MACAO: A Maliciously-Secure and Client-Efficient Active ORAM Framework

Thang Hoang†, Jorge Guajardo‡, Attila A. Yavuz†

†CSE, University of South Florida
hoangm@mail.usf.edu, attilaayavuz@usf.edu

‡Robert Bosch LLC – RTC
Jorge.GuajardoMerchan@us.bosch.com
Oblivious RAM

- Oblivious Random Access Machine (ORAM) allows a client to hide the access pattern when accessing data stored on untrusted memory.

**ORAM applications:** Cloud storage-as-a-service (personal data storage, health-record database, password management), searchable encryption, secure multiparty computation
ORAM was first introduced by Goldreich for software protection.

Recent attempts focused on reducing ORAM communication overhead.

- **Square-root ORAM** (STOC) $O(\sqrt{N})$ comm.
- **Partition ORAM** (NDSS) $O(\log N)$ comm. $O(\sqrt{N})$ client storage
- **Tree-ORAM** (AsiaCrypt) $O(1)$ client storage $O(\log^2 N)$ comm.
- **Path-PIR** (NDSS) $O(\log N)$ comm.
- **Apon et al.** (PKC) FHE $O(1)$ comm.
- **Path-ORAM** (CCS) $O(\log N)$ comm. $O(\lambda)$ client storage
- **Multi-cloud ObliviStore** (CCS) $O(\sqrt{N})$ client storage $O(1)$ comm.
- **Hierarchical ORAM** (JACM) $O(\log^3 N)$ comm.
- **ORAM lower bound** (JACM)
- **Circuit-ORAM** (CCS) $O(\log N)$ comm.
- **C-ORAM** (CCS) Insecure
- **Bucket-ORAM** FHE
- **S^3ORAM** (CCS) $O(1)$ comm. $O(1)$ client storage semi-honest security
- **Ring-Onion ORAM** (CCS) $O(1)$ comm. $O(\log N)$ client storage semi-honest security
- **Passive ORAM lower bound** (Crypto)
- **2-server ORAM** (Asiacrypt) $O(N)$ computation
Tree-ORAM Paradigm [SCSL11]

- Binary tree data structure
- Block data located somewhere in the tree path
- Empty nodes are filled with dummy data

General Access Protocol

1. Get pID of A: 1
2. **Retrieve** path of A
3. Update A (if needed)
4. Randomly select new path for A: 4
5. **Evict**

<table>
<thead>
<tr>
<th>Client</th>
<th>Stash</th>
<th>Position map</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Block</td>
</tr>
<tr>
<td></td>
<td></td>
<td>A</td>
</tr>
<tr>
<td></td>
<td></td>
<td>pID</td>
</tr>
</tbody>
</table>
PIR-based ORAM: Malicious Security Concern

- Due to unit vectors created in retrieval phase
  - Contain only one element 1, while others are 0
  - Malicious adversary can tamper with the blocks corresponding to elements “0”
    - Computation result is still correct → cannot be detected by client
    - Learn real block positions
    - Access pattern leakage
MACAO Framework

- Based on (authenticated) additive secret sharing [DPSZ11]

- $x \in F_p$ is authenticated shared if each party $P_i$ has random values $x_i, a_i, m_i \in F_p$ s.t.

\[
x = \sum_i x_i
\]

\[
\alpha = \sum_i a_i
\]

\[
\alpha x = \sum_i m_i
\]

- Authenticated share of $x$ is denoted as $\langle x \rangle = ([x], [\alpha x])$

Any linear function of shared values can be computed locally

- Given constants $v_1, v_2$ and shared values $[x], [y]$

\[
v_1 \cdot [x] + v_2 \cdot [y] = [v_1 x + v_2 y] = [z]
\]
MACAO Framework

Harness Circuit-ORAM eviction [WCS15] and permutation matrix [HOY+17] principles

- \( O(1) \) client bandwidth overhead
- Bucket size \( Z = O(1) \)
- Each eviction takes a block from the stash and writes it back to the tree

Circuit-ORAM Eviction Principle:
- Only scan once from root to leaf
- For each level, pick or drop (at most) 1 block
- At any time, can only hold (at most) 1 block

Create \((H + 1)\) permutation matrices \(I_h\) sized \((Z + 1) \times (Z + 1)\) s.t.

- \(I_h[j, Z + 1] = 1\) : Pick the block at index \(j\)
- \(I_h[1, j] = 1\) : Drop the holding block to index \(j\)
- \(I_h[1, Z + 1] = 1\) : Move the holding block to next level \(h + 1\)
- \(I_h[j + 1, j] = 1\) : Keep the block at index \(j\) remain

\[ 7 \]
MACAO Framework

Two main schemes

- $\Pi_{rss}$
  - Replicated secret sharing (RSS)
  - 3-server setting with honest majority

- $\Pi_{spdz}$
  - SPDZ secret sharing
  - General $\ell$-server setting with dishonest majority
**MACAO Framework - \( \Pi_{\text{rss}} \) scheme**

**Retrieval**
- Select query \( q = (0, \ldots, 1, \ldots, 0)^{H+1} \)

1. **XOR-PIR**: a pair of PIR queries \( (q_i^{(1)}, q_i^{(2)}) \) per authenticated share \( \langle T \rangle_i \)
   - \( q_i^{(1)} \leftarrow \{0,1\}^{H+1}, q_i^{(2)} \leftarrow q \oplus q_i^{(1)} \)

\[
\begin{align*}
\langle B \rangle_1 & \leftarrow R_1^{(1)} \oplus R_1^{(2)} \\
\langle B \rangle_0 & \leftarrow R_0^{(1)} \oplus R_0^{(2)} \\
\langle B \rangle_2 & \leftarrow R_2^{(1)} \oplus R_2^{(2)}
\end{align*}
\]

\( (X, Y) \leftarrow \langle B \rangle_0 + \langle B \rangle_1 + \langle B \rangle_2 \)

Check if \( \alpha X =? Y \)
MACAO Framework - $\Pi_{\text{rss}}$ scheme

**Retrieval**

- Select query $q = (0, \ldots, 1, \ldots, 0)^{H+1}$

2. **RSS-PIR**: two RSS queries $(q_i, q_{i+1})$ per server $S_i$
   - $q_0 + q_1 + q_2 = q$, where $q_i \leftarrow \mathbb{F}_p^{H+1}$

$$(X, Y) \leftarrow \langle R \rangle_0 + \langle R \rangle_1 + \langle R \rangle_2$$

Check if $\alpha X = ? Y$
MACAO Framework - $\Pi_{\text{rss}}$ scheme

**Eviction:** based on RSS-based matrix multiplication protocol

- RSS-share of evicting block $B$ and $(H + 1)$ RSS-shares of permutation matrices $M_h$

\[
M_h = [M_h]_0 \cdot [M_h]_1 \cdot [M_h]_2
\]
\[
B = [B]_0 \cdot [B]_1 \cdot [B]_2
\]

Jointly execute MACCheck($\langle T\rangle_h$) to verify eviction integrity

$\text{RSSMatMult}([U], [V])$

- $X_i \leftarrow [U]_i \times [V]_i + [U]_{i+1} \times [V]_i + [U]_i \times [V]_{i+1}$
- $S_i$ sends $\left( R_{i-1}^{(1)}, R_{i-1}^{(0)} \right)$ to $S_{i-1}$, $\left( R_{i+1}^{(1)}, R_{i+1}^{(0)} \right)$ to $S_{i+1}$, where $X_i = \sum_{j=0}^{i} R_j^{(i)}$

Output:

\[
\begin{align*}
& [U \times V]_i \leftarrow R_i^{(0)} + R_i^{(1)} + R_i^{(2)} \\
& [U \times V]_{i+1} \leftarrow R_{i+1}^{(0)} + R_{i+1}^{(1)} + R_{i+1}^{(2)}
\end{align*}
\]

$\text{MACCheck}([T])$

- $x \leftarrow \sum_h \sum_i \sum_j r_T [T[i,j]]_h$
- $y \leftarrow \sum_h \sum_i \sum_j r_T [\alpha T[i,j]]_h$
- Pass if $\alpha \cdot x = ? y$

(Random linear combination)

$T_h$: holding block and current blocks at level $h$
MACAO Framework - $\Pi_{\text{spdz}}$ scheme

Both retrieval and eviction are based on SPDZ-based authenticated matrix multiplication protocol

- **Retrieval:** Select query $\langle q \rangle = (\langle 0 \rangle, \ldots, \langle 1 \rangle, \ldots, \langle 0 \rangle)^{H+1}$

- **Eviction:** SPDZ-share of evicting block $B$ and $(H + 1)$ SPDZ-shares of permutation matrices $M_h$

![Diagram]

**SPDZMatMult((U), (V))**

Initialization: Each $S_i$ has $\langle A_i \rangle$, $\langle B_i \rangle$, $\langle C_i \rangle$, authenticated shares of Beaver triples ($C = A \times B$, $\alpha C = \alpha (A \times B)$)

- $[E]_i \leftarrow [U]_i - [A]_i$, $[P]_i \leftarrow [V]_i - [B]_i$
- Open $E$ and $P$
- MACCheck($E$) and MACCheck($P$)

Output: $[u \times v]_i \leftarrow [c]_i + e \times [b]_i + [a]_i \times p + e \times p$
$[\alpha u \times v]_i \leftarrow [\alpha c]_i + e \times [b]_i + [a]_i \times p + [\alpha]_i \times e \times p$

**MACCheck($T$, $[\alpha T]$)**

- $x \leftarrow \sum_i \sum_j r^i T[i, j]$
- $y \leftarrow \sum_i \sum_j r^i [\alpha T[i, j]]$
- Pass if $\alpha \cdot x = y$

(Random linear combination)
MACAO Framework - Extension

- **Bandwidth Reduction**
  - Pseudo-random function (PRF) to generate additive shares locally \[\text{[CDI05, DSZ14, RWTS+17]}\]

- **Client Storage Reduction**
  - Stash sized $O(\log N)$ was stored at the client (due to Circuit-ORAM eviction)
  - Two ways to reduce client stash storage
    1. **Store stash at the server-side**, and use Private-Information Writing (PIW) to privately put the block into the stash
    2. **Triplet Eviction** \[\text{[SvDFR+16]}\]
      - Stash not needed in place of $O(\log N)$ bucket size
MACAO Framework – Performance (1/3)

- MACAO schemes were $7 \times$ faster than single-server ORAMs and up to $1.5 \times$ slower than $S^3$ORAM

**Figure 13:** End-to-end delay of MACAO schemes and their counterparts.

**Configuration:** Library: NTL, tomcrypt, zeroMQ, pthread; Client: Macbook Pro 2018; Servers: Amazon EC2 c5.4xlarge, EBS-based storage; Client-server bandwidth: 29/5 Mbps; Inter-server bandwidth: 250/250 Mbps; DB Size: 1GB – 1TB; Block size: 4KB, 256KB
MACAO Framework – Performance (2/3)

- Server computation contributed the most portion to the overall delay
- Bandwidth reduction trick significantly reduced the communication costs

**Figure 14**

Cost breakdown of MACAO schemes

**Configuration:** Library: NTL, tomcrypt, zeroMQ, pthread; Client: Macbook Pro 2018; Servers: Amazon EC2 c5.4xlarge, EBS-based storage; Client-server bandwidth: 29/5 Mbps; Inter-server bandwidth: 250/250 Mbps; DB Size: 1GB – 1TB; Block size: 4KB, 256KB
MACAO Framework – Performance (3/3)

- Bandwidth reduction trick also helped to reduce the delay when increasing number of servers for higher privacy levels

![Diagram showing end-to-end delay with varied privacy levels]

End-to-end delay with varied privacy levels

**Configuration:**
- Library: NTL, tomcrypt, zeroMQ, pthread
- Client: Macbook Pro 2018
- Servers: Amazon EC2 c5.4xlarge, EBS-based storage
- Client-server bandwidth: 29/5 Mbps
- Inter-server bandwidth: 250/250 Mbps
- DB Size: 1GB – 1TB
- Block size: 4KB, 256KB
Conclusion & Future Work

- Proposed MACAO, a multi-server active ORAM framework providing integrity, access pattern obliviousness against active adversaries, and secure computation capability.
- Based on Authenticated additive secret sharing and tree ORAM paradigm

Future Work

- Oblivious Distributed File System (ODFS) implementation
- Multi-user Oblivious Storage based on MACAO

The proposed ODFS Model
Thank you for your attention!

MACAO code: https://github.com/thanghoang/MACAO
References


Our Motivation

- Single-server active ORAM (e.g., Onion-ORAM) offers $O(1)$ bandwidth blowup and malicious security
  - High computation overhead due to Homomorphic Encryption (HE)
  - Cut-and-choose technique → incurs higher communication and computation overhead for malicious security
- Multi-server active ORAM (i.e., $S^3$ORAM) offers $O(1)$ bandwidth with efficient computation
  - However, it only offers semi-honest security

An efficient multi-server ORAM with active security?
**S^3ORAM [HOY+17]**

- Tree-ORAM paradigm
- Exploit the efficiency of multi-party computation in distributed setting
  - Shamir Secret Sharing (SSS) Scheme
  - **Retrieval:** SSS-Private Information Retrieval – **Eviction:** Permutation Matrix

**S^3ORAM System Model:**
- $\ell \geq 2t + 1$ servers
- # colluding servers $\leq t$
- All servers are **semi-honest**
MACAO Framework – Summary

Asymptotic comparison of state-of-the-art ORAM schemes.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Bandwidth Client-server</th>
<th>Overhead† Server-server</th>
<th>Block Size*</th>
<th>Client Block Storage‡</th>
<th># servers§</th>
<th>Security</th>
<th>Comp. over Enc. Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ring-ORAM [53]</td>
<td>(O(\log N))</td>
<td>-</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>1</td>
<td>Semi-Honest</td>
<td>×</td>
</tr>
<tr>
<td>CKN+18 [16]</td>
<td>(O(\log N))</td>
<td>-</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>3</td>
<td>Semi-Honest</td>
<td>×</td>
</tr>
<tr>
<td>GKW18 [32]</td>
<td>(O(\log N))</td>
<td>-</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>2</td>
<td>Semi-Honest</td>
<td>×</td>
</tr>
<tr>
<td>S³ORAM [33]</td>
<td>(O(1))</td>
<td>(O(\log N))</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>2(t + 1)</td>
<td>Semi-Honest</td>
<td>✓</td>
</tr>
<tr>
<td>Path-ORAM [64]</td>
<td>(O(\log N))</td>
<td>-</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>1</td>
<td>Malicious</td>
<td>×</td>
</tr>
<tr>
<td>Circuit-ORAM [66]</td>
<td>(O(\log N))</td>
<td>-</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>1</td>
<td>Malicious</td>
<td>×</td>
</tr>
<tr>
<td>SS13 [61]</td>
<td>(O(1))</td>
<td>(O(\log N))</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>2</td>
<td>Malicious</td>
<td>×</td>
</tr>
<tr>
<td>LO13 [42]</td>
<td>(O(\log N))</td>
<td>-</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>2</td>
<td>Malicious</td>
<td>×</td>
</tr>
<tr>
<td>Onion-ORAM [22]</td>
<td>(O(1))</td>
<td>-</td>
<td>(\Omega(1))</td>
<td>(O(\log N))</td>
<td>1</td>
<td>Malicious</td>
<td>✓</td>
</tr>
<tr>
<td>MACAO (Π_\text{rss})</td>
<td>(O(1))</td>
<td>(O(\log N))</td>
<td>(\Omega(\log N))</td>
<td>(O(\log N))</td>
<td>(3)</td>
<td>Malicious</td>
<td>✓</td>
</tr>
<tr>
<td>MACAO (Π_\text{spdz})</td>
<td>(O(1))</td>
<td>(O(\log N))</td>
<td>(\Omega(\log N))</td>
<td>(O(\log N))</td>
<td>(t + 1)</td>
<td>Malicious</td>
<td>✓</td>
</tr>
</tbody>
</table>

\(*\) Block size is the maximum number of blocks that needs to be stored during a retrieval query. 
† Overhead denotes the number of blocks being transmitted between the client and the server(s) or between the servers. 
‡ Client block storage is defined as the number of data blocks being temporarily stored at the client. This is equivalent to the stash component. 
§ Number of servers required for the ORAM scheme. 
‡ Comp. over Enc. Data indicates the additional overhead due to secure computation.
MACAO Security

Definition 1 *(Simulation-based Multi-server ORAM Security with Verifiability).* Considering the ideal and real worlds as follows.

- **Ideal world.** Let $\mathcal{F}_{\text{oram}}$ be an ideal functionality, which maintains the latest version of the database on behalf of the client, and answers the client’s requests as follows.

  - **Setup:** Environment $\mathcal{Z}$ provides $\text{DB}$ to the client, who sends $\text{DB}$ to $\mathcal{F}_{\text{oram}}$. $\mathcal{F}_{\text{oram}}$ notifies simulator $\mathcal{S}_{\text{oram}}$ the setup is complete and the DB size. $\mathcal{S}_{\text{oram}}$ returns ok or abort to $\mathcal{F}_{\text{oram}}$. $\mathcal{F}_{\text{oram}}$ returns ok or $\bot$ to client accordingly.

  - **Access:** Environment $\mathcal{Z}$ specifies $\text{op} \in \{\text{read}(\text{bid}, \bot), \text{write}(\text{bid}, \text{data})\}$ as client’s input. Client sends $\text{op}$ to $\mathcal{F}_{\text{oram}}$. $\mathcal{F}_{\text{oram}}$ notifies $\mathcal{S}_{\text{oram}}$ (without revealing $\text{op}$). If $\mathcal{S}_{\text{oram}}$ returns ok to $\mathcal{F}_{\text{oram}}$, $\mathcal{F}_{\text{oram}}$ sends $\text{data}' \leftarrow \text{DB}[\text{bid}]$ to client, and updates $\text{DB}[\text{bid}] \leftarrow \text{data}$ if $\text{op} = \text{write}$. Client returns $\text{data}'$ to $\mathcal{Z}$. If $\mathcal{S}_{\text{oram}}$ returns abort to $\mathcal{F}_{\text{oram}}$, $\mathcal{F}_{\text{oram}}$ returns $\bot$ to client.

- **Real world.** $\mathcal{Z}$ gives the client $\text{DB}$. Client executes Setup protocol with servers $(S_0, \ldots, S_{\ell-1})$ on $\text{DB}$. For each access, $\mathcal{Z}$ specifies an input $\text{op} \in \{\text{read}(\text{bid}, \bot), \text{write}(\text{bid}, \text{data})\}$ to client. Client executes Access protocol with servers $(S_0, \ldots, S_{\ell-1})$. $\mathcal{Z}$ gets the view of the adversary $\mathcal{A}$ after each access. Client outputs to $\mathcal{Z}$ the accessed block or abort.

A protocol $\Pi_{\mathcal{F}}$ securely realizes $\mathcal{F}_{\text{oram}}$ in the presence of a malicious adversary corrupting $t$ servers iff for any PPT real-world adversary corrupting $t$ servers, there exists a simulator $\mathcal{S}_{\text{oram}}$, such that for all non-uniform, polynomial-time $Z$, there exists a negligible function $\text{negl}$ such that

$$\Pr[\text{REAL}_{\Pi_{\mathcal{F}}, \mathcal{A}, Z}(\lambda) = 1] - \Pr[\text{IDEAL}_{\mathcal{F}_{\text{oram}}, \mathcal{S}_{\text{oram}}, Z}(\lambda) = 1] \leq \text{negl}(\lambda)$$

**Theorem 1 (MACAO security).** MACAO framework is statistically (information-theoretically) secure by Definition 1.