CS848
Oblivious RAM

Sujaya Maiyya
Slides partially acquired from Prof. Amr El Abbadi
Data encryption to achieve privacy?

Honest-but-curious adversary

<table>
<thead>
<tr>
<th>Id</th>
<th>Medicine</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Humira</td>
</tr>
<tr>
<td>2</td>
<td>Januvia</td>
</tr>
<tr>
<td>3</td>
<td>Tivicay</td>
</tr>
<tr>
<td>4</td>
<td>Herceptin</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Id</th>
<th>Medicine</th>
</tr>
</thead>
<tbody>
<tr>
<td>X12</td>
<td>S6C...23</td>
</tr>
<tr>
<td>2SD</td>
<td>1NW...SJ</td>
</tr>
<tr>
<td>D45</td>
<td>3G8...SO</td>
</tr>
<tr>
<td>F4A</td>
<td>DJW...O8</td>
</tr>
</tbody>
</table>
Encryption is **not** sufficient for data privacy

<table>
<thead>
<tr>
<th>Id</th>
<th>Medicine</th>
</tr>
</thead>
<tbody>
<tr>
<td>X12</td>
<td>S6C...23</td>
</tr>
<tr>
<td>2SD</td>
<td>1NW...SJ</td>
</tr>
<tr>
<td>D45</td>
<td>3G8...SO</td>
</tr>
<tr>
<td>F4A</td>
<td>DJW...O8</td>
</tr>
</tbody>
</table>

- 57%
- 6%
- 16%
- 21%
Encryption is **not** sufficient for data privacy

---

**Percent of Medicines Sold in 2018 [1]**

<table>
<thead>
<tr>
<th>Medicine</th>
<th>Humira (Arthritis)</th>
<th>Januvia (Diabetes)</th>
<th>Tivicay (HIV)</th>
<th>Herceptin (Breast Cancer)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Id</td>
<td>Medicine</td>
<td>Percent Sold</td>
<td>Id</td>
<td>Medicine</td>
</tr>
<tr>
<td>X12</td>
<td>S6C...23</td>
<td>57%</td>
<td>2SD</td>
<td>1NW...SJ</td>
</tr>
</tbody>
</table>

Encryption is **not** sufficient for data privacy

![Access Pattern Attacks](https://truecostofhealthcare.org/pharmas-50-best-sellers/)

<table>
<thead>
<tr>
<th>Id</th>
<th>Medicine</th>
<th>Percent Sold</th>
</tr>
</thead>
<tbody>
<tr>
<td>X12</td>
<td>S6C...23</td>
<td>57%</td>
</tr>
<tr>
<td>2SD</td>
<td>1NW...SJ</td>
<td>6%</td>
</tr>
<tr>
<td>D45</td>
<td>3G8...SO</td>
<td>16%</td>
</tr>
<tr>
<td>F4A</td>
<td>DJW...O8</td>
<td>21%</td>
</tr>
</tbody>
</table>

**PERCENT OF MEDICINES SOLD IN 2018 [1]**

- Humira (Arthritis): 59%
- Januvia (Diabetes): 14%
- Tivicay (HIV): 20%
- Herceptin (Breast Cancer): 7%


Access Pattern Attacks

Many practical attacks: [IKK NDSS’12], [NKW CCS’15], [CGPR CCS’15], [KKNO CCS’16], [GLMP S&P’19], [KPT S&P’19], [OK Security’21], [OK Security’22]
Workload independence to protect against these attacks by hiding...

- which data is being accessed
- how old it is (when it was last accessed)
- whether the same data is being accessed
- access pattern (skewed vs. uniform)
- whether the access is a read or a write
Random accesses ensures workload independence

Goal: Oblivious Access
Translate each logical access
to a sequence of random-looking accesses

OBLIVIOUS RAM (ORAM)
Initially proposed by [Goldreich and Ostrovsky, JACM’96]
ORAM provides workload independence

• Clients wish to outsource data to an untrusted cloud storage
• Honest-But-Curious cloud can control & observe network & cloud storage
• Keep the data and access pattern private

\[
A = \begin{bmatrix}
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]
ORAM provides workload independence

- Clients wish to outsource data to an untrusted cloud storage
- Honest-But-Curious cloud can control & observe network & cloud storage
- Keep the data and access pattern private

\[ A = \begin{bmatrix} 0 & v & 0 & 0 \end{bmatrix} \]

1. The cloud never learns about \( v \)
2. The cloud never learns that clients accessed index 1
Typical (but not all) ORAM architecture

<table>
<thead>
<tr>
<th>Id</th>
<th>Loc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>..</td>
</tr>
<tr>
<td>2</td>
<td>..</td>
</tr>
<tr>
<td>3</td>
<td>..</td>
</tr>
<tr>
<td>4</td>
<td>..</td>
</tr>
</tbody>
</table>
Tree-based ORAM Developments

- While other forms ORAM constructions exist, most are theoretical in nature

A practical and famous solution
- Path ORAM: an extremely simple oblivious RAM protocol [Stefanov et al. CCS’13]
1000 ft overview of ORAM (PathORAM[1])

Step 1. Read path

Physical read entire path

Proxy

Read or write id 3

Step 2. Shuffle and Write path

Path id changes to another random path after each access to 3

Proxy

Shuffle and physical write to random path

Path ORAM [Stefanov et al. CCS’13]

Storage is organized as a binary tree

Every access to a random path
Items randomly re-assigned after every access

Server

Z blocks

Leaf 1
Leaf 2
Leaf 3
Leaf 4

Proxy

Stash

Pos Map
Path ORAM [Stefanov et al. CCS’13]

Storage is organized as a binary tree

Every access to a random path
Items randomly re-assigned after every access

Server

Proxy

Stores overflowed blocks

Stores the assignment
Path ORAM [Stefanov et al. CCS’13]

Storage is organized as a binary tree

Every access to a random path
Items randomly re-assigned after every access

Possible to outsource position map recursively
But need many rounds of communication
Path ORAM

Server

Read/Write block a

1) Read path
   • Fetch associated path
   • Read/Modify block
   • Assign block to a new random path in position map
   • Move all read blocks to stash

Proxy

Stash

a→3

Pos Map
Path ORAM

1) Read path
   - Fetch associated path
   - Read/Modify block
   - Assign block to a new random path in position map
   - Move all read blocks to stash
Path ORAM

1) Read path
• Fetch associated path
• Read/Modify block
• Assign block to a new random path in position map
• Move all read blocks to stash

2) Flush
• Push every block to the lowest non-full node that intersects with its assigned path (otherwise → stash)
Path ORAM

1) Read path
   - Fetch associated path
   - Read/Modify block
   - Assign block to a new random path in position map
   - Move all read blocks to stash

2) Flush
   - Push every block to the lowest non-full node that intersects with its assigned path (otherwise $\rightarrow$ stash)
Path ORAM

1) Read path
- Fetch associated path
- Read/Modify block
- Assign block to a new random path in position map
- Move all read blocks to stash

2) Flush
- Push every block to the lowest non-full node that intersects with its assigned path (otherwise → stash)

If root is full move to stash
Path ORAM

1) Read path
   - Fetch associated path
   - Read/Modify block
   - Assign block to a new random path in position map
   - Move all read blocks to stash

2) Flush
   - Push every block to the lowest non-full node that intersects with its assigned path (otherwise \(\rightarrow\) stash)

3) Write-back
   - Re-encrypt w/ fresh randomness
PathORAM

• Steps to access block $B$:
  1. Fetch path $P$ containing block $B$ from Server
  2. Update requested block $B$ (if write)
  3. Answer Client Request
  4. Assign block $B$ to random path
  5. Flush path $P$
  6. Writeback to server
Does PathORAM provide workload independence?

Say a client requested block $b$ stored in path $p$. From an adversary’s perspective

- Which data is accessed? ➔ One of the $Z^*\log N$ objects accessed
- When was $b$ last accessed? ➔ Only knows when $p$ was last accessed, not when $b$ was last accessed
- Did 2 subsequent requests access $b$? ➔ Only knows two random paths $p$ and $p'$ being accessed in subsequent requests
- Access pattern (uniform or skewed)? ➔ Observes accesses to random paths
- Is $b$ read or written? ➔ Each path is read and then written with fresh encryption

Yes! PathORAM provides workload independence!
ORAM – Security

• Let $A = \{(op_1, bid_1, val_1), \ldots (op_m, bid_m, val_m)\}$ represent a sequence of accesses $op_i \in \{\text{read, write}\}$, $bid_i$ is the block identifier, and $val_i$ is either updated value writes or null for reads.

• An ORAM scheme is secure if given two such sequences $A_0$ and $A_1$ and the system executed $A_i$, the adversary cannot guess which sequence was executed with probability $>> 1/2$. 
GAME

\[ b \leftarrow \{0, 1\} \]

ORAM - Security
ORAM - Security

GAME

$\$ \quad b \leftarrow \{0, 1\}$

ORAM

$D_0$

\begin{array}{c}
a \\
c
\end{array}

$D_1$

\begin{array}{c}
a \\
b
\end{array}$

Attacker
ORAM - Security

GAME

$\begin{align*}
&b \leftarrow \{0, 1\} \\
\end{align*}$

ORAM

$\begin{array}{c}
D_0 \\
\hline
a \\
c \\
\end{array}$

$\begin{array}{c}
D_1 \\
\hline
a \\
b \\
\end{array}$

Attacker
ORAM - Security

GAME

\( b \leftarrow \{0, 1\} \)

\[ \begin{align*}
D_b & \begin{array}{c}
? \\
? \\
\end{array} \\
\end{align*} \]

ORAM

GAME

\( b = 0 \)

\[ \begin{align*}
D_0 & \begin{array}{c}
a \\
c \\
\end{array} \\
\end{align*} \]

ORAM

GAME

\( b = 1 \)

\[ \begin{align*}
D_1 & \begin{array}{c}
a \\
b \\
\end{array} \\
\end{align*} \]

ORAM
ORAM - Security

\[
\text{GAME} \quad b \overset{\$}{\leftarrow} \{0, 1\}
\]

\[
\text{GAME} \quad b = 0
\]

\[
\text{GAME} \quad b = 1
\]

\[
op_0(\text{Read}(a)) \quad \op_1(\text{Read}(a))
\]
ORAM - Security

\[ b \leftarrow \{0, 1\} \]

\[ \text{GAME} \]

\[ \text{ORAM} \]

\[ D_b \]

\[ \text{op}_b \]

\[ \text{GAME} \]

\[ D_0 \]

\[ \text{GAME} \]

\[ D_1 \]

\[ \text{op}_0(\text{Read}(a)) \quad \text{op}_1(\text{Read}(a)) \]
ORAM - Security

GAME

\( b \leftarrow \{0, 1\} \)

\( \text{ORAM} \)

\( D_b \)

\( \text{op}_b \)

GAME

\( b = 0 \)

\( \text{ORAM} \)

\( D_0 \)

\( a \quad c \)

\( \text{op}_0(\text{Read}(a)) \)

GAME

\( b = 1 \)

\( \text{ORAM} \)

\( D_1 \)

\( a \quad b \)

\( \text{op}_1(\text{Read}(a)) \)

\( \text{Attacker} \)

\( \text{op}_0(\text{Read}(a)) \quad \text{op}_1(\text{Read}(a)) \)
ORAM - Security

GAME

\[ b \leftarrow \{0, 1\} \]

\[
\begin{array}{c}
 \text{Game 1} \\
 D_b \\
 \text{ORAM} \\
 \text{op}_b
\end{array}
\]

\[
\begin{array}{c}
 \text{Game 2} \\
 D_0 \\
 \text{ORAM} \\
 \text{op}_0(\text{Read}(a))
\end{array}
\]

\[
\begin{array}{c}
 \text{Game 3} \\
 D_1 \\
 \text{ORAM} \\
 \text{op}_1(\text{Read}(a))
\end{array}
\]

\[ \text{op}_0(\text{Read}(a)) \quad \text{op}_1(\text{Read}(b)) \]
ORAM - Security

GAME \( b \leftarrow \{0, 1\} \)

\[
\begin{array}{c}
D_b \\
\begin{array}{c}
? \\
? \\
\end{array}
\end{array}
\]

\( op_b \)

GAME \( b = 0 \)

\[
\begin{array}{c}
D_0 \\
\begin{array}{c}
a \\
c
\end{array}
\end{array}
\]

\( op_0(\text{Read}(a)) \)

GAME \( b = 1 \)

\[
\begin{array}{c}
D_1 \\
\begin{array}{c}
a \\
b
\end{array}
\end{array}
\]

\( op_1(\text{Read}(a)) \)

\( op_1(\text{Read}(b)) \)

\( op_0(\text{Read}(a)) \)
A secure ORAM has

\[ \Pr(A \diamond G \rightarrow \text{true}) = \frac{1}{2} + \text{negl} \]

i.e., adversary has negligible advantage is guessing bit \( b \)
Two observations on PathORAM

• Bandwidth overhead: $2^*Z^*\log N \rightarrow$ Depends on $Z$

• The online rounds of communication b/w client and server: 2 rounds
  • Even for read reqs, need an online write step

• Can these two limitations be improved?
RingORAM [Ren et al. Usenix Security’15]

Goals:

1. Eliminate the ORAM bandwidth’s dependence on \( Z \)
   How?
   Read exactly one block per bucket along the path

2. Reduce online communication rounds to 1
   How?
   Only read path for each client request, buffer writes, and
   write path back in an offline step
Ring ORAM

Each bucket stores at most $Z$ real blocks and at least $S$ dummy blocks.
Ring ORAM

Each bucket stores at most $Z$ real blocks and at least $S$ dummy blocks.

Every access to a random path reads only one block per bucket.
Ring ORAM

Each bucket stores at most $Z$ real blocks and at least $S$ dummy blocks.

Every access to a random path reads only one block per bucket.

Bucket metadata stores info on:
1. count: how many times is this bucket accessed
2. valid: which of the $Z+S$ blocks are not yet accessed
3. addr: ids of real blocks in a bucket

*Note: Bucket metadata actually stored at server*
Ring ORAM

1) Read path
For each bucket in path
- From valid and addr, either read real block or a valid dummy block
- Invalidate the read block in valid
- Increment count
Assign block to a new random path in position map
1) Read path
For each bucket in path
- From valid and addr, either read real block or a valid dummy block
- Invalidate the read block in valid
- Increment count
Assign block to a new random path in position map
Ring ORAM

1) **Read path**
For each bucket in path
- From *valid* and *addr*, either read real block or a valid dummy block
- Invalidate the read block in *valid*
- Increment *count*
Assign block to a new random path in position map
Ring ORAM

1) **Read path**
   For each bucket in path
   - From *valid* and *addr*, either read real block or a valid dummy block
   - Invalidate the read block in *valid*
   - Increment *count*
   Assign block to a new random path in position map

---

Server

```
Leaf 1
Leaf 2
Leaf 3
Leaf 4
```

---

Proxy

```
Stash

a\rightarrow 1
```

**Pos Map**

Only one real block
Ring ORAM

1) Read path
For each bucket in path
- From *valid* and *addr*, either read real block or a valid dummy block
- Invalidate the read block in *valid*
- Increment *count*
Assign block to a new random path in position map

Proxy

"Only one real block"

Server

```
<table>
<thead>
<tr>
<th>Leaf 1</th>
<th>Leaf 2</th>
<th>Leaf 3</th>
<th>Leaf 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Pos Map</th>
</tr>
</thead>
<tbody>
<tr>
<td>a→3</td>
</tr>
</tbody>
</table>
```
Ring ORAM

1) Read path
For each bucket in path
• From valid and addr, either read real block or a valid dummy block
• Invalidate the read block in valid
• Increment count
Assign block to a new random path in position map
1) **Read path**

For each bucket in path

- From *valid* and *addr*, either read real block or a valid dummy block
- Invalidate the read block in *valid*
- Increment *count*
- Assign block to a new random path in position map

---

**Ring ORAM**

**Server**

- **d**: dummy
- **a**: dummy
- **e**: dummy
- **f**: dummy

- Leaf 1
- Leaf 2
- Leaf 3
- Leaf 4

**Proxy**

- **a**:
  - Stash
  - a→1
  - Pos Map
Ring ORAM

1) Read path
For each bucket in path
• From valid and addr, either read real block or a valid dummy block
• Invalidate the read block in valid
• Increment count
• Assign block to a new random path in position map

2) Evict
• After $A$ read paths, in a deterministic order pick the next path to evict
• For each bucket, read all remaining valid real blocks (if < $Z$, read dummy) to stash
• Write each bucket from stash and reset all metadata
## Ring ORAM

### 1) Read path
For each bucket in path
- From valid and addr, either read real block or a valid dummy block
- Invalidate the read block in valid
- Increment count
- Assign block to a new random path in position map

### 2) Evict
- After A read paths, in a deterministic order pick the next path to evict
- For each bucket, read all remaining valid real blocks (if < Z, read dummy) to stash
- Write each bucket from stash and reset all metadata
**Ring ORAM**

1) **Read path**
   - For each bucket in path
     - From *valid* and *addr*, either read real block or a valid dummy block
     - Invalidate the read block in *valid*
     - Increment *count*
     - Assign block to a new random path in position map

2) **Evict**
   - After $A$ read paths, in a deterministic order pick the next path to evict
   - For each bucket, read all remaining valid real blocks (if < $Z$, read dummy) to *stash*
   - Write each bucket from *stash* and reset all metadata

---

**Server**

- Leaf 1: dummy
dummy
- Leaf 2: dummy
- Leaf 3: dummy
- Leaf 4: dummy

**Proxy**

- Stash
- Pos Map

**Real and dummy blocks will be shuffled**
1) Read path
For each bucket in path
- From valid and addr, either read real block or a valid dummy block
- Invalidate the read block in valid
- Increment count
- Assign block to a new random path in position map

2) Evict
- After $A$ read paths, in a deterministic order pick the next path to evict
- For each bucket, read all remaining valid real blocks (if < $Z$, read dummy) to stash
- Write each bucket from stash and reset all metadata

3) Early reshuffle
- If a bucket is accessed $s$ times, read all valid real blocks, permute, and write back
- Reset metadata for the bucket
Security arguments for Ring ORAM

1. Read path leaks no information
   • For each access, a random path is read
   • For each bucket, a random offset is read

2. Evict path leaks no information
   • Every $A$ accesses, a deterministically chosen path is read
   • Each bucket reads $Z$ blocks
   • Path written back

3. Early shuffle leaks no information
   • After $S$ accesses to a bucket, $Z$ blocks are read
   • Bucket is written back
Limitations of Path and Ring ORAM

• Both are sequential
  • TaoStore by Sahin et al. S&P’16 [Jan 25th]

• They both require a proxy to be practical
  • ConcurORAM by Chakraborti et al. NDSS’19 [Jan 30th]

• They do not support transactions or complex queries
  • Obladi by Crooks et al. OSDI’18 [Feb 1st]
  • ObliDB by Eskandarian et al. VLDB’19 [Mar 12th]

• Neither is fault tolerant
  • QuORAM by Maiyya et al. Usenix Security’22 [Feb 6th]

• Neither is scalable
  • ObliviStore by Stefanova et al. S&P’13 (not reading)
  • Snoopy by Dauterman et al. SOSP’21 [Mar 14th]
Conclusion

• Access patterns leak information

• Need workload independence

• Databases using ORAM ensure workload independence

• PathORAM: a highly efficient tree-based ORAM
  • Simple abstraction & easy to implement

• RingORAM: optimizes PathORAM by reducing online bandwidth cost