Oblivious RAM
Classical Results and Recent Developments

Marios Georgiou
2nd Exam

Committee:
Nelly Fazio (Mentor)
Rosario Gennaro
William Skeith
ORAM Definition

- \( \langle s, EM \rangle \leftarrow \text{Setup}(M, \bot) \)
- \( \langle (s', M[i]), EM' \rangle \leftarrow \text{Access}(\langle (s, i, v), EM \rangle) \)

Correctness
Returns what a regular RAM would return

Security
Let \( y = ((i_1, v_1), \ldots, (i_n, v_n)) \) be a sequence of accesses
Let \( A(y) \) be the access pattern of \( y \)
For any initial memory and for any \( y_1, y_2 \) such that \(|y_1| = |y_2|\)
\[ A(y_1) \sim_c A(y_2) \]
ORAM with Server Computation
aka: Verifiable Oblivious Storage (VOS)

• In Classical ORAM we can only read from and write to specific locations
• In VOS the Server/RAM can perform computations to improve efficiency
  • e.g. Homomorphic Encryption, Evaluation of Garbled Circuits
  • Semi-Honest vs Malicious Adversary
Main Complexity Measures

• Local Storage
• Remote Storage
• Communication Overhead
• Round Complexity
• in VOS: Server Computation
• in PRAM: Slowdown
For a memory of $n$ blocks, assuming Local Storage $O(1)$, we need communication overhead $\Omega(\log n)$
Goldreich – Ostrovsky Lower Bound [GO96]

Applies only to ORAM without Server Computation

For a memory of $n$ blocks, assuming Local Storage $O(1)$, we need communication overhead $\Omega(\log n)$
This Survey

- Main Construction Techniques
  - Hierarchical ORAM [GO96]
  - Tree-based ORAM [SCSL11]
- Results
  - Classical ORAM
  - ORAM with Server Computation
- Some recent constructions
- Open Problems
  - (Proof of Lower Bound)
Hierarchical ORAM [GO96]

Buffers $B_i$ of $2^i$ buckets with $i \in [k, \log n]$
Buckets of constant size $b$
$B_i$ associated with random hash function $h_i$
Block $M[j]$ is stored as $\text{Enc}(j, M[j])$ in bucket $h_i(j)$
Hierarchical ORAM [GO96]

Buffers $B_i$ of $2^i$ buckets with $i \in [k, \log n]$
- Each bucket of constant size $b$
- Associated with random hash function $h_i$
- Block $M[j]$ is stored as $\text{Enc}(j, M[j])$ in bucket $h_i(j)$

$h_{\log n}(3) = 11$
Hierarchical ORAM [GO96]

Buffers $B_i$ of $2^i$ buckets with $i \in [k, \log n]$

- Each buffer of constant size $b$
- Associated with random hash function $h_i$
- Block $M_j$ is stored as $\text{Enc}(j, M[j])$ in bucket $h_i(j)$

Read $(j)$: $\forall i$, Bucket. Read($B_i[h_i(j)]$)

Write $(j, v)$: Bucket. Write($B_1[h_1(j)]$, $\text{Enc}(j, v)$)
Hierarchical ORAM [GO96]

Buffers $B_i$ of $2^i$ buckets with $i \in [k, \log n]$
- Each bucket of constant size $b$
- Associated with random hash function $h_i$
- Block $M[j]$ is stored as $\text{Enc}(j, M[j])$ in bucket $h_i(j)$

Eviction:
Every $2^i$ accesses, empty buffer $i$ into buffer $i + 1$ and pick new $h_{i+1}$

$h_{\log n}(3) = 11$

Read ($j$): $\forall i$, Bucket. Read ($B_i[h_i(j)]$)
Write ($j, v$): Bucket. Write ($B_1[h_1(j)], \text{Enc}(j, v)$)
Cuckoo Hashing [PR01]

Buffer $B$ is associated with two hash functions $h_1, h_2$

A block $j$ can live either in $B[h_1(j)]$ or in $B[h_2(j)]$

Insertion of $j$ is always on $B[h_1(j)]$

If occupied, kick out previous block and put it in its other position

\begin{tabular}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\hline
7 & 4 & 1 & 8 & 2 & 3 & & \\
\end{tabular}

\[ h_1(5) = 6 = h_2(2) \]  
\[ h_1(2) = 2 \]
Tree-based ORAM [SCSL11]

Block is associated to a path
Nodes are buckets hosting \( \leq b \) blocks
Block lives in a bucket along its path

Read\((j)\): \( \forall \) buckets \( b \in \text{Path}[j], \text{Bucket. Read}(b) \)
Write\((j, v)\): \( \text{Bucket. Write( root, Enc(j, v))} \)
Tree-based ORAM [SCSL11]

Block is associated to a path. Nodes are buckets hosting \( \leq b \) blocks. Block lives in a bucket along its path.

\[ \text{Path: Stored Locally} \]

\begin{array}{cccccccc}
1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
101 & 001 & 111 & 100 & 010 & 101 & 110 & 001 \\
\end{array}

Read\((j)\): \( \forall \) buckets \( b \in \text{Path}[j] \), Bucket. Read\((b)\)

Write\((j, v)\): Bucket. Write\((\text{root, Enc}(j, v))\)
Tree-based ORAM [SCSL11]

Eviction:
Every $Z$ accesses, pick a path and push its elements down

Block is associated to a path
Nodes are buckets hosting $\leq b$ blocks
Block lives in a bucket along its path

Path: Stored Locally

<table>
<thead>
<tr>
<th>Path: Stored Locally</th>
</tr>
</thead>
<tbody>
<tr>
<td>1  2  3  4  5  6  7  8</td>
</tr>
<tr>
<td>101 001 111 100 010 101 110 001</td>
</tr>
</tbody>
</table>

$Read(j)$: $\forall$ buckets $b \in \text{Path}[j]$, Bucket. $Read(b)$

$Write(j, v)$: Bucket. $Write(root, Enc(j, v))$
Eviction: Every $Z$ accesses, pick a path and push its elements down.

Block lives in a bucket along its path. Nodes are buckets hosting $\leq b$ blocks. This saves a constant factor. Apply recursively $\log n$ times.

Block is associated to a path.

Path: Stored Locally

Read($j$): $\forall$ buckets $b \in$ Path[$j$], Bucket. Read($b$)
Write($j, v$): Bucket. Write(root, Enc($j, v$))
## Results on Classical ORAM

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Type</th>
<th>Block size</th>
<th>Client Storage</th>
<th>Worst Case Communication overhead</th>
<th>Amortized Communication overhead</th>
</tr>
</thead>
<tbody>
<tr>
<td>[GO96]</td>
<td>H</td>
<td>$\Omega(\log n)$</td>
<td>$O(1)$</td>
<td>$O(n \log n)$</td>
<td>$O(\log^3 n)$</td>
</tr>
<tr>
<td>[GMOT12]</td>
<td>H</td>
<td>$\Omega(\log n)$</td>
<td>$O(1)$</td>
<td>$O(\log^2 n)$</td>
<td>$0(\log n)$</td>
</tr>
<tr>
<td>[KLO12]</td>
<td>H</td>
<td>$\Omega(\log n)$</td>
<td>$O(1)$</td>
<td>$O(\log^2 n/ \log \log n)$</td>
<td>$O(\log^2 n/ \log \log n)$</td>
</tr>
<tr>
<td>[SCSL11]</td>
<td>T</td>
<td>$\Omega(\log n)$</td>
<td>$O(1)$</td>
<td>$O(\log^3 n)$</td>
<td>$O(\log^3 n)$</td>
</tr>
<tr>
<td>[SVDS+13]</td>
<td>T</td>
<td>$\Omega(\log^2 n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>[SVDS+13]</td>
<td>T</td>
<td>$\Omega(\log n)$</td>
<td>$O(\log n)$</td>
<td>$O(\log^2 n)$</td>
<td>$O(\log^2 n)$</td>
</tr>
<tr>
<td>[WCS15]</td>
<td>T</td>
<td>$\Omega(\log^2 n)$</td>
<td>$O(1)$</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
## Results on ORAM w/ Server Computation

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Block Size</th>
<th>Communication Overhead</th>
<th>Server Computation</th>
<th>Assumption</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>[AKST14]</td>
<td>$\text{polylog}(n)$</td>
<td>$O(1)$</td>
<td>$\text{polylog}(n)$</td>
<td>FHE + bootstrapping</td>
<td>SH</td>
</tr>
<tr>
<td>[MMB15]</td>
<td>$\Omega(\log^4 n)$</td>
<td>$O(1)$</td>
<td>$O(B\lambda \log n)$</td>
<td>LWE</td>
<td>M</td>
</tr>
<tr>
<td>[DvDF+16]</td>
<td>$\Omega(\log^6 n)$</td>
<td>$O(1)$</td>
<td>$O(B\lambda \log n)$</td>
<td>Add. HE</td>
<td>M</td>
</tr>
</tbody>
</table>

$B = \text{Block Size}$

$\lambda = \text{security parameter}$
Onion ORAM [DvDF+16]
Onion ORAM [DvDF+16]

Read($j$):
1. Retrieve $j$’s exact location by scanning its path and looking only at the addresses
2. Create the vector $v \in \{0,1\}^{b(L+1)}$, where $v_{i^*} = 1$ and $v_{i \neq i^*} = 0$, with $i^*$ the exact position of $j$
3. Send $\langle\text{Enc}(v_1), ..., \text{Enc}(v_{b(L+1)})\rangle$
4. Server computes
   \[
   \bigoplus_i (\text{Enc}(v_i) \otimes \text{ct}_i) = \text{Enc} \left( \sum_i v_i \cdot \text{ct}_i \right) = \text{Enc}^{\ell+1}(\text{pt}_i)
   \]
5. Re-encrypt the addresses and put block in root
Onion ORAM [DvDF+16]

Read($j$):
1. Retrieve $j$’s exact location by scanning its path and looking only at the addresses
2. Create the vector $v \in \{0,1\}^{b(L+1)}$, where $v_{i^*} = 1$ and $v_{i \neq i^*} = 0$, with $i^*$ the exact position of $j$
3. Send $\langle \text{Enc}(v_1), \ldots, \text{Enc}(v_{b(L+1)}) \rangle$
4. Server computes
   $$\bigoplus_i (\text{Enc}(v_i) \otimes \text{ct}_i) = \text{Enc} \left( \sum_i v_i \cdot \text{ct}_i \right) = \text{Enc}^{\ell+1}(\text{pt}_i)$$
5. Re-encrypt the addresses and put block in root
Path ORAM [SVDS+13]

\[ y_L \overset{\text{def}}{=} y \]
\[ y_i \overset{\text{def}}{=} \left\lfloor \frac{y_{i+1}}{2} \right\rfloor \]
\[ b_i \overset{\text{def}}{=} I(y_i \text{ is even}) \]
\[ x_1 \ldots x_L \leftarrow (x_1 \ldots x_{2L})_0 \]
\[ x_{L+1} \ldots x_{2L} \leftarrow (x_1 \ldots x_{2L})_1 \]

\[ y = 4 = 13 \quad b_4 = 0 \]
\[ y_3 = 7 \quad b_3 = 0 \]
\[ y_2 = 4 \quad b_2 = 1 \]
\[ y_1 = 2 \quad b_1 = 1 \]
Path ORAM [SVDS+13]

Client wants $x_3 = A_2[y_2]_{b_2}$

Client wants $x_2 = A_1[y_1]_{b_1}$

Client wants $x_1 = A_3[y_3]$
Garbled Circuit

\[(\tilde{C}, \text{lab}) \leftarrow \text{Setup}(C)\]
\[y \leftarrow \text{Eval}(\tilde{C}, \text{lab}_x)\]

Correctness
\[y = C(x)\]

Security
\[\forall A \exists \text{Sim}:\]

A \[\overset{C}{\rightarrow} E\]
\[\overset{\tilde{C}}{\leftarrow} (\tilde{C}, \text{lab}) \leftarrow \text{Setup}(C)\]
\[\overset{x}{\rightarrow} (\tilde{C}, \text{lab}_x)\]

A \[\overset{C}{\rightarrow} E\]
\[\overset{\tilde{C}}{\leftarrow} (\tilde{C}, \sigma) \leftarrow \text{Sim}\]
\[\overset{x}{\rightarrow} \text{lab}_x \leftarrow \text{Sim}(\sigma)\]
\[\downarrow\]
\[(\tilde{C}, \text{lab}_x)\]
TWORAM [GMP15]
TWORAM [GMR15]

Each node = Garbled Circuit
Hardcoded blocks

Garbled input

left, right or root
next garbled input
TWORAM: Circuit to be garbled

Input: cState = (p, y, π)
Hardcoded: u, bucket, leftInputs, rightInputs, nState
Output: Next node, garbled input

if π = ⊥
    π ← ExtractBucket(bucket, y)
if u is not a leaf
    Based on p return (left, leftInputs(p,y,π)) or (right, rightInputs(p,y,π))
else
    if u is in last tree
        return π
    else
        return (nextroot, nState(π,y,⊥))

p: leaf of current path
y: wanted location in A_L
π: carries the leaf of the next tree
u: current node
Parallel ORAM [BCP16]

Classical ORAM

Every access incurs update of local state and modification of memory
Parallel ORAM [BCP16]

Classical ORAM

Every access incurs update of local state and modification of memory

If $\geq 2$ CPUs want the same block
Parallel ORAM [BCP16]

Lookup: CPUs choose a representative to read the block and broadcast it. Rest: Dummy reads
Put-back: Pick a random leaf $\ell$ and communicate the block to the CPU associated tree with $\ell$
Flush: Each node picks a path. Conflicting nodes are taken care by a representative
Possible Directions

• Untrusted CPUs (not only RAM)
• Similarly to MIPs: Introduction of more RAMs that do not communicate
  • Maybe improves efficiency or complexity assumptions
• Server that does not have enough storage so it uses another server