
3. Set params s.t. Pr[child overflows] = negl

Example: evict to path 6

Empty!
Reverse lexicographic order

- For any non-leaf bucket: eviction paths alternates between its two children
Layer Bound

• Reverse lexicographic order

Key property: Green bucket is written to twice before being emptied

Theorem: buckets at level $k < L$ have $\leq 2k + 1$ layers

• With optimizations: $k + 1$
Avoid Overflow

New parameter $A$ : 1 eviction per $A$ accesses

**Theorem:** $Z \geq A$, $N \leq A \times 2^{L-1}$

$\Rightarrow \text{Pr[overflow]} = e^{-\frac{(2Z-A)^2}{6A}}$

- $Z = A = \theta(\lambda) \Rightarrow \text{Pr[overflow]} = 2^{-\theta(\lambda)}$

- $\text{negl}(N)$ overflow: $Z = A = \log N \omega(1)$
  $\Rightarrow \text{Pr[overflow]} = N^{-\omega(1)}$
O(1) Bandwidth on Eviction

Client

Server

eviction path known by server

Compute $\Pi = \{\pi_0 ... \pi_{Z*L}\}$

(|$\pi_i$| = $O(Z)$ encrypted coefficients)

$\Pi, E$(updated metadata on path)
O(1) Bandwidth on Eviction

Client

Server

eviction path known by server

E(metadata on path)

O(1) bandwidth evictions

when block size $\geq$ total select coeff size
$\geq Z^2 \times L / A$ chunks
$\geq Z \times L$ chunks
Eviction post-processing

Problem: layers in leaves are not bounded

- At end of each eviction, client “manually” “refreshes” a leaf bucket

Client: $E(E\ldots(E(\text{blocks in leaf})))$  
Server: $E(\ldots E(\text{blocks in leaf}))$

- Layer theorem now applies to leaves as well
Problem: layers in leaves are not bounded

- At end of each eviction, client “manually” “refreshes” a leaf bucket

  Adds $O(1)$ bandwidth
  When $A = Z$

- Layer theorem now applies to leaves as well
The Encryption Scheme

- **Layer bound** = $O(\log N)$

- **Damgård-Jurik (2001) cryptosystem:**
  - $\mod n^s \rightarrow \mod n^{s+1}$ \hspace{1cm} ($n =$ RSA modulus)
  - $s|n| \rightarrow (s + 1)|n|

- **Strategy:** set $s_0 = O(\log N)$
  - $|n| O(\log N) \rightarrow |n| O(\log N)$ \hspace{1cm} (chunk size)

- **Block size > ZL chunks**
  - $\Omega(|n| \log^2 N \log^2 \lambda)$ \hspace{1cm} w/ some optimizations

- **Subsequent work [MMB’15]: block size** = $\Omega(|n|\log^2 \lambda)$
  - Works with any AHE (no onions)
The Malicious Case

1. Start with the semi-honest Construction

2. Cut-and-choose verification chunks

3. ECC to amplify detection probability
Idea 1: Verification chunks

- $\Theta(\lambda)$ random verification chunks for each block
- Normal memory checking to verification chunks
- Server: $\text{Select}(\pi, \ldots \text{all chunks} \ldots)$
- Client: $\text{Select}(\pi, \ldots \text{verification chunks} \ldots)$
- Adds $O(1)$ bandwidth when block size $\geq \text{sum(v. chunks)}$
Idea 2: Error correcting code

• Problem: adversary tampers with one chunk
• \( \Pr[\text{chooses non-v. chunk}] > \text{negligible} \)

• Solution:
  – Encode block using ECC e.g., Reed-Soloman
  – Decode failure: adversary tampers with \( \geq O(1) \) fraction (e.g., \( \Delta = \frac{1}{4} \))
  – Block blowup is constant (e.g., 2x)

• Client gets wrong data or fails ECC
  \( \rightarrow \) adversary tampers with \( \geq \Delta \) fraction of chunks
• \( \Pr[\text{none are v. chunks} | \text{tamper } \geq \Delta \text{ fraction}] = (1 - \Delta)^\lambda \)
Conclusion

- **Onion ORAM**
  - A fine-tuned ORAM eviction algorithm to ensure progress
  - $O(1)$ bandwidth
  - $O(1)$ client storage, $O(N)$ server storage
  - polylog server computation
  - Damgård-Jurik additive-HE (extended to any additive-HE)
  - No FHE (can be instantiated with SWHE)

- Secure in the malicious setting without resorting to SNARKs

Thank you!
Backup
Bandwidth vs. block size

Multiplier for O(\text{Block size}) bandwidth

- **Limit = 2**

**Damgård-Jurik cryptosystem**
- Modulus = 2048 bits
- Pr[bucket overflow] = 2^{-80}
Citations

- [GO’96] Software Protection and Simulation on Oblivious RAMs
- [SSS’12] Towards Practical Oblivious RAM
- [LO’12] How to Garble RAM Programs
- [WS’12] Single round access privacy on outsourced storage
- [CKW’13] Dynamic Proofs of Retrievability via Oblivious RAM
- [DSS’14] Burst ORAM: Minimizing oram response times for bursty access patterns
- [MBC’14] Efficient private le retrieval by combining oram and pir
- [AKST’14] Verifiable oblivious storage
ORAM + FHE

• Folklore in the community: $O(1)$ bandwidth ORAM is trivially achievable with FHE

• Need FHE with
  – $O(1)$ ciphertext expansion
  – Efficient bootstrapping

• Can we do it without FHE?
Additively Homomorphic Encryption (AHE) basics

- \( E(x) \oplus E(x') = E(x + x') \)
- \( Y \otimes E(x) = E(x) \oplus E(x) \oplus \cdots \oplus E(x) = E(Yx) \)

Private Information Retrieval (PIR) using AHE

- Goal: server stores \( \{Y_1, Y_2, \ldots, Y_m\} \), user selects \( Y_{i^*} \) without revealing \( i^* \)
- User sends \( \{E(x_1), E(x_2), \ldots, E(x_m)\} \) where \( x_{i^*}=1 \) and other \( x_i=0 \)
- Server evaluates

\[
Y_1 \otimes E(x_1) \oplus Y_2 \otimes E(x_2) \oplus \cdots \oplus Y_m \otimes E(x_m)
\]

\[
= E(Y_1x_1 + Y_2x_2 + \cdots + Y_m x_m) = E(Y_{i^*})
\]