Onion ORAM: Constant Bandwidth ORAM with Server Computation

Chris Fletcher

Joint work with:
Ling Ren, Marten van Dijk, Srini Devadas
• State of the art schemes
  – Bandwidth: $O(\log N)$
  – Client storage: $O(1)$ (Path ORAM = $O(\log N)$)
  – Server storage: $O(N)$

• Is “optimal” ORAM possible?
  $O(1)$ bandwidth, $O(1)$ client storage, $O(N)$ server storage

• Goldreich-Ostrovsky lower bound [1987, 1996]
  Given a program that runs in $T$ time and an $N$ block ORAM with $O(1)$ client storage, the program+ORAM must run in $\Omega(T \log N)$ time

• $\Omega(T \log N)$ doesn’t mean $\Omega(T \log N)$ bandwidth!
Example: Outsourced storage (Honest but curious)

- "Read X, Y, Z, return F(X, Y, Z)"
- Message stream must be oblivious

Client

Server

Storage (ORAM tree)

(insecure) Processor

Arbitrary messages

limited storage, compute

Trust boundary
• XORing reads [Dautrich et al.], PIR+ORAM [Mayberry et al.]

• XOR + Ring ORAM
  – Permuted buckets $\rightarrow$ one real block touched / read
  
  – $B, d1, d2, d3, \ldots$
  – $E(B, r), E(0, r1), E(0, r2), E(0, r3) \ldots$

  – **Server sends:** $E(B, r) \oplus E(0, r1) \oplus E(0, r2) \oplus E(0, r3) \oplus \ldots$
  – **Client computes:** $E(0, r1) \oplus E(0, r2) \oplus E(0, r3) \oplus \ldots$

• Both schemes make read bandwidth $O(\log N) \rightarrow O(1)$
• Does not help on evictions!
Can we make evictions $O(1)$ Bandwidth?
Example: Ring ORAM

- ORAM on server is encrypted under FHE scheme $E^{\text{FHE}}$
- Reads

\[
\begin{align*}
\text{Client} & \quad \text{Server} \\
\text{leaf, } E^{\text{FHE}}(\text{address}) & \quad E^{\text{FHE}}(d) = \text{Select}(E^{\text{FHE}}(\text{address}), \ \text{Path}(\text{leaf})) \\
\text{RemoveBlock}(E^{\text{FHE}}(\text{address}), \ \text{Path}(\text{leaf})) & \\
E^{\text{FHE}}(d) & \\
\text{- Evictions} & \\
\text{Path}(\text{leaf}_g)' = \text{EvictPath}(\text{Path}(\text{leaf}_g))
\end{align*}
\]

Read bandwidth is $O(1)$, no bandwidth for evictions!
Path(leaf_g)’ = EvictPath(Path(leaf_g))

m_A = metadata for block A

\[ \pi_i = \text{MoveBlock}(m_A \ldots m_F) \]

Block C can get stuck

- Only Select() touches blocks
- Server computation: polylog(N)
- Bootstrap to manage noise [Apon et al., Mayberry et al.]
*Discuss later: Does the previous scheme achieve optimal Bandwidth/storage?

Do we need bootstrapping?
Do we need FHE?
Do we need FHE?

- **Additive-HE (e.g., Paillier)**
  - Addition: \( E^{AHE}(a) \oplus E^{AHE}(b) = E^{AHE}(a + b) \)
  - Scalar multiplication: \( E^{AHE}(a) \otimes c = E^{AHE}(ca) \)

- **Select from \((X, Y)\):**
  \[
  E^{AHE}(0) \otimes X \oplus E^{AHE}(1) \otimes Y = E^{AHE}(0 + Y) = E^{AHE}(Y)
  \]

- **\(Y = E^{AHE}(\text{plaintext})\)**
- **Select op \(\rightarrow E^{AHE}(E^{AHE}(\text{plaintext}))\)**
  - Client decrypts twice
  - (Possible) ciphertext blowup per layer

- **Layers(output) = \( max( \text{Layers(Block}_i) : \text{Blocks} ) + 1 \)**
• ORAM encrypted using 1 layer of $E^{AHE}$ (abbreviated E)

**Client**

- Decrypt metadata
- Compute $\pi = E(0), E(0), \ldots, E(1), \ldots, E(0)$
- Compute $E(E(d)) = \text{Select}(\pi, \text{Path(leaf)})$

**Server**

- Write updated metadata
- Compute $E(E(d)) = \text{Select}(\pi, \text{Path(leaf)})$
- Update $d \rightarrow d'$
- $E(d')$
- $\text{Path(leaf)[root].append}(E(d'))$
• Problem: Continuous reshuffling $\rightarrow$ Unbounded layers

• Reason: Blocks can get stuck in buckets after evictions

$$\text{Layers(output)} = \max(\text{Layers(Block}_i) : \text{Blocks}) + 1$$

$A_1 = \text{Block A with 1 layer}$

$O(T)$ evictions $\rightarrow$ Slot with $C$ gets $O(T)$ layers
ORAM with $O(1)$ bandwidth, $O(1)$ client storage, $O(N)$ server storage

...with only additive-HE
Design our ORAM eviction algorithm such that buckets are guaranteed to be empty regularly.

\[ \text{Select}(\pi_C, B, A, C) \]
Design our ORAM eviction algorithm such that buckets are guaranteed to be empty regularly

1. Evict over reverse-lexicographic order of paths
2. Also evict to sibling buckets
3. Set \( Z, A \) s.t. \( \Pr[\text{bucket overflow}] = \text{negl}(\text{security parameter}) \)
4. Evict to 1 bucket triplet at a time

Example:
evict to leaf 6

Informal guarantee:
Max layers = \( O(\log N) \)
Theorem: $Z \geq A$, $N \leq A \times 2^{L-1}$

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

- $Z = A = \theta(\log N)\omega(1) \implies \Pr[\text{bucket overflow}] = N^{-\omega(1)}$

*Note: $N = \text{poly(\text{security parameter})}$*

- **Asymptotics w/o server computation**
  - Bandwidth = $O(\log^2 N)\omega(1)$ blocks
  - Client storage = $O(\log N)\omega(1)$ blocks
  - Server storage = $O(N)$ blocks

- **Not competitive w/o server computation**
• Same as previous proposal
  – Client sends leaf
  – Server sends metadata
  – Client sends $\pi = E(0), E(0), \ldots E(1), \ldots E(0)$
  – Server sends block

• Simple scheme factoring in layers
  – Elements of $\pi$ have 1 layer
  – Pad blocks on path to $S = \text{Max}( |\text{Block}_i| : \text{Blocks} )$ bits
  – Split each padded block into $C$ chunks s.t. $S / C = \text{Plaintext}(\pi_i) = P$
Eviction Terminology

Path(leaf)[i] = $i^{th}$ triplet on path
Path(leaf)[i].dest[j] = $j^{th}$ block in $i^{th}$ triplet’s dest. bucket
Layer Analysis

• Useful properties:
  1. At eviction start: non-leaf sibling buckets are empty
  2. At eviction end: non-leaf destination buckets are empty

Leaves:
- Blocks get stuck in the leaves
- Non-leaves empty at regular intervals
Layer Analysis

• Analyze: Layers on destination bucket at start of select

Key intuition: destination bucket was sibling on last eviction

# Layers?

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

• $c$ is constant, $c = 1$ in our final scheme
Client

\[ \text{leaf}_g \text{ (eviction path) known by server} \]

\[ E(\text{metadata for Path(leaf}_g)) \]

Server

\[ \Pi = \{\pi_0 \ldots \pi_{Z*L}\} \]

\(|\pi_i| = O(Z) \text{ encrypted coefficients} \)

\[ \Pi, E(\text{updated metadata for Path(leaf}_g)) \]

For triplet \( i \):

\[ \text{Path(leaf}_g)[i].\text{src} = \text{Path(leaf}_g)[i].\text{dst} \]

For slot \( j \):

\[ \text{args} = \{\text{Path(leaf}_g)[i].\text{dst}[j], \text{Path(leaf}_g)[i].\text{src}\} \]

\[ \text{Path(leaf}_g)[i].\text{dst}[j] = \text{Select}(\pi_{Z*i+j}, \text{args}) \]
Problem: layers in leaves are not bounded

- At end of each eviction...

  \[ E(d_j) = \text{Path}(\text{leaf}_g)[L].\text{dst}[j] \]

  Decrypt \( E(d_j) \) to a constant number of layers, yielding \( e_j \)
  Compute \( E(e_j) \)

- Layer theorem now applies to all levels
- Adds constant amortized bandwidth if \( Z \sim A \)
Setting parameters
Problem: each layer can add ciphertext blowup

- Layer bound = $O(\log N)$
- Paillier (1999): $n \rightarrow n^2$ (n = RSA modulus)
- Damgård-Jurik (2001): $n^s \rightarrow n^{s+1}$
  - $s$ = free parameter
  - Strategy: set $s_0 = \log N$, log N layers $\rightarrow n^{s_0+\log N} = n^{O(\log N)}$

- Operations are like Paillier:
  \[
  E(a) \oplus E(b) = E(a)E(b) \quad E(a) \otimes b = E(a)^b
  \]

- Best attack: factor $n$, complexity $\exp(|n|^{1/3}(\log |n|)^{2/3})$
  $\therefore |n| = \Theta(\log^3 N) \rightarrow$ defeat attacks w/ complexity $N^{\omega(1)}$
So far ... Select = $\bigoplus_{i} \pi_i \otimes Block_i$  

"trivial linear PIR"

- Each select adds 1 layer
  layer bound = $\log N$
- $Z$ inputs $\rightarrow |\pi| = Z \times$ layer bound $\times |n| = \log^5 N \omega(1)$

Hierarchical PIR [Lipmaa 2005]

- Multiplexer tree
- $Z$ inputs $\rightarrow |\pi| = \log Z$ coefficients
  $\rightarrow$ select adds $\log Z$ layers
- $\therefore$ layer bound$' = \log N \log \log N$
- $Z$ inputs $\rightarrow |\pi| = \log Z \times$ layer bound$' \times |n|$
  $= \log^4 N \log^2 \log N$

At least better in theory 😊
Parameterization

- **Strategy:** set $|\Pi| = |\{\pi_0 \ldots \pi_{Z*L}\}| = O(B)$
- I.e., $\Pi$ contributes constant (amortized) bandwidth

- **Let** $Z = A = \log N \omega(1)$
- $|\pi_{read}| = |n| \times \log^2 N \omega(1)$ \hspace{1cm} (n = RSA modulus)
- $|\pi_{evict}| = |n| \times \log N \log^2 \log N$ \hspace{1cm} (mux tree)
- $|\Pi_{evict}| = |n| \times \log^2 N \log^2 \log N = \log^5 N \log^2 \log N$

- **Final asymptotics:**
  - Block size $B = \Omega(\log^5 N \log^2 \log N)$
  - Bandwidth = $O(B)$
  - Client storage = $O(B)$
  - Server storage = $O(BN)$
Ongoing/Future work

• **Decrease block size** $B = \Omega(k \cdot \log^2 N \cdot \log^2 \log N)$
  – Modern schemes w/o computation: $B = O(\log^2 N)$

• **How?**
  – Server computation is $O(\log^2 N)\omega(1)$ blocks - is $O(\log N)$ possible?
  – Is there a suitable additive-HE scheme with $k = o(\log^3 N)$?

• **Protect against malicious servers**
  – Server performs select incorrectly

• **Improve Garbled RAM schemes?**
  – Use ORAM as a blackbox

• **Parameterization for SWHE for Onion ORAM**
  – No bootstrapping needed