- Scribes?
- Lecture recording

1) Communication complexity
2) Size of memory accessed by the server
3) Support for writes

PIR - Private Information Retrieval
- Read-only memory
- Two-round protocols
- Server stores data in cleartext
- Single server, computational security
- Multiple servers, each storing a copy of the data, information-theoretic security (assuming servers do not collude)
Oblivious RAM (ORAM) - Goldreich-Ostrovsky
- interactive protocols
- single-server setting (primarily)
- read/write access
- server updates its storage as part of the protocol

Square-root ORAM $n = \sqrt{n}$

Server:

$\begin{array}{cccccccc}
& x_0 & x_1 & x_2 & x_3 & \cdots & x_{\sqrt{n}} & \cdots & x_{\sqrt{n}} \\
\hline
\text{stash} & \text{stash} & \text{stash} & \text{stash} & \text{stash} & \cdots & \text{stash} & \cdots & \text{stash} \\
\end{array}$

$j = F_k(i)$

Client:
- choose key $k$ defining permutation over $\{0, \ldots, N - \sqrt{n}\}$
- shuffle data so $x_i$ is stored at position $F_k(i)$

To read the value at position $i$:

$\rightarrow$ scan the entire stash to see if $j$ entry $(i, *)$
- if so, accept right-away such value
- if no such entry in the stash
- request from the server the data stored at position $F_k(i)$
- write $x_i$ into next available location in stash
- if there was an entry in the stash
- request from the server the data stored at position $F_k(N + \text{ctr})$, \text{ctr}++
- write $1$ into next available location in the stash

After $\sqrt{n}$ accesses, client needs to refresh data stored at the server
Oblivious sorting: in theory, can be done using $O(n \log n)$ swaps
in practice, done using $O(n \log^2 n)$ swaps

Complexity:
- ignoring refresh, communication = #memory accesses = $O(Sn)$
- refresh: communication = $O(n \log n)$
  amortized comm. = $O\left(\frac{n \log n}{Sn}\right) = O(Sn \log n)$
  $\Rightarrow$ overall, amortized complexity $O(Sn \log n)$

Path ORAM

Server

Client
- maintain a mapping $M$ from $\mathbb{F}_2^{\geq c(n)}$ to leaves of tree
- element $x_i$ will be stored at some node on the path from the root to $M(x_i)$
- to read the value at index $i$,
  - request from server all nodes on the path to $M(x_i)$
- Choose fresh random value for $M(i)$
- Write back the values to the same path
  - push down all values as far down the path as possible
  - eliminate any duplicates; keep foremost value

$M(1) = 4$
$M(2) = 3$

If $|x_i| = 2\log N$, and tree has $N$ leaves, then

storing $M_0$ requires $N \log N$ bits

$\Rightarrow$ factor of 2 less than trivial

Recurse: store $M_0$ using a smaller version of the scheme

$M_0$:

\[
\begin{array}{c|c|c}
M_0 & M_1 & M_2 \\
\hline
2\log N & 2\log N & \epsilon/2 \\
0 & n/2 & \end{array}
\]

Storing $x_1, \ldots, x_N$