Lecture 8: More ORAM
Today

- Summary of survey

- Recap: Square-Root ORAM

- Tree-based ("modern") ORAM

---

**Logistics**

- HW3 out now

- Please answer questions on Piazza via app feedback form!
Summary of Survey

- Most people are familiar w/ Shamir
- Little coverage of other topics
- Likely plan: "Modern" MPC. 2k 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
  * 2kSNARKs from linear PCPs (GGPR, ...)
  * 2k 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

3. Use standard encryption (e.g. AES-GCM) and re-encrypt/re-randomize of every block.

⇒ Will show an even simpler √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 elsms in list, swapping order of elsms in wrong order
- Continue until sorted.

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

Better: Bucket 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

$\Rightarrow \Theta(\sqrt{n})$ amortized cost per access

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

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

(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) + web after.

We will see “Simple ORAM” of Chuy & Pass (2013)

Client storage: $O(\log^2 n)$
RAM storage: $O(n \log^2 n)$
Comp overhead: $O(\log^4 n)$ 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 $n$.
   $\Rightarrow O(\log^3 n)$ comp overhead

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

   Overhead: $O(\log^3 n) \cdot \log n$

   $\frac{1}{2}$ of recursion.

Only need to explain step 1.
### ORAM Client

```
logical addr, loc in RAM
```

| 0x3afe | 2 |

(Store pairs of logical addr next to each other)

### RAM

- n/2 leaves

- n leaves

**Invariant:** data for addr is stored on path to leaves indicated in position map.
ORAM Operations

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

\textbf{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 $l' \leftarrow \{rl\}$, 
   $\text{PosMap}[\text{addr}] \leftarrow l'$.
4. Add $(\text{addr, data})$ - encrypted to root bucket. 
   $\rightarrow$ If no space, fail.
5. Pick a leaf $l' \leftarrow \{r\}$, walk down path 
   from root to $l'$. 
   $\rightarrow$ "flush" blocks down towards $l$ as far as they can go while still 
   maintaining the Invariant. 
   $\rightarrow$ 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 is 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 $O(\log^3 n)$ of them $\Rightarrow O(\log^3 n)$.
- Server stores $O(n)$ buckets $\Rightarrow O(n\log^3 n)$.

To show there’s no overflow.
- Bound leaf overflow: Chernoff bound.
- Bound node overflow: Slightly more involved, but still not too bad.

See pass paper.
Summarizing

- ORAM lets a client outsource its 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.