Lecture 8: More ORAM

MIT - 6.893

Fall 2020

Henry Corrigan-Gibbs

Today

- Summary of survey
- Recap: Square-Root ORAM
- Tree-based ("modern") ORAM

Logistics

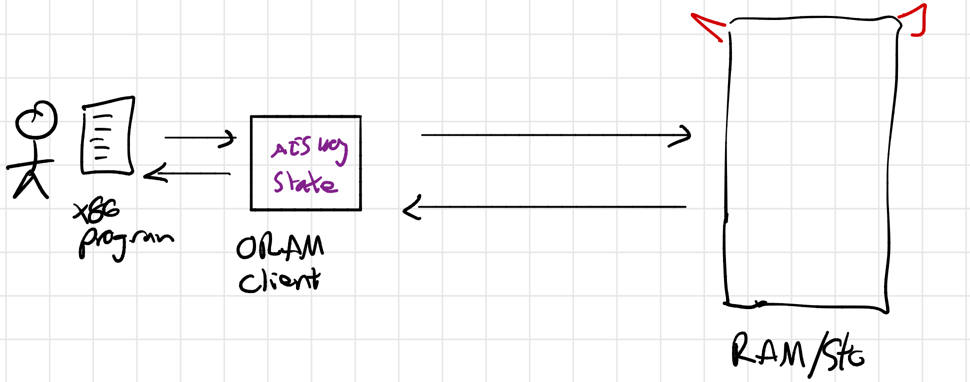
- HW3 out now
- Please answer questions on Piazza via anon feedback form!

Summary of Survey

- Most people are familiar w/ Shamir
- Little coverage of other topics
- Likely plan: "Modern" MPC + ZK w/ focus on applications (disjoint from standard crypto theory course) ...mostly

  * Overview of MPC, Secret Sharing (fast)
  * BGW protocol ... practical limitations
  * MPC in practice (applications)
  * Linear PCPs
  * zkSNARKs from linear PCPs (GGPR, ...) + applications
  * ZK Proofs on secret-shared data + applications

Recap: Square-Root ORAM



- Client can run x86 program...
- Adv that sees memory access pattern to RAM
" learns nothing about what accesses client is making.

With all ORAMs:

* We don't bother talking about hiding the data in memory words

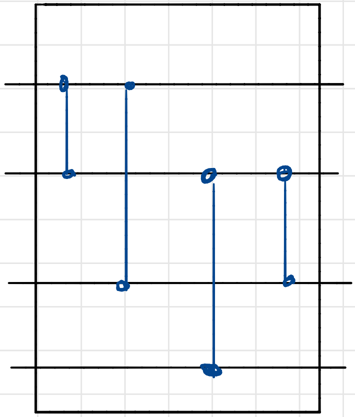
↳ Use standard encryption (e.g. AES-GCM) and reencrypt/rerandomize w/ every R/W.

⟹ Will show an even simpler $\sqrt{n}$ ORAM... essentially the same as last class but slightly pared down.

Oblivious sorting algorithm (not about ORAM... just normal alg)

Sorting alg whose RAM access pattern is independent of data.

Bad example: Bubble sort
- Iterate over all elms in list, swapping order of elms in wrong order
- Continue until sorted.



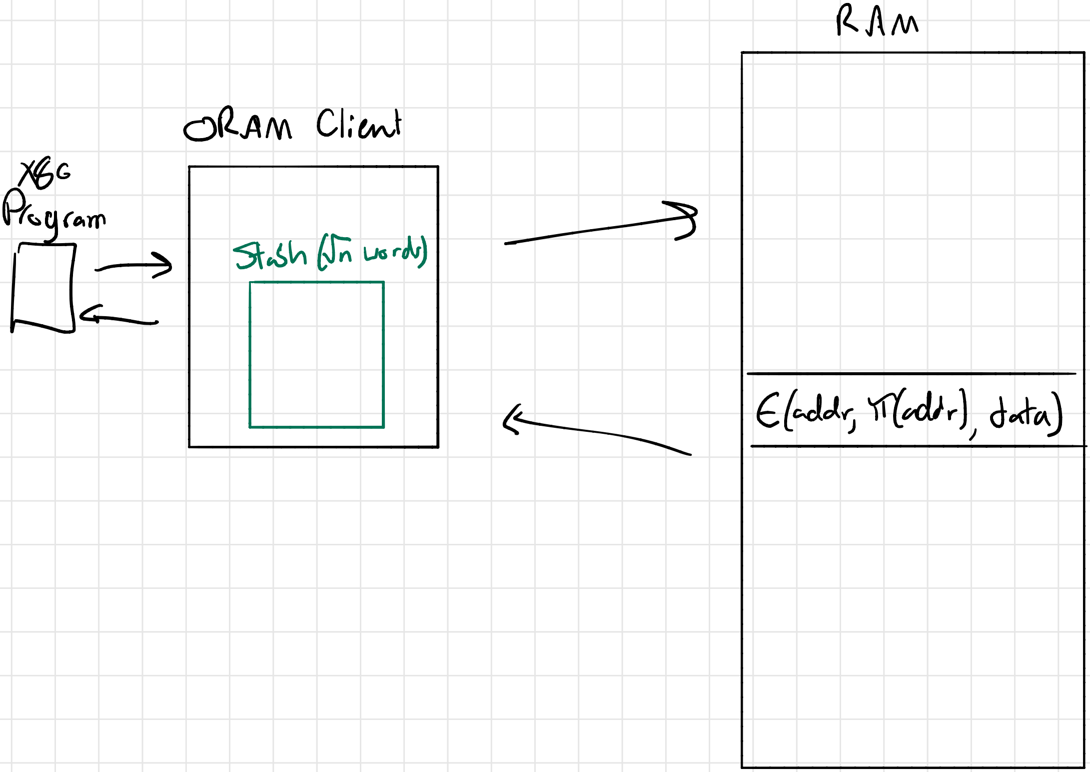
$O(n^2)$ time to sort n items

Better: Batcher sort

$O(n \log^2 n)$ time ... simple to implement

Better: AKS + Goodrich

$O(n \log n)$ time ... not so simple.



* Initialize RAM w/ all zeros. (encrypted)

* While true:

  - Shuffle memory locations according to random permutation $\pi : [n] \to [n]$

  - Process $\sqrt{n}$ R/W ops

    * if desired addr in stash: execute op on stash, read random addr in RAM

    * o.w. read desired element in RAM

    * Store result of read in stash

Square-root ORAM

Oblivious shuffle: $O(n \log^2 n)$ time

Each memory op requires 1 RAM access

$\Longrightarrow O(\sqrt{n})$ amortized cost per access

As described, client stores $O(\sqrt{n})$ blocks of stash

↳ Can store stash in RAM w/o affecting complexity by much ($\approx 3\times$).

(Read entire stash each time)

Stretch

Break?

Tree-Based ORAM

Developed in a long series of really nice papers. Relatively recent
↳ Shi, Chan, Stefanov, Li (2011)
+ work after

We will see "Simple ORAM" of Chung & Pass (2013)

Client storage: $O(\log^2 n)$

RAM storage: $O(n \log^2 n)$

Comp overhead: $O(\log^4 n)$ R/W to RAM per logical op.

Remember: More recent ORAMs give improvements in theory & practice. But this is simple.

Plan: 1. Construct a "bad ORAM" in which client stores $n/2$ blocks instead of n.
↳ $O(\log^3 n)$ comp overhead

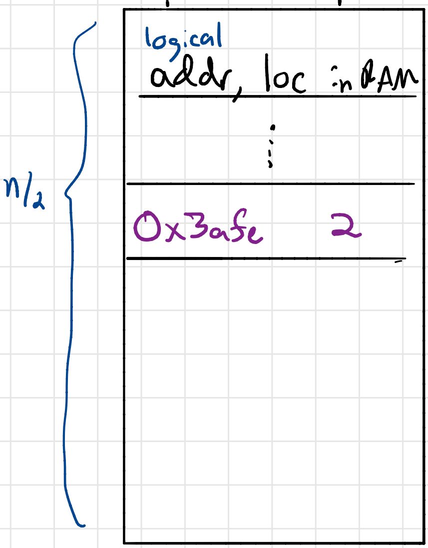
2. Recursively store the $n/2$ blocks in another ORAM... recurse all the way down

Overhead: $O(\log^3 n) \cdot \underbrace{\log n}_{\text{b/c of recursion.}}$

Only need to explain Step 1.

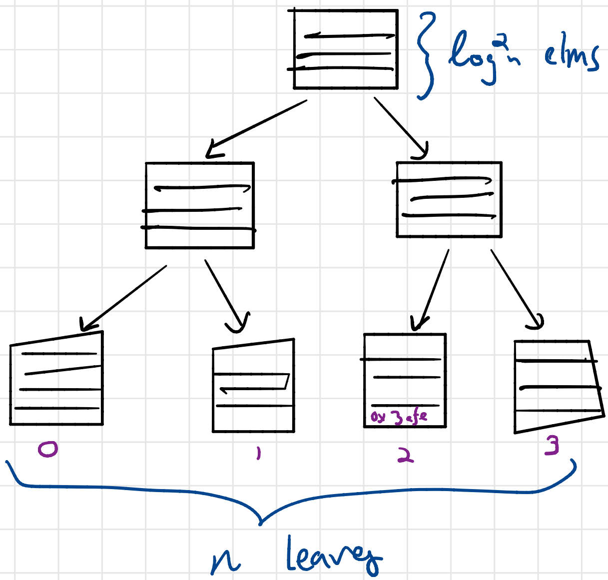
ORAM Client

"position map"



(Store pairs of logical addrs next to each other)

RAM



Invariant: data for addr is stored on path to leaf indicated in position map.

ORAM Operations

- Read & Write are essentially the same.
- Let's look at a read...

Read (addr)

1. Look up leaf l of addr in position map.
2. Read contents of all buckets on path to leaf l.
3. Pick a new random leaf $l' \xleftarrow{R} [n]$, PosMap[addr] $\leftarrow l'$.
4. Add (addr, data) ... encrypted to root bucket.
   ↳ If no space, fail.
5. Pick a leaf $l \xleftarrow{R} [n]$, walk down path from root to l.
   ↳ "flush" blocks down towards l as far as they can go while still maintaining The Invariant.
   ↳ If no space, fail.

That's it! So simple!

For writes, just update new contents before putting data back into tree root.

Properties

Correctness: As long as there's no overflow, all read/write ops return right answer. ✓

Security: On each R/W, client reads two random paths from root to leaves. ✓

Overhead:
- $n/2$ client storage
- Buckets have size $\log^2 n$, need to read/write $O(\log n)$ of them $\Rightarrow O(\log^3 n)$.
- Server stores $O(n)$ buckets $\Rightarrow O(n \log^2 n)$.

To show there's no overflow.

- Bound leaf overflow: Chernoff bound.
- Bound node overflow: Slightly more involved, but still not too bad.

↳ See Ross paper.

Summarizing

- ORAM lets a client outsource it's storage while hiding access patterns.
- Best constructions have logarithmic overhead (in # of ram accesses) & have varying levels of practicality.

Not sure whether any deployed systems have used ORAMs...

Can speculate on why not.