Oblivious RAM (ORAM)

more precisely: Path ORAM

Cloud storage

- Amazon S3, EBS
- Dropbox
- SkyDrive
- EMC Atmos
- Windows Azure Storage
- Mozy
- iCloud
- Google Storage
Cloud storage

Sensitive documents should be encrypted

Files

User

Encrypted Files

Server
But: the *metadata* is leaked

The *access pattern* leaks:
- Which data is being accessed
- When the data was last accessed
- Whether the same data is being accessed
- Access randomly or sequentially
- Whether the access is a read or write
Path ORAM

• Server storage
  Blocks of data

• Client storage
  position map, stash
Path ORAM

**Server:** data blocks stored in nodes of a tree

- Some nodes are empty
- Nodes can contain multiple blocks

**Diagram notes:**
- Block
- Bucket
- Block #
- Data
"Position" of a block is the leaf sequence number.
A block must be placed on the path leading to its “position”

If a BLOCK’s position is 5, it can only be in the this path
A block must be placed on the path to its “position”

If a BLOCK’s position is 1, it can only be in the this path
A block’s position is **chosen randomly** from 1 to $2^L$ (number of leaves).

Some nodes (buckets) have multiple blocks while some are empty.
Path ORAM

**Client storage:** position map & stash

Position map: to store each block’s “position” (assume there are N blocks in the tree)

<table>
<thead>
<tr>
<th>Block #</th>
<th>Block’s “position”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>N-1</td>
<td></td>
</tr>
<tr>
<td>N</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
</tr>
</tbody>
</table>
Path ORAM

**Client storage:** position map & stash

*position map:* to store each block’s “position” (assume there are N blocks in the tree)

<table>
<thead>
<tr>
<th>Block #</th>
<th>Block’s “position”</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Block #</th>
<th>1</th>
<th>2</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Client storage: position map & stash
Path ORAM

**Client storage**: position map & stash

Stash: 1. block cache
2. stores blocks “overflowed” from the server

-one block-
Path ORAM

Access(op, a, data*):

1: \( x \leftarrow \text{position}[a] \)
2: \( \text{position}[a] \leftarrow \text{UniformRandom}(0 \ldots 2^L - 1) \)
3: for \( \ell \in \{0, 1, \ldots, L\} \) do
4: \( S \leftarrow S \cup \text{ReadBucket}(\mathcal{P}(x, \ell)) \)
5: end for
6: data \leftarrow \text{Read block a from } S
7: if op = write then
8: \( S \leftarrow (S - \{(a, \text{data})\}) \cup \{(a, \text{data}^*)\} \)
9: end if
10: for \( \ell \in \{L, L - 1, \ldots, 0\} \) do
11: \( S' \leftarrow \{(a', \text{data'}) \in S : \mathcal{P}(x, \ell) = \mathcal{P}(\text{position}[a'], \ell)\} \)
12: \( S' \leftarrow \text{Select min}(|S'|, Z) \text{ blocks from } S'. \)
13: \( S \leftarrow S - S' \)
14: \( \text{WriteBucket}(\mathcal{P}(x, \ell), S') \)
15: end for
16: return data
Path ORAM

An example for access (op, a, data)

Example:
A user wants to modify block “7”
Example: A user wants to modify block “7”

Path ORAM

position map

stash
Path ORAM

Example: A user wants to modify block “7”

1. Lookup block’s position

Server

Client

position map

stash
Path ORAM

Example: A user wants to modify block “7”

2. Read the entire path

Server

Client

position map

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
</tr>
</tbody>
</table>

stash

| 8 | 9 | 1 | 7 | 13 | 10 | 6 |

decrypt
Path ORAM

Example: A user wants to modify block “7”

Server

Client

Position Map

3. Client can now read/modify data in block

decrypt

stash
Path ORAM

Example: A user wants to modify block “7”

4. Assign a new random position.

Position[block 7] = UniformRandom(1, 2, 3, 4)

here = 1

Note: the blocks on the tree are encrypted
Path ORAM

Principles:
A. Blocks should be pushed down as deep as possible
B. Blocks should always be on the path to its position

Note: the blocks on the tree are encrypted

5. Write path back
Principle A:
Push blocks as deep as possible

Try the deepest possible node first

Note: the blocks on the tree are encrypted

Server

Client

position map

5. Write path back.

stash
Path ORAM

Principle A:
Push blocks as deep as possible

If full, try the upper level

Note: the blocks on the tree are encrypted

5. Write path back.
Path ORAM

Principle A: Push blocks as deep as possible

If full, try the upper level

Note: the blocks on the tree are encrypted

5. Write path back.

Stash

Client

Position map

Server
**Path ORAM**

**Principle A:**
**Push blocks as deep as possible**

If root is full, store in the stash

Note: the blocks on the tree are encrypted

5. Write path back.
Path ORAM

Principle B: The block should always be on its path to “position”

Note: the blocks on the tree are encrypted

5. Write path back

This diagram illustrates the structure and functionality of Path ORAM (Path-Oriented Oblivious RAM). It shows the interaction between the server and client, with focus on the position map and the stash. The server and client communicate through a series of steps, ensuring privacy and security. The position map and stash are key components in this process, allowing for efficient data retrieval without revealing the access pattern.
**Path ORAM**

**Principle B:**
The block should always be on its path to “position”

Block 8, 1, 13 can end up at any buckets on the entire path

5. Write path back

Note: the blocks on the tree are encrypted
Path ORAM

Principle B: The block should always be on its path to “position”

Block 9, 7, 10, 6 can only end up at these buckets

Note: the blocks on the tree are encrypted

5. Write path back
Path ORAM

Get back to our example:

Note: the blocks on the tree are encrypted

Server

Client

position map

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

5. Write path back

stash

<p>| | | | | | | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>9</td>
<td>1</td>
<td>7</td>
<td>13</td>
<td>10</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Path ORAM

Note: the blocks on the tree are encrypted

5. Write path back
Path ORAM

Note:
the blocks on the tree are encrypted

5. Write path back
Path ORAM

Note: the blocks on the tree are encrypted

5. Write path back
Path ORAM

Server

Client

position map

1 2 3 4 5 6 7 8 9 10 11 12 13 14

2 4 3 4 3 1 1 2 1 1 4 3 2 1

Note: the blocks on the tree are encrypted

5. Write path back

stash
Client storage

Position map:
- Log N bits per block
- Totally N log N bits

Too much !!!
Recursion

- Store the position map in another ORAM.
- Do this recursively.
Theorems

1. Given two accesses, an adversary that controls the network and the cloud service provider cannot see what blocks were read or written, or if the two accesses referred to the same block or different blocks.

2. The stash is very unlikely to get more than a few blocks in it.