Privacy Enhancing Technologies

lecture 6 : ORAM

Oblivious RAM

see Moodle

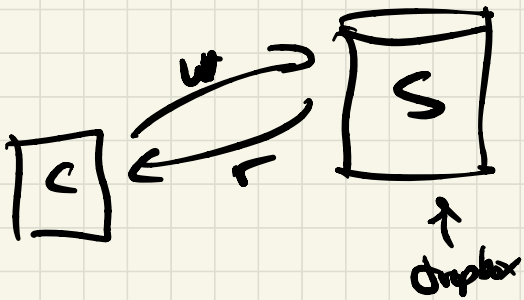
- PIR
- ORAM

} close to production
still very interesting research developments

ORAM: privately read & write to some database

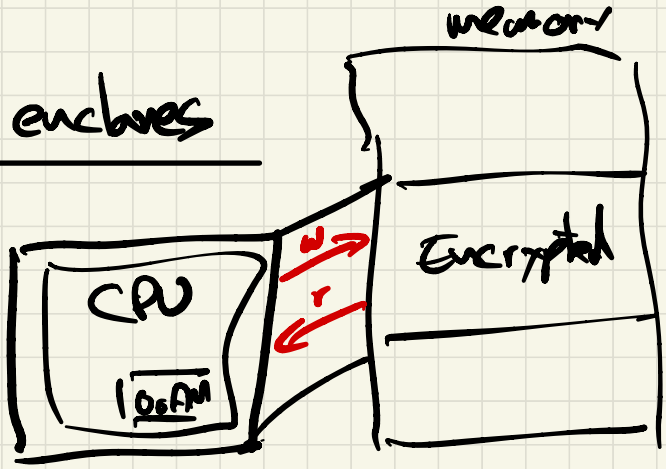
Examples

Private Dropbox



Hardware enclaves

Intel SGX

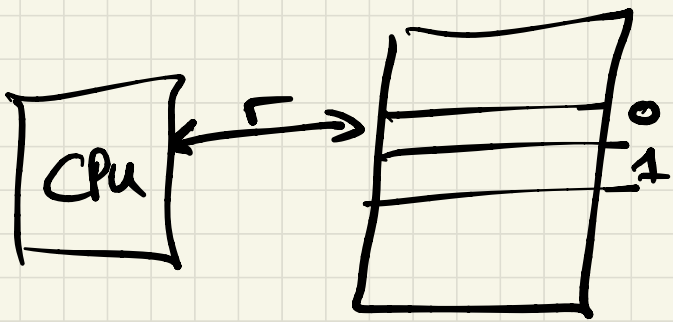


```
if (secret):
    read mem[0]
else
    read mem[1]
```

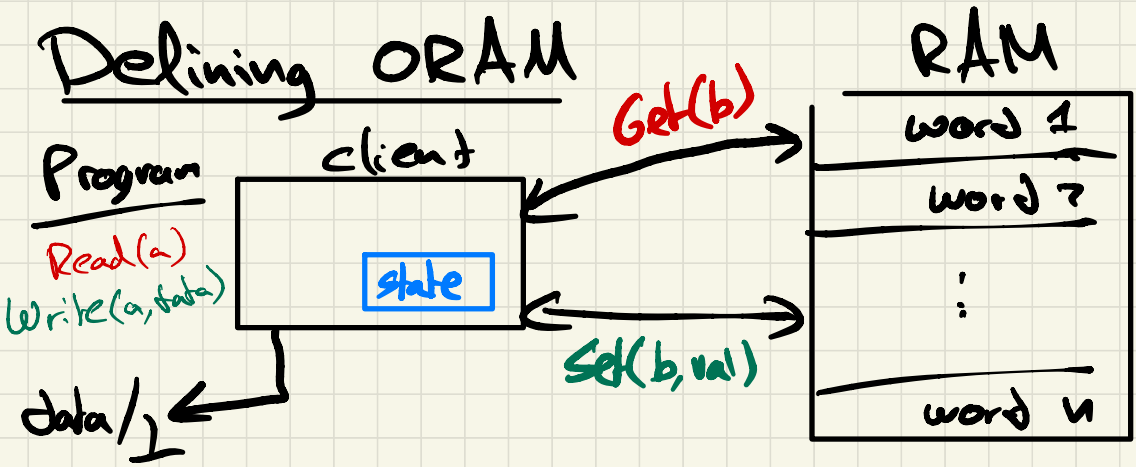
encrypted

encrypted

encrypted



Defining ORAM



Correctness: The program behaves correctly (as if it was directly accessing RAM)

Privacy: the access patterns to the RAM leak nothing about the program's reads & writes

Def: • An operation $op \in \{$ read(·), write(·) $\}$

• Access(op): set of RAM accesses made for op

Security/Privacy

read(0)

let $O = \{op_1, op_2, \ldots op_k\}$

$O' = \{op'_1, op'_2, \ldots, op'_k\}$

write(7, "hello")

$\{Access(op_1), \ldots, Access(op_k)\} \overset{c}{\approx}$

$\{Access(op'_1), \ldots, Access(op'_k)\}$

"Trivial", or naive solutions (RAM with n words)

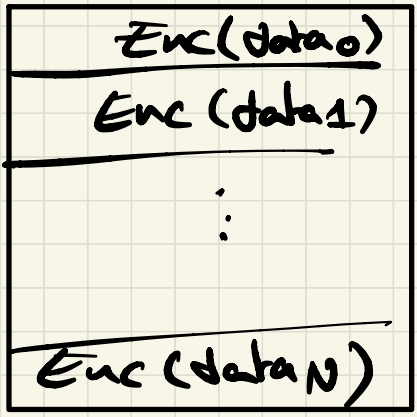
- local storage : n
- access overhead per op. : 0

↳ put all the RAM in the client storage

- local storage: $O(\lambda)$ bits + $O(1)$ words
- overhead: n RAM accesses per op.

read(i)
write(i, data) }

- read all the RAM one-by-one
- re-encrypt each word
- and re-write to the RAM



What we can do:

- n^{ε} local storage (for constant $\varepsilon > 0$)
- $O(\log n)$ accesses per op.

PIR vs ORAM

Private reads from public DB

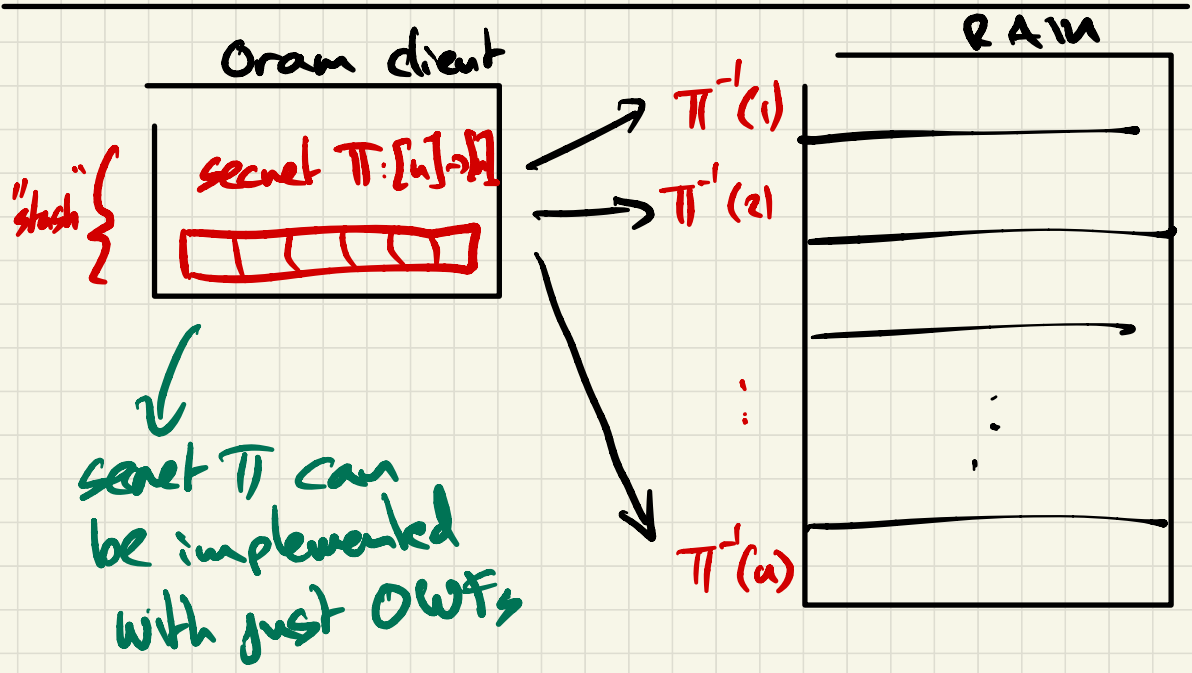
- DB is public & static
- Many clients ⟷ 1 server
- client is stateless
- You need PK crypto in single-server setting

Private reads & writes to private DB/RAM

- DB is private and changes w. every op
- 1 client ⇔ 1 server
- client must have state
- Can be built from just OWFs!

A $\sqrt{n}$ Construction

* local storage : $O(\lambda)$ bits $+O(\sqrt{N})$ words
* overhead per op : $O(\sqrt{n})$ (amortized)



secret Π can be implemented with just OWFs

- Read(i), Write(i, data) $\Rightarrow$ read $\Pi(i)$ & write $\Pi(i)$
- Store (i, data) in local stash

- read/write something that is already in the stash:
  - do the op on the stash
  - do a random read/re-write on RAM

- this works until your stash is full ...
  - ↳ say you evict (i) from stash
  - ↳ next time you have to access (i) you leak to the RAM that you're re-accessing the same memory

- when the stash is full, we re-shuffle the entire RAM 😢

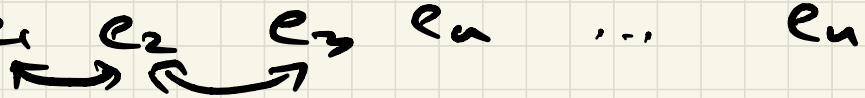
  ⇒ How to do this with little overhead?

Oblivious sorting

How to sort data with uniform access patterns

Bad example: merge sort

Inefficient example: Bubble sort



$O(n^2)$ oblivious sort

Ideal: oblivious $O(n(\log n)$ sort

Practical: Batcher sort $O(n \log^2 n)$ oblivious

The protocol

init:
- client inits a stash of $\sqrt{n}$ words
- RAM is init. to $Enc(0)$

while (true):

- client sample a permutation Π
- client shuffles the RAM according to Π
  ↳ empty the stash
- Process $\sqrt{n}$ ops:
  - if addr is in stash: do a random RAM read and store
  - otherwise, read addr to stash
  - if write, do it on the stash

Per $\sqrt{n}$ ops, overhead is:

- once: $O(n \log^2 n)$
- $\sqrt{n}$ times: 1

$\Rightarrow$ Amortized overhead per op. is $\tilde{O}(\sqrt{n})$

↖ ignoring $\log(n)$ factors

Tree-based ORAMs

↳ $O(\text{poly}\log(n))$ overhead

↳ Here: "Simple ORAM"
$O(\log^4 n)$

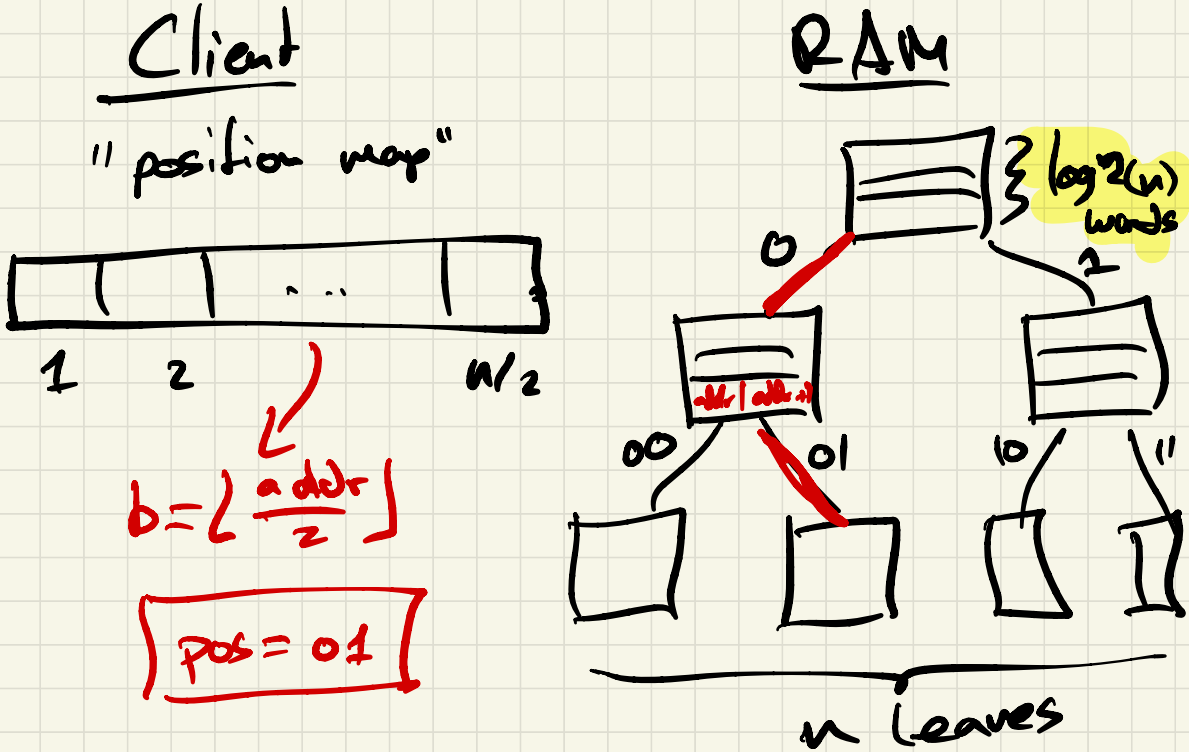
How this works

1) build a "bad" ORAM with stash $\frac{n}{2}$ and $O(\log^3 n)$ overhead

2) Recurse !! store the stash in another ORAM

3) Repeat $\log n$ times
   - constant stash
   - overhead $O(\log n) \cdot O(\log^3 n)$

building - BAD ORAM



Invariant: The data at address a is somewhere on the path from the root to the leaf node indicated at position $\lfloor a/2 \rfloor$ in the position map

How this works?

Read (addr):

- look up pos at element $[\frac{addr}{2}]$ in the position map
- read $\log^2(n) \cdot (\log(n)$ elements of RAM from root to leaf pos
  ↳ [addr : val , addr+1 : val']
- pick a new random leaf pos' and update position map $[\frac{addr}{2}] \to pos'$
- re-encrypt (addr, val) (addr+1, val') and insert this into root node
- if overflow: walk down the path from root to pos' and "flush" all elements down the tree
  ↳ if this overflows, abort ☹