Last week review:

\[ C \left( pk, sk \right) \leftarrow \text{Gen} \left( m \right) \]

\[ \frac{S}{\text{Enc}_m \left( x_1 \right), \ldots, \text{Enc}_m \left( x_n \right)} \]

\[ \frac{\text{Circuit}}{\left[ b_0 \right] = C_0 \quad C_1 = \left[ b_1 \right]} \]

\[ C_2 = C_0 \cdot C_1 = \left[ b_0 + b_1 \right] \]

- Fully homomorphic encryption (FHE):

\[ C \]

\[ S \]

local hom. eval of circuit

\[ \rightarrow \text{Enc}_m(\text{out}_1), \ldots, \text{Enc}_m(\text{out}_n) \]

- Client can off load computations to server

- Hom. enc is expensive
Secure computation in the RAM model:

Turing Machine: need at least linear time for computation

\[ A^m(x) \]
\[ st = x \quad \text{ (state)} \]
\[ Getched = \bot \quad \text{ (the thing)} \]

\[
\text{While } (i) \{ \\
\quad (st, addr, val) \leftarrow \text{Next Inst}(st, Getched) \\
\quad \text{if } st = \text{halt} \\
\qquad \text{output } val \\
\quad \text{else} \\
\qquad \text{Getched} = M(addr) \\
\qquad \text{if } (val) = \text{None} \\
\qquad \quad M(addr)
\}
\]
Oblivious RAM (ORAM)

Trivial ORAM:
- To read address \( r \), do linear scan over \( M \)
  (and encrypt contents)

Non-trivial ORAM scheme:
- (Goldreich-Ostrovsky '94)

\[
\begin{align*}
\text{Space:} & \quad \beta + \sqrt{\beta} & \quad \sqrt{\beta} \\
& \quad \text{original contents (permanently)} & \quad \text{cache}
\end{align*}
\]

Initialize:
- Store in CPU key \( k \) for pseudorandom permutation over \( \{1, \ldots, \beta + \sqrt{\beta}\} \)
- In virtual memory, store \( \{(i, M[i])\}_{i=1}^{\beta} \)
in position \( F_k(i) \)
- In virtual memory, store \((i, \text{null})\) \( \beta + \sqrt{\beta} \)

To read address \( i \) do:
1) Perform linear scan of cache for element of the form \((i', \text{val})\)
   \( i' \leq 16 \text{ bound}, \text{set val = v} \)
2) If not bound, probe position \( F_k(i) \) to set value \( i' \leq 16 \text{ bound}, \text{probe position } F_k(\beta + \text{ctr}) \), \( \text{ctr} \) ++
3) Linear scan of cache to write the (now) value at \( M[i] \)

\( \sqrt{\beta} \text{ times refresh} \)
Refresh (every $\sqrt{B}$ steps)
- Choose fresh key $k'$
- (obliviously) re-shuffle elements according to $k'$
- empty cache

$O(B \log B)$ sorting +
$O(n)$ additional operations

- get good amortized complexity

Amortized complexity:
for $\sqrt{B}$ elements:
access $\sqrt{B} \cdot (2 \sqrt{B} + 1) + O(B \log B)$ memory patterns

$O(B \log B) = O(\sqrt{B} \log B)$

$B^ε$: cache $B^ε$

$B^ε = O(B^ε) + O(B \log B) = O(B^{1-ε})$