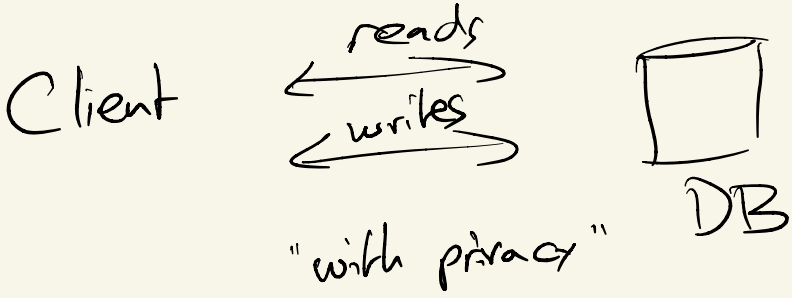
Privacy Enhancing Technologies

Oblivious RAM

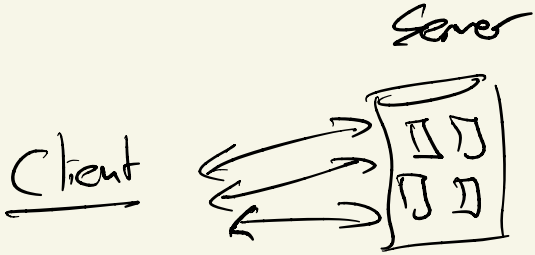
ORAM $\approx$ PIR for "writing"

not quite



Motivation

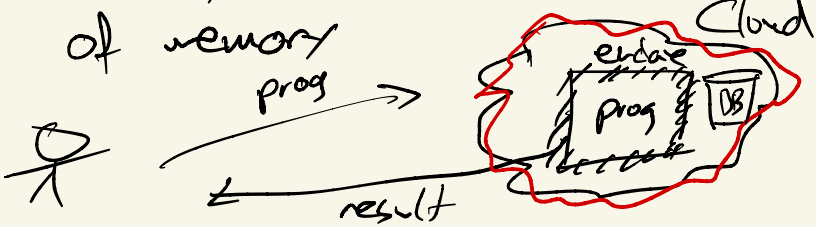
1) private Dropbox



hide content of files and access patterns

2) Hardware Enclaves

Idea: run programs on an untrusted machine, within a "secluded" area of memory

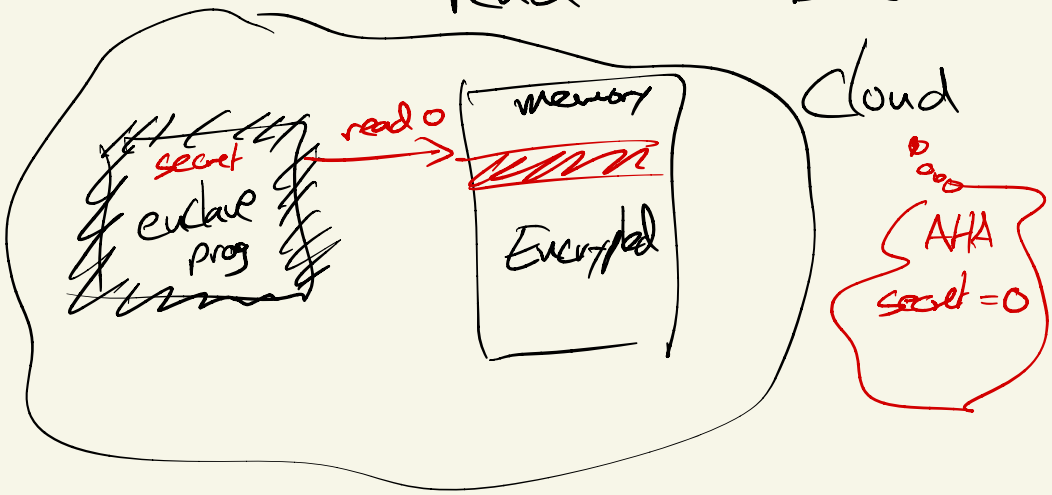


program: if (secret = 0):

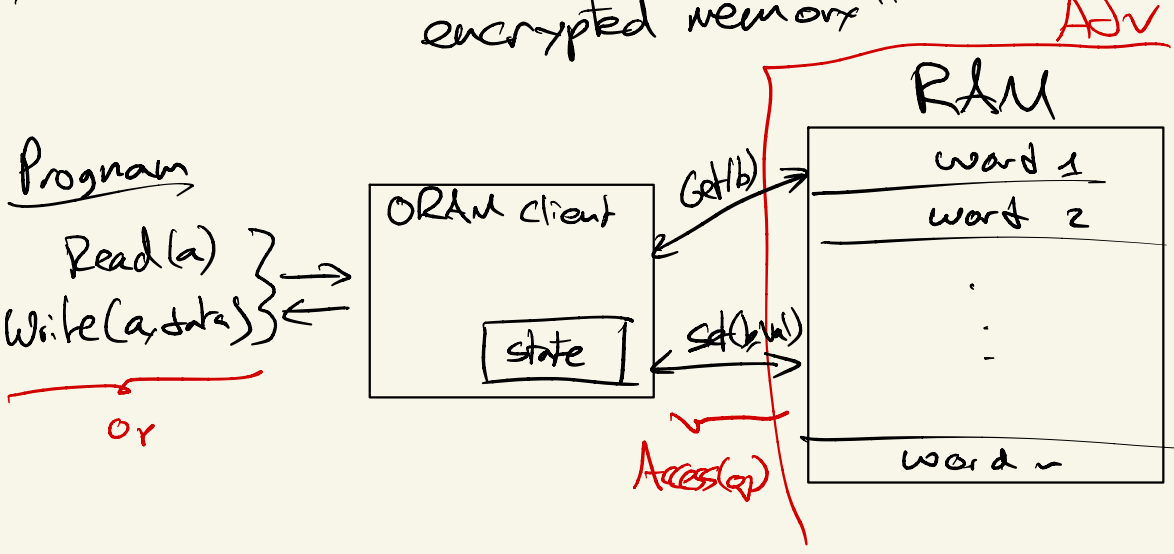
read mem[0]

else

read mem[1]



ORAM : "hiding access patterns to an encrypted memory"



Properties

- Correctness: the program behaves as if it was interacting with the RAM directly
- Security: the access patterns to the RAM should not leak anything about the reads/writes made by the program

Formally:
- An $op \in \{Read(\cdot), Write(\cdot,\cdot)\}$
- $Access(op)$ is the set of accesses made by the ORAM client to the RAM when servicing op

- Correctness: For any sequence $O = \{op_1, \dots, op_k\}$ from the program, the ORAM client answers each op correctly
- Security: let $O = \{op_1, \dots, op_k\}$, $O' = \{op_1', \dots, op_k'\}$ where $k = poly(\lambda)$ then

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

Trivial solutions

- No outsourcing: The ORAM client stores the whole RAM in its private state

  → $|state| = n$ words

  → Overhead = 0 RAM accesses per op.

- Linear Scans: for every op, the ORAM client will read and reencrypt every word of RAM

  – $|state| = O(1)$ (store an encryption key)

  – Overhead per op is N RAM accesses

Goal: $|state| = o(N)$

$|overhead| = o(N)$

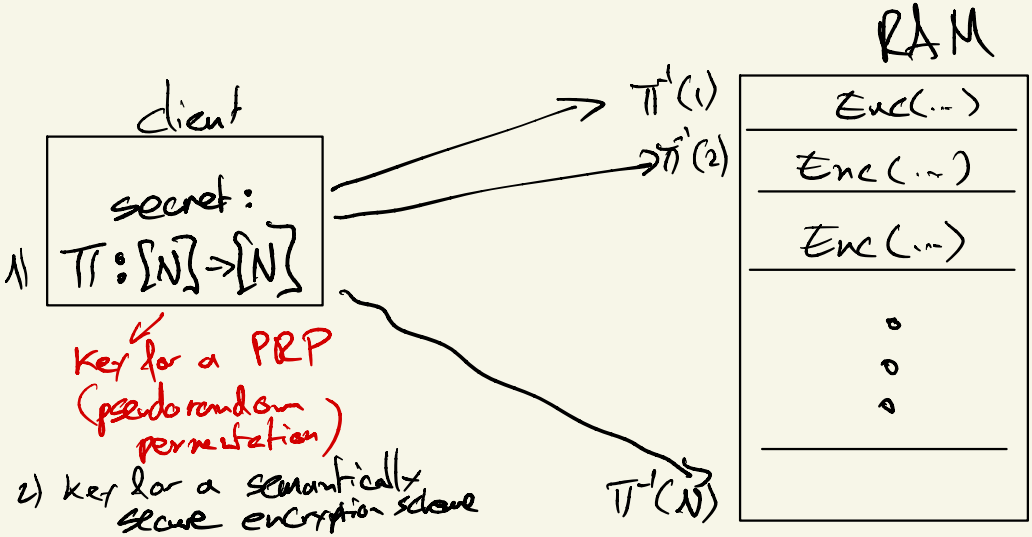
Lower bound: in an "online" setting, you need $O(\log N)$ RAM accesses per op. if $|state| = N^{\varepsilon}$ for $\varepsilon > 0$

Comparing ORAM to PIR:

PIR	ORAM
DB is public / static	RAM is private and dynamic
one DB ⇐ Many clients	one RAM ⇔ one client
Server has to do $O(N)$ work per query	RAM has $O(\log N)$ overhead
Client is stateless	client with private state
single server PIR "needs" PKC assumptions	Can be built from PRFs

---

A $\sqrt{N}$ overhead construction:



Insight: If we have $k \leq N$ ops that access distinct RAM addresses, we can just execute them.

The view of the RAM:

read(a_1)/write(a_1) ... read(a_k)/write(a_k)

where $a_1, \ldots, a_k \xleftarrow{\$} [N]$ without replacement

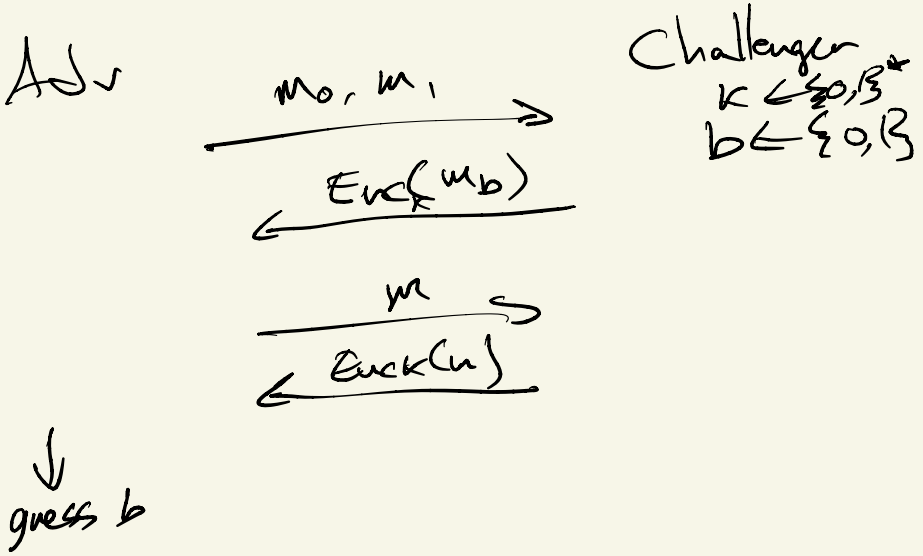
Program

read(0)
write(1)
read(2)

ORAM Client

$a \leftarrow$ read($\pi(0)$)
write($\pi(0)$, Enc(a))
read($\pi(1)$)
write(...

Semantic security for Enc



How to handle arbitrary programs?

- The client keeps a private "stash" of $\sqrt{N}$ words initialized to $\perp$
- The RAM is initialized to 0 (or to encryptions of something)

while (True):

- the client picks a new random permutation Π and stores it in private memory
- The client shuffles the entire RAM according to Π in an oblivious manner. While doing this, flush out the stash to the RAM. $O(N \log N)$ accesses
- Process $\sqrt{N}$ ops: read(a)/write(a)
  1) if a is in the stash, access a random location of RAM and discard it
  2) otherwise, read $\Pi(a)$ into the stash $O(1)$ accesses
  3) if the op is a write, do it in the stash

Amortized cost per op: $O(\sqrt{N} \log N)$

Example of a non-oblivious shuffle

```
val ← read(σ)
write(π(σ), val)
:
```

You can sort (which implies shuffling) an array of N elements in a way that the access pattern to the array is independent of the array elements.

ex: merge sort : not "in place" / not oblivious
quick sort : not oblivious $O(n \log n)$
bubble sort : oblivious!
$O(n^2)$ worst case
:(

We'd like: better than $O(n^2)$ sorting that is oblivious

For example:
- Batcher sort $O(n \log^2 n)$
- More complex ones $O(n \log n)$

$O(n \log n)$ oblivious sorting $\Rightarrow$ $O(n \log n)$ oblivious shuffling

How: pick π

write value $\pi(i)$ (or $\pi^{-1}(i)$: don't know) into word i

sort
