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

• XOR + Ring ORAM
  – Permuted buckets → one real block touched / read

  – $B$, $d_1$, $d_2$, $d_3$, …
  – $E(B, r)$, $E(0, r_1)$, $E(0, r_2)$, $E(0, r_3)$ …

  – Server sends: $E(B, r) \oplus E(0, r_1) \oplus E(0, r_2) \oplus E(0, r_3) \oplus …$
  – Client computes: $E(0, r_1) \oplus E(0, r_2) \oplus E(0, r_3) \oplus …$

• Both schemes make read bandwidth $O(\log N) \rightarrow O(1)$

• Does not help on evictions!
Example: Ring ORAM

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

---

**Client**

leaf, $E^{\text{FHE}}(\text{address})$

---

**Server**

$E^{\text{FHE}}(d) = \text{Select}(E^{\text{FHE}}(\text{address}), \text{Path}(\text{leaf}))$

RemoveBlock($E^{\text{FHE}}(\text{address}), \text{Path}(\text{leaf}))$

$E^{\text{FHE}}(d)$

---

- Evictions

Path(leaf$_g$)$' = \text{EvictPath}(\text{Path}(\text{leaf}_g))$

---

Read bandwidth is $O(1)$, no bandwidth for evictions!
Eviction circuit for FHE

\[
\text{Path}(\text{leaf}_g)' = \text{EvictPath}(\text{Path}(\text{leaf}_g))
\]

- \(m_A = \text{metadata for block A}\)
- \(\pi_i = \text{MoveBlock}(m_A \ldots m_F)\)
- Select(\(\pi_C, m_A, m_B, m_C\) \(\rightarrow \) Select(\(\pi_C, A, B, C\))
- Select(\(\pi_D, m_A, m_B, m_D\) \(\rightarrow \) Select(\(\pi_D, A, B, D\))
- Select(\(\pi_E, m_A, m_B, m_E\) \(\rightarrow \) Select(\(\pi_E, A, B, C, D, E\))
- Select(\(\pi_F, m_A, m_B, m_F\) \(\rightarrow \) Select(\(\pi_F, A, B, C, D, F\))

- Only \text{Select()} touches \textit{blocks}
- \(\smiley\) Server computation: \text{polylog}(N)
- \(\sad\) 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?
• 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) = \( max( \text{Layers}(\text{Block}_i) : \text{Blocks} ) + 1 \)
• ORAM encrypted using 1 layer of $E^{AHE}$ (abbreviated E)

**Client**

```
Client

Decrypt metadata
Compute $\pi = E(0), E(0), \ldots, E(1), \ldots, E(0)$

\[ \pi, E(\text{updated metadata for Path(leaf)}) \]
```

**Server**

```
Server

Write updated metadata
Compute $E(E(d)) = \text{Select}(\pi, \text{Path(leaf)})$

\[ E(E(d)) \]
```

```
Block i

\[ \pi \downarrow i \pi \downarrow i \pi \downarrow i \pi \downarrow i \pi \downarrow i \]
```

```

Decryption

\[ d = E(E(d)) \]

Update $d \rightarrow d'$
```

```
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 $= \mathcal{O}(\log N)$
Theorem: $Z \geq A, \ N \leq A \times 2^{L-1}$

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

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

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

• **Asymptotics w/o server computation**
  - Bandwidth = $O(\log N) \omega(1)$ 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. $S / C = \text{Plaintext}(\pi_{\downarrow i}) = P$

Assume layers $\rightarrow$ ciphertext blowup
Eviction Terminology

Source

Destination

Triplet 0

Sibling

Path(leaf)[ i ] =
Path(leaf)[ i ].dest[ j ] =

\( i^{th} \) triplet on path
\( j^{th} \) 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
• Analyze: Layers on destination bucket at start of select

Key intuition: destination bucket was sibling on last eviction

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

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

Client

Server

leaf_g (eviction path) known by server

Compute Π = {π_0 ... π_{Z*L}}

(|π_i| = O(Z) encrypted coefficients)

Π, E(updated metadata for Path(leaf_g))

Onion ORAM evict w/ Additive-HE

For triplet i:

Path(leaf_g)[i].sibling = Path(leaf_g)[i].src

For slot j:

args = {Path(leaf_g)[i].dst[j] , Path(leaf_g)[i].src}

Path(leaf_g)[i].dst[j] = Select(π_{Z*i+j} , args)
Problem: layers in leaves are not bounded

- At end of each eviction...
  
  **Client**

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

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

  Compute $E(e_j)$

  **Server**

  For $j = 0 \ldots Z - 1$

- 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): \(n \rightarrow n^{12}\) (\(n = RSA\) modulus)
- Damgård-Jurik (2001): \(n^{s} \rightarrow n^{s+1}\)
  - \(s\) = free parameter
  - Strategy: set \(s \downarrow 0 = \log N\), \(\log N\) layers \(\rightarrow\) \(n^{s0} + \log N = n^{\Omega(\log N)}\)
- Operations are like Paillier:
  \[E(a) \oplus E(b) = E(a)E(b)\]
  \[E(a) \otimes b = E(a)^{b}\]
- Best attack: factor \(n\), complexity \(\text{exp}(\log |n|^{1/3} (\log |n|))^{2/3}\)
  \[\therefore |n| = \Theta(\log^{3} N) \rightarrow \text{defeat attacks w/ complexity} N^{\omega(1)}\]
Optimization: Hierarchical PIR

- So far … Select = $\bigoplus_{i} \pi \downarrow i \otimes \text{Block} \downarrow i$ “trivial linear PIR”

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

- Hierarchical PIR [Lipmaa 2005]
  - Multiplexer tree
  - $Z$ inputs $\rightarrow |\pi| = \log Z$ coefficients
    - $\rightarrow$ select adds $\log Z$ layers
  - $\therefore$ layer bound $\uparrow' = \log \log \log N$
  - $Z$ inputs $\rightarrow |\pi| = \log Z \times \text{layer bound} \uparrow' \times |n|$
    - $= \log \uparrow 4 N \ \log \uparrow 2 \log N$

At least better in theory 😊
Parameterization

• Strategy: set \(|\Pi| = |\{\pi \downarrow 0 \ldots \pi \downarrow Z \ast L\}| = O(B)\)
  
• i.e., \(\Pi\) contributes constant (amortized) bandwidth

• Let \(Z = A = \log N \omega(1)\)

• \(|\pi \downarrow \text{read}| = |n| \ast \log \tau 2 N \omega(1)\) \quad (n = RSA modulus)

• \(|\pi \downarrow \text{evict}| = |n| \ast \log N \log \tau 2 \log N\) \quad (mux tree)

• \(|\Pi \downarrow \text{evict}| = |n| \ast \log \tau 2 N \log \tau 2 \log N = \log \tau 5 N \log \tau 2 \log N\)

• Final asymptotics:
  
  – Block size \(B = \Omega(\log \tau 5 N \log \tau 2 \log N)\)
  
  – Bandwidth = \(O(B)\)

  – Client storage = \(O(B)\)

  – Server storage = \(O(BN)\)
Ongoing/Future work

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

• 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