Cryptography Meets Algorithms (15893) Lecture Notes

Lecture 11: ORAM Lower Bound

Scribe: Kunming Jiang

March 1, 2024

In this lecture, we will prove that any Oblivious RAM (ORAM) scheme must suffer from logarithmic overhead. We will show two proofs. The first proof was described in Goldreich and Ostrovsky's original paper on ORAM [GO96]. Their lower bound has two restrictions: 1) it works only for *statistically* secure ORAMs and 2) it assumes that the ORAM is in the *balls-and-bins* model, i.e., the scheme does not perform any encoding on the payload strings stored in memory. Many years later, in 2018, the work of Larsen and Nielsen [LN18] proved a new lower bound removing both of these restrictions. Interestingly, their proof uses techniques from the data structure lower bound literature. We will also cover Larsen and Nielsen's lower bound in today's lecture.

1 Goldreich and Ostrovsky's Lower Bound

Theorem 1. Consider any perfectly secure ORAM scheme in the balls-and-bins model such that the ORAM begins with a memory already loaded with n words. Then, any logical request sequence of length t must incur $\max(n, \Omega(t \log t))$ total cost. Further, the lower bound works even for read-only requests.

Proof. Consider the following game. Initially, there are n balls, and ball i is stored in cell i of the memory. There is a sequence of t logical requests, to read the balls indexed $i_1, \ldots, i_t$ respectively. A player can hold up to m balls in her hand. In every time step indexed $1, 2, \ldots, q$, she can visit a memory cell of her choice and perform one of the following hidden operations:

1. Take a ball from the memory cell and put it in her hand;
2. Place a ball from her hand to the memory cell (if it is currently empty);
3. Do nothing.

The player's action sequence can satisfy the request sequence, iff there is a subsequence $1 \leq j_1 \leq j_2 \leq \ldots \leq j_t \leq q$, such that for all $k \in [t]$, the ball indexed i_k is in the player's hands at the end of time step j_k.

Suppose that an adversary can observe which memory cell the player visits in every time step, but cannot observe which hidden operation the player performs. Similarly, the adversary cannot observe which balls are stored in the memory cells or the player's hands. Now, the player's job is to satisfy the logical request sequence $i_1, \ldots, i_t$ without revealing any information about the logical request sequence. $\square$

Goldreich-Ostrovsky Lower Bound [GO96] Any ORAM scheme (in the balls-and-bins model) must have at least logarithmic overhead.
To prove, let

- m = number of balls the client can hold in its hand

- t = logical request sequence length
- n = memory size

Every step, client can visit some memory location i, and

1. take no action
2. take a ball from i
3. place a ball into i

Now given

- initial memory with n balls,
- requests $r_1 \dots r_t$
- implementation $(\vec{\text{addr}}, \vec{\text{action}})$ and $q = |\vec{\text{addr}}|$

an observer can only see $\vec{\text{addr}}$ but not $\vec{\text{action}}$.

Q: Assume perfect security, how many request sequences can $\vec{\text{addr}}$ realize?

1. For every request, there are $m+2$ possible action (1 from doing nothing, 1 from taking a ball, and m from placing a ball).
2. At the end of each $(\vec{\text{addr}}, \vec{\text{action}})$, the client can use the m balls it has to express m different results.
3. Need to realize all n^t possible memory access sequences.

Thus,

$$\begin{aligned} (m+2)^q \cdot m^q \geq n^t \quad &\Rightarrow \quad q \log m + q \log(m+2) \geq t \log n \\ &\Rightarrow \quad q/t \geq \tfrac{\log n}{2 \log(m+2)} \end{aligned}$$

Restrction of G-O LB:

1. Balls and bins assumptions
2. Only works for statistically secure schemes

2 Lauren-Nielsen

Lauren-Nielsen Lower Bound [LN18] Logarithmic LB for ORAM but removing these restrictions.
Assumptions:

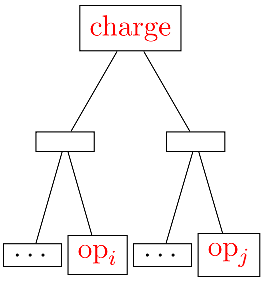
- read and write in "word"
- word size $\geq \log N$ (memory size)

Assume that there is a binary tree, where each leaf node corresponds to a consecutive (read, write) pair. W.L.o.G, fix

$$\vec{\text{op}} = \text{read}(0), \text{write}(0,0), \dots, \text{read}(0), \text{write}(0,0)$$

Want to show: number of probes into memory must be "high" for $\vec{\text{op}}$.

How the tree helps us count: suppose op_j probes some mem location, and the last time this location was probed was during op_i. Then we charge this probe to the least common ancestor in the tree of op_i and op_j.



Assume that the adversary can observe the physical probe locations and the boundary between each op. Then it can construct this tree in polynomial time, i.e. how many probes are charged to each node. (*This is where computational security kicks in.*)

By ORAM security: $\forall\ \vec{\text{op}}, \vec{\text{op}}'$ of same length, the two trees constructed must be computationally indistinguishable from each other.

For every subtree v of size $2m$, let the left half of the leaves denote

$$\text{read}(0), \text{write}(1, r_1)$$

$$\vdots$$

$$\text{read}(0), \text{write}(m, r_m)$$

and the right half denote

$$(\text{read}(1), \text{write}(0, 0))$$

$$\vdots$$

$$(\text{read}(m), \text{write}(0, 0))$$

Idea: when we count the probes assigned to each node v, we can use the worst-case sequence for v.

Intuition: imagine balls-and-bins model, number of probes assigned to $v \geq \frac{|\text{leaves under } v|}{2}$. Thus, at each level, there will be at least $T/2$ probes. Since there are $\log T$ levels, total number of probes at least $O(T \log T)$.

Information Transfer Technique (Encoding Argument): let coins be the randomness consumed by ORAM.

- Encode $(r_1, \ldots r_m$, coins$)$
  1. Execute ORAM over prefix $\text{read}(0), \text{write}(0, 0), \ldots$
  2. Execute $\text{read}(0), \text{write}(1, r_1), \ldots, \text{read}(0), \text{write}(m, r_m)$
  3. Execute $\text{read}(1), \text{write}(0, 0), \ldots, \text{read}(m), \text{write}(0, 0)$
- Encoding (C) = for each memory location probed during 2 and 3, record (location, last value written during 2) and the CPU register at the end of 2
- Decode $(C$, coins$)$
  1. Same as 1 in Encode
  2. Reset CPU state to C.cpuState for every (loc, val) in C, let mem[loc]$\leftarrow$ val
  3. same as 3 in Encode
- Decoder output the outcomes of the read ops in 3

References

[GO96] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious rams. *J. ACM*, 43(3):431–473, may 1996.

[LN18] Kasper Green Larsen and Jesper Buus Nielsen. Yes, there is an oblivious ram lower bound! Cryptology ePrint Archive, Paper 2018/423, 2018. `https://eprint.iacr.org/2018/423`.