FHE

Client	Server
x, sk	f, pk
$ct = Enc_{pk}(x)$	$f' = Circuit(f)$

$$\xrightarrow{ct}$$

$ct' = Eval_{pk}(f', ct)$

$$\xleftarrow{ct'}$$

$y = Dec_{sk}(ct')$

x not learned
(apart from length)

To avoid the trivial soln. we insist that

$\exists$ polynomial p s.t. $|Circuit(Dec_{sk}(\cdot))| = p(|sk|)$

$\implies$ $|ct'|$ independent of f

$\implies$ Client work independent of f

Notice: Client work *does* depend on $|x|$

Our setting

Client	Server
f	x

memory independent of x. in particular constant.

$$\xleftrightarrow{\text{Some interaction}}$$

f not learned
(apart from runtime*)

$y = f(x)$

*explain in a moment.

Tempting Solution

Client (f) | Server (x)

$sk \leftarrow \$$

$f' = Obf(Enc_{sk}(f(\cdot)))$

Does not exist!

Very Slow

$\xrightarrow{f'}$

$ct = f'(x)$

Also very slow!

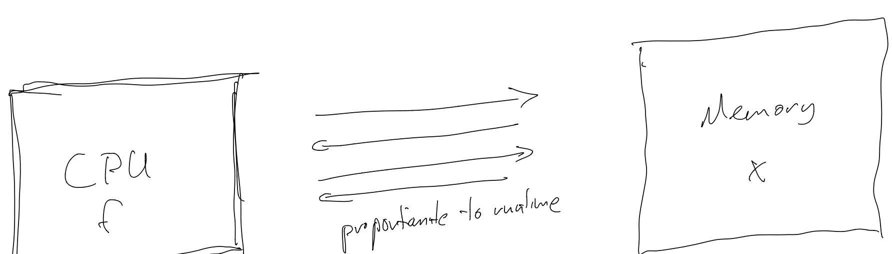
$\xleftarrow{ct}$

leaks output size

$y = Dec_{sk}(ct)$

Instead

Client (f) | Server(x)



$y = f(x)$

* learns "runtime"

= number of accesses

= max required memory

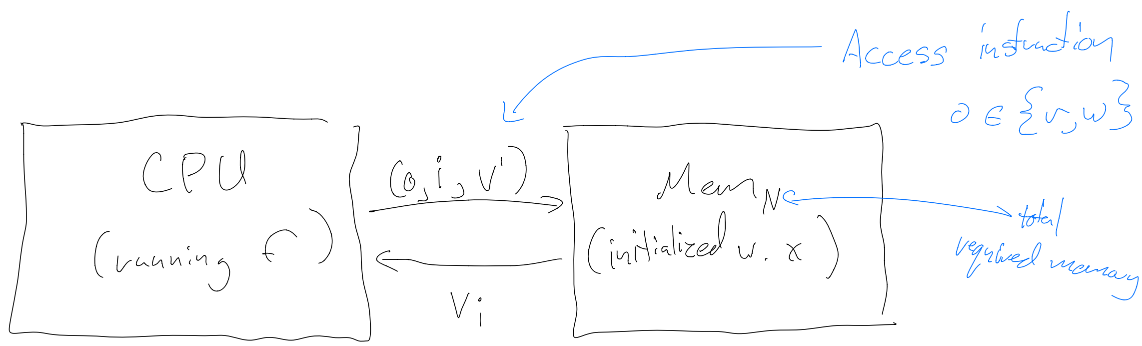
Model:

CPU has only a few registers (maybe const.)

f looks like a program on your computer

operations on register values

read/write to memory in order to store intermediate values.

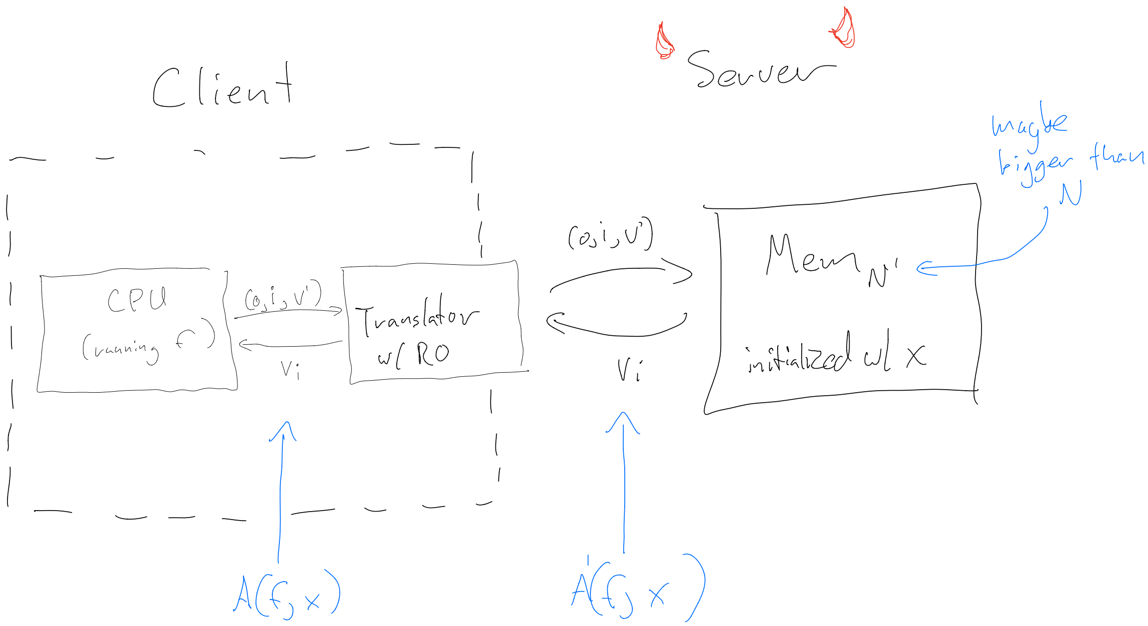


What can a sequence of reads/writes reveal?

Everything! Many algs have characteristic access patterns. Secrets could be baked into f; x could be an encryption, and accesses leak the plaintext.

ORAM $\triangleq$ Oblivious Random Access Machine/Memory let $A(f, x) = a_1, a_2, \ldots$

s.t. a_j is the j^{th} access instruction



Security

$$\forall f^1, f^2, x^1, x^2$$

$$|A(f^1, x^1)| = |A(f^2, x^2)| \Longrightarrow A'(f^1, x^1) \approx A'(f^2, x^2)$$

Alternately, $\exists$ Sim s.t. $\forall f, x$

$$A'(f, x) \approx Sim(|A(f, x)|)$$

Discuss Nomenclature

Discuss universal f, hiding x

Discuss Memory contents, hiding $\overbrace{o \text{ and } v' \text{ and } v_i}^{\text{intuitively easy}}$

An oram client has an overhead g if $\forall f, x, T$ s.t.

$$|A(f,x)| = T \quad , \quad |A'(f,x)| = g(T) \cdot T + c$$ ← initialization! might depend on $|x|$

To Avoid trivial solutions we want $g(T) < T$

strict less than

Also, the translator must use $< N$ cells of local storage.

Simple and intuitive ORAM formula:

At the beginning of each epoch, obliviously permute the memory randomly

To access element i:

stash

access every element you have previously accessed this epoch.

if you have already found element i

dummies help!

access an untouched element at random

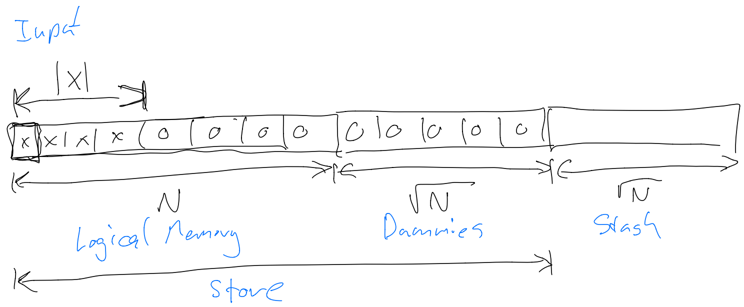
else

compute the permuted location of i and access it.

At the end of each epoch, depermute the memory

Square Root ORAM

Mem:



CPU: ectr (reset each epoch) dctr (reset each epoch) sk (random once) k (random once per epoch)

+ space for two memory elements

(Enc, Dec) - randomized, symmetric

F - PRF

Initialization:

Add metadata $O(N)$

Replace each j^{th} memory cell v with

$(\tilde{\imath} = \text{Enc}_{sk}(j), \tilde{v} = \text{Enc}_{sk}(v), b = \text{false})$ ← virtual memory cell i

logical addr — Data — used bit

Epoch Start

Choose prf key k uniformly

Sort Mem + Dummies using airwise compares on

$F_k(i_1)$ and $F_k(i_2)$

AKS: $O(N \log N)$ Batcher: $O(N \log^2 N)$

let ectr = dctr = 0

Access (i, o, v'):

For each $(\tilde{i}, \tilde{v})$ in stash: $O(\sqrt{N})$

if $Dec(\tilde{i}) = i$:

keep $v_{out} = Dec(\tilde{v})$ as output

if o = write, encrypt (i, v') and write back

else reencrypt (i, v_{out}) and write back

else

re-encrypt and write back

$O(\sqrt{N})$

If v_{out} was found:

let $i' = F_k(N + dctr)$

$dctr \mathrel{+}= 1$

else

let $i' = F_k(i)$

Binary search store. For each node $(\tilde{i}, \tilde{v}, b)$ $O(\log_2(N))$

if $i' = F_k(Dec(\tilde{i}))$

if v_{out} not found, let $v_{out} = Dec(\tilde{v})$

update b = true 1

re-encrypt and store $(\tilde{i}, \tilde{v})$ in stash 1

Epoch End

Move each stash element into a store space where b=true $O(N)$

Sort Mem + Dummies using pairwise compares on

$Dec(\tilde{i}_1)$ and $Dec(\tilde{i}_2)$ $O(N \log N)$

Correctness

Amortized Complexity

Cost of epoch $O(N \log N)$

$\div$ Logical accesses $O(\sqrt{N})$

$=$ Overhead $O(\sqrt{N} \log N)$

Security

Permutation is indist from uniform = physical addrs are uniform

Perm is applied obliviously

Binary Search has deterministic access given target addr.

Each physical addr is searched ≤ 1 time

on each access, 1 store item is searched

every stash item is visited once, in order

1 new item is added to stash

Where Does the unsatisfying overhead come from?

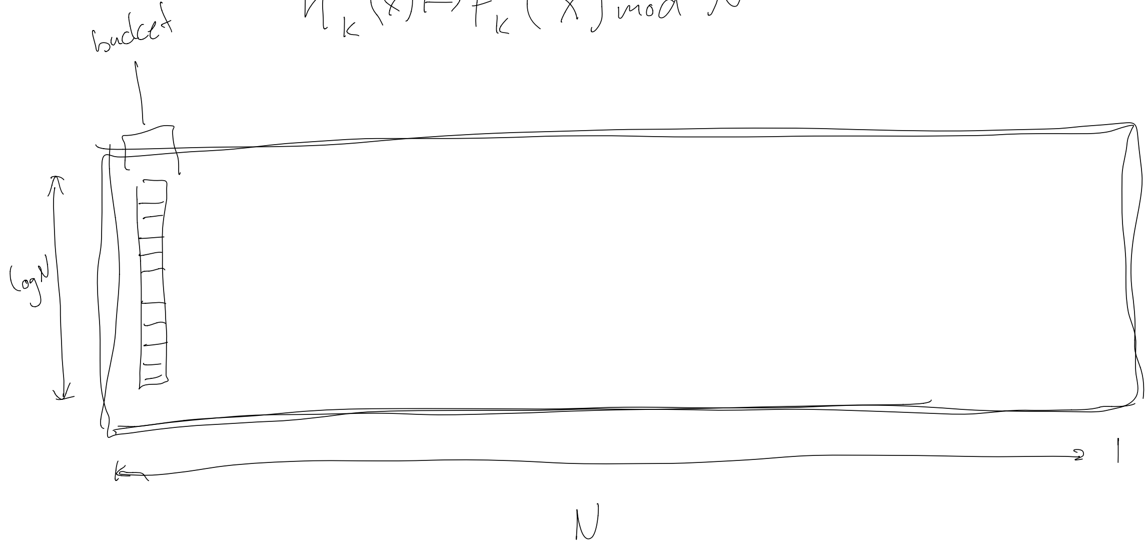
Observe, it takes $\sqrt{N}$ time to scan the stash, but we only need ≤ 1 element. What if the stash were in a smaller ORAM? Then we could make it larger and refresh less frequently

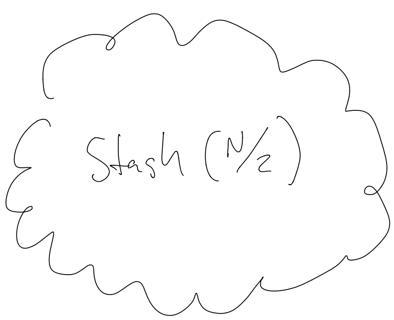
Problem: this one does not handle sparseness well.

Soln: modify construction to handle sparse indices

Hierarchical ORAM

$$h_k(x) \mapsto F_k(x) \bmod N$$





API: request index i
(get back V or $\perp$; index i is erased)
save (i, V)

Init / Beginning of epoch
Hash all elements into buckets by logical index $\underbrace{O(N \log^2 N)}_{\text{GO show how}}$
Put indexless Dummies in all free space.

Access (i, o, v')
Look for element i in stash and remove if it exists. $O(\text{stash}(N/c))$
let $i' = i$ if element i not found, else i' = next ctr

Compute $h_{k, \log N + 1}(i')$, scan bucket. Re-encrypt every value, and if element i is found, replace it with a dummy. $O(\log N)$

Save old or new value to stash as appropriate.

End of Epoch

Collect stashed values and remaining N/2 stored values, remove dummies, begin again

Amortized Complexity

Cost of epoch $O(N \log^2 N + N \cdot (\log N + \mathrm{Stash}(N/2)))$

$\div$ Logical accesses $O(N)$

= Overhead $O(\log^2 N + \mathrm{Stash}(N/2)) = O(\log^3 N)$

what goes here?

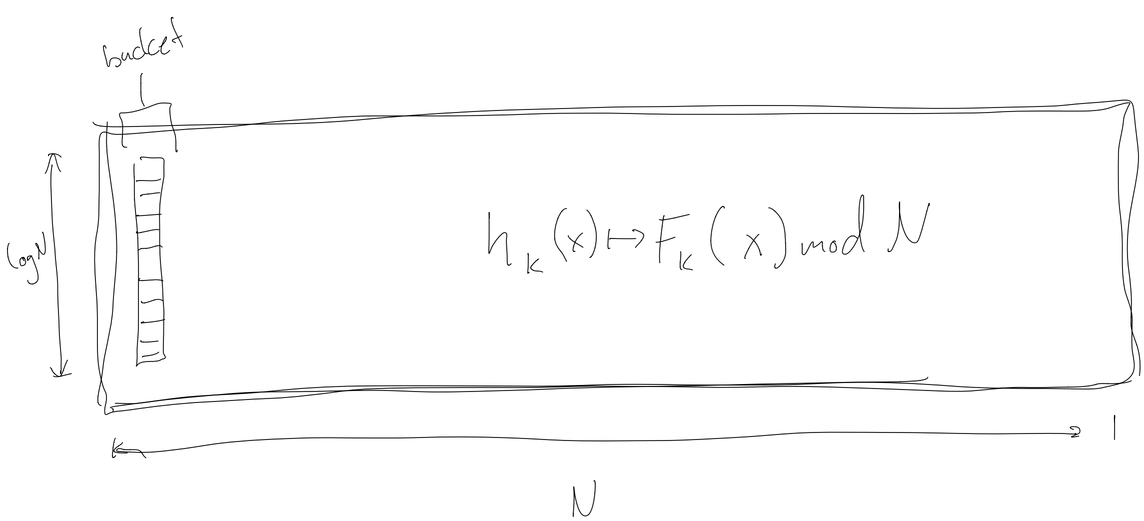
why, the same!

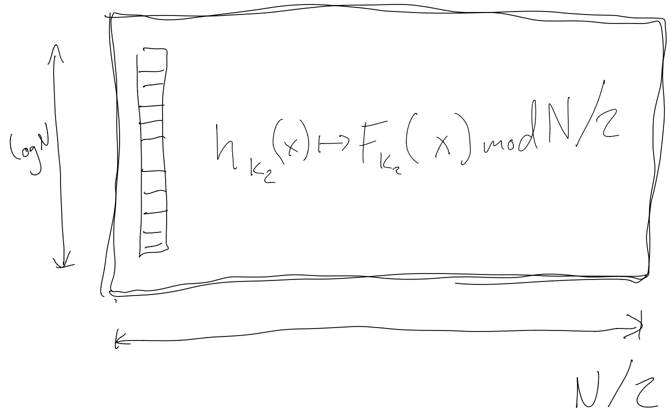
Base case: const. buckets. Always scan.

Security Note: Now we can use arbitrary sparse indices.

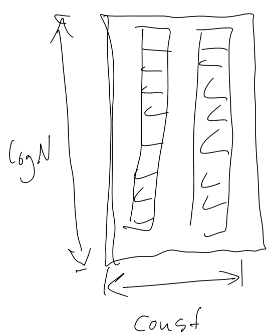
Querying an index not stored looks like querying one that is.

We must still touch no index more than once per epoch.





Stash 1

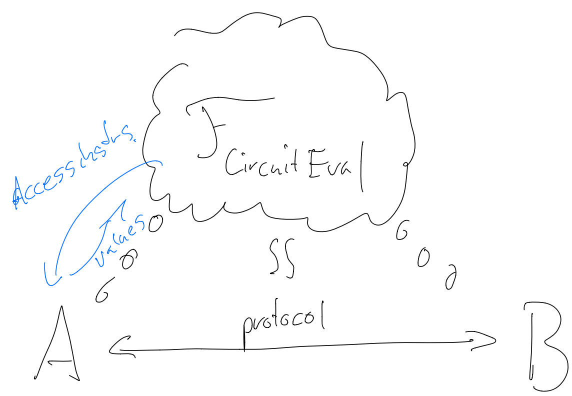


Stash log N

Overflow Probability:

$$\Pr[\log N \text{ collision}] \leq \binom{N}{\log N} \left(\frac{1}{N}\right)^{\log N - 1}$$

Application: MPC



Asymptotic efficiency gains. MPC sublinear in input size!

"Memory Gates"

Hide computed function's description?

Application: FHE

open for a while... until