Cryptography Meets Algorithms (15893) Lecture Notes

Lecture 11: ORAM Lower Bounds

Scribe: Kunming Jiang

March 18, 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 (Goldreich-Ostrovsky ORAM lower bound)**.** Consider any perfectly secure ORAM scheme in the balls-and-bins model such that the memory is initialized with n words, and the client has m space. Then, any logical request sequence of length t must incur $\max(n, \Omega(t \log_m n))$ 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, and initially, her hand is empty. In every time step indexed $1, 2, \ldots, q$, she can visit a memory cell of her choice and take one of the following hidden actions:

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 action the player takes. 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. We assume that the adversary knows the parameters n, t, m.

First, it is easy to see that $q \geq m$. To see this, consider a logical request sequence of length 1. To satisfy the request, if there is some memory cell the player does not visit, it directly reveals that the request is not for that index. In the remainder of the proof, we focus on proving $q \geq \Omega(t \log_m n)$.

Consider a fixed sequence of memory cells visited $(v_1, \ldots, v_q)$ that happens with non-zero probability when the logical request sequence is $(1, 1, \ldots, 1)$. Because of the privacy requirement, $(v_1, \ldots, v_q)$ must be able to satisfy any logical request sequence of length t, and the total number of logical request sequences of length t is n^t.

Now, how many logical request sequences can $(v_1, \ldots, v_q)$ satisfy? When we fix the physical access sequence $(v_1, \ldots, v_q)$, in each of the q time steps, the player can choose one of at most $m + 2$ hidden actions. Specifically, if the player chooses to place a ball, the ball can be one of the (up to) m balls in her hand. Therefore, fixing $(v_1, \ldots, v_q)$, there are $(m + 2)^q$ possible action sequences. Further, when we fix the sequence of memory cells visited $(v_1, \ldots, v_q)$ as well as the sequence of hidden actions, it can satisfy at most $\binom{q}{t} \cdot m^t$ logical requests, where $\binom{q}{t}$ is the number of ways to choose t out of the q time steps, and for each of the t chosen time steps, the player can choose one out of up to m balls in her hand to satisfy the next request.

Summarizing the above, we have that

$$\binom{q}{t} \cdot (m + 2)^q \geq n^t$$

Using the fact that $\binom{q}{t} \leq \left(\frac{eq}{t}\right)^t$, we have

$$\left(\frac{eq}{t}\right)^t \cdot (m + 2)^q \geq n^t$$

Therefore,

$$q \log(m + 2) \geq t(\log n - \log(eq/t))$$

If $q/t > \sqrt{n}$, then $q > t \log n$ trivially follows. Therefore, we may assume that $q/t \leq \sqrt{n}$. In this case, we have that $q \geq \Omega(t \log n / \log(m + 2))$ which gives the desired bound. □

In the above proof, the game setup where the player can only grab and place balls means that this proof works only for ORAM schemes in the balls-and-bins model. Moreover, the counting-based argument implicitly assumes that the scheme is perfectly secure. With some more work, it is possible to extend the above proof to work for statistical (rather than perfect) security.

2 Larsen and Nielsen's Lower Bound

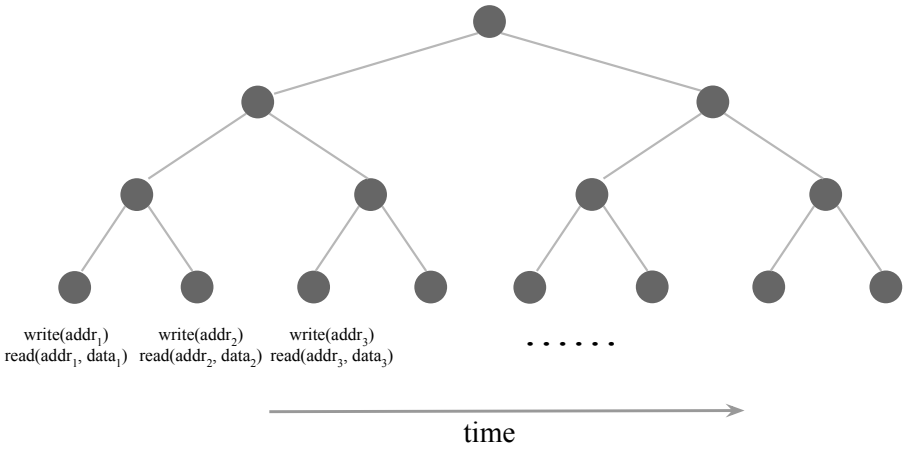
Goldreich and Ostrovsky's lower bound suffers from two restrictions: 1) the ORAM scheme must be statistically secure; and 2) the ORAM scheme must follow the balls-and-bins model, i.e., the algorithm cannot perform any encoding of the payload strings.

We now show a more recent lower bound proof by Larsen and Nielsen [LN18] which removes these two restrictions. Their proof uses elegant techniques from the data structure lower bound literature, called the "information transfer" technique.

We want to realize a memory abstraction that supports two types of operations, `read(addr)` and `write(addr, data)`. The address space is from 0 to $n-1$. We will make a couple simplifying assumptions. We assume that all read and write operations operate on *words*, and we assume that the word size is at least $\log n$ which is a standard assumption for the RAM model. We assume that the ORAM client has $O(1)$ space.

Imagine a physical memory array that contains *cells* and each cell stores one word. Our ORAM algorithm will read/write some of the cells upon receiving a request. In our proof, we will show that the following canonical request sequence of length $2T$ must make many cell probes.

$$op^* := \texttt{write}(0, 0), \texttt{read}(0), \texttt{write}(0, 0), \texttt{read}(0), \ldots$$



We will count the cell probes in a clever way, by assigning the cell probes to to nodes in a binary tree. Imagine a binary tree with T leaf nodes. Each leaf node i is associated with two operations $\texttt{write}(\texttt{addr}_i, \texttt{data}_i)$ and $\texttt{read}(\texttt{addr}'_i)$. Imagine that these operations occur in chronological order from left to right. Given a leaf node i, to satisfy the requests associated with it, we need to make some cell probes. We will charge these cell probes to ancestors of the leaf node i (including i itself) in the following way. Suppose a cell probe visits some cell that was last written to during the requests associated with some leaf node $j \leq i$. Then, this cell probe is charged to the lowest common ancestor of i and j. Therefore, once we complete the entire request sequence on the leaf nodes, we can assign all cell probes to nodes in the binary tree. To get the total number of cell probes, we can sum up the cell probes assigned to each node of the tree.

Let $P_u(op)$ denote the expected number of cell probes assigned to some node u after executing the request sequence op. We want to show that $P_u(op^*)$ is large for any u. To do so, we will need to use the security requirement of ORAM, which says that the access patterns of any two request sequences of the same length must be computationally indistinguishable. Now, suppose that an adversary observes the cell probes for each request, it can then assign these probes to nodes in the binary tree in polynomnial time. Therefore, the security requirement of ORAM implies that for any request sequence op of the same length $2T$, $|P_u(op) - P_u(op^*)| \leq \mathsf{negl}(n)$, and this must hold for any node u. For the rest of the proof, we will simply assume $P_u(op^*) = P_u(op)$ for any request sequence op of the same length — it is not hard to modify the argument to account for the negligibly small difference.

Now, to lower bound $P_u(op^*)$ for each u, we will come up with the *worst-case request sequence* for u, since the expected number of probes assigned to u under op^* is the same as under any request sequence op' of the same length. We use d to denote the height of the node u where the leaf is at height 0 and the root is at height $\log_2 T$. We use "Epoch L" to refer to the requests associated with leaves in the left subtree of u, and we use "Epoch R" to refer to the requests associated with leaves in the right subtree of u. See Figure 1 for an illustration.

Consider the following request sequence:

- *Before Epoch L*:

$$\texttt{write}(0,0), \texttt{read}(0),\ \texttt{write}(0,0), \texttt{read}(0),\ \ldots,\ \texttt{write}(0,0), \texttt{read}(0)$$

- *Epoch L*:

$$\texttt{write}(1, r_1), \texttt{read}(0),\ \texttt{write}(2, r_2), \texttt{read}(0),\ \ldots,\ \texttt{write}(2^d, r_{2^d}), \texttt{read}(0),$$

where $r_1, r_2, \ldots, r_{2^d}$ are randomly and independently sampled words.

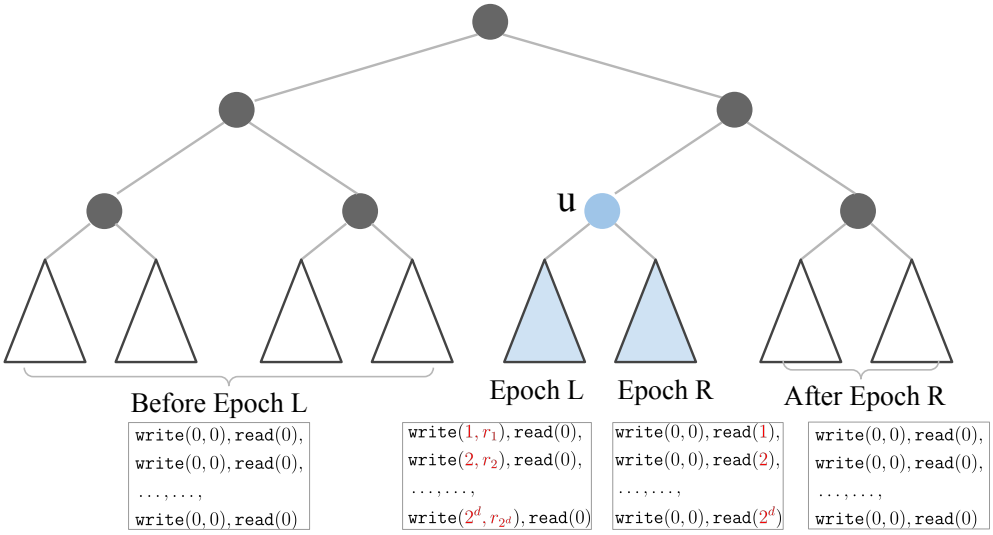


Figure 1: Use the worst-case sequence for u to lower bound the number of probes assigned to u.

- *Epoch R*:

$$\texttt{write}(0,0), \texttt{read}(1),\ \texttt{write}(0,0), \texttt{read}(2),\ \ldots,\ \texttt{write}(0,0), \texttt{read}(2^d),$$

- *After Epoch R*:

$$\texttt{write}(0,0), \texttt{read}(0),\ \texttt{write}(0,0), \texttt{read}(0),\ \ldots,$$

Informal intuition. The intuition is that to satisfy the read requests in Epoch R, we will need to transfer sufficient information from the write requests in Epoch L. Ignoring the $O(1)$ client space, information can only be transferred from the write requests in Epoch L to the read requests in Epoch R when Epoch R probes memory cells last written to during in Epoch L. Since the data strings written during Epoch L are all randomly chosen, roughly speaking, we need to transfer 2^d words of information from Epoch L to Epoch R, and thus the expected number of probes assigned to node u needs to be at least 2^d.

Using the same argument, we can find the worst-case request sequence for any node u, and argue that the expected number of probes assigned to u must be proportional to the subtree size of u. Thus, when we sum up all nodes u, we get that the total number of probes in the tree is at least $\Omega(T \log T)$. Amortized to $2T$ requests, it means that the average cost of each request is at least $\Omega(\log T)$.

Formal encoding argument. To formalize the above the intuition, we can use an encoding-based argument. Fix the node u, and fix the randomness tape of the ORAM algorithm henceforth denoted coins. The encoder and decoder will share coins and note that the data strings $r_1, \ldots, r_{2^d}$ are chosen at random and independent of coins.

- $\mathsf{Encode}(\mathsf{coins}, r_1, \ldots, r_{2^d})$: The encoding algorithm executes the ORAM parametrized with coins, over requests before Epoch L. followed by requests in Epoch L, followed by requests in Epoch R. Now, output the following encoding:
  - the client's memory state at the end of Epoch L henceforth denoted cmem;
  - for every cell probed in Epoch R which were last written in Epoch L, output the index of the cell and the value of the cell at the end of Epoch L — henceforth this part is denoted $\{\texttt{addr}_i, v_i\}_i$.

- $\mathsf{Decode}(\mathsf{coins}, (\mathsf{cmem}, \{\mathtt{addr}_i, v_i\}_i))$: The decoder executes the ORAM algorithm parametrized with coins, over requests before Epoch L. At this moment, the decoder resets the ORAM's client memory to cmem; further, for each $(\mathtt{addr}_i, v_i)$, it resets the memory cell at $\mathtt{addr}_i$ to the value v_i. Now the decoder executes the requests in Epoch R. Output the output of each read request in Epoch R.

Note that the decoder cannot execute the request in Epoch L itself because it does not know $r_1, \ldots, r_{2^d}$. However, it knows all the requests before Epoch L and after Epoch L.

Correctness of decoding is easy to verify. By our word size assumption, an address can be stored in a single memory word. Thus, the encoding contains at most $m + 2P$ words, where m denotes the client memory size (in terms of words), and P is the number of cells probed in Epoch R that were last written to in Epoch L.

By Shannon's source coding theorem, for every choice of coins, the expected length of the encoding must be at least the entropy of the string $r_1, \ldots, r_{2^d}$ (where expectation is taken over the random choice of $r_1, \ldots, r^{2^d}$). Thus, we have that

$$\mathbf{E}[m + 2P] \geq 2^d \quad \text{i.e.,} \quad \mathbf{E}[P] \geq 2^{d-1} - m/2$$

In other words, for every choice of coins of the ORAM algorithm, the expected number of probes assigned to a node u over the choice of $r_1, \ldots, r_{2^d}$ is at least $2^{d-1} - m/2$ where d is the height of u. Suppose that $m = O(1)$, and summing over all nodes in the tree whose height is at least $\log_2 m + 1$, we have that the total expected number of probes is at least $\Omega(T \log T)$ for a random choice of $r_1, \ldots, r_{2^d}$. Amortizing over $2T$ requests, the amortized cost per request is at least $\Omega(\log T)$.

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`.