Onion ORAM: Constant Bandwidth ORAM with Server Computation

Chris Fletcher

Joint work with:

Ling Ren, Marten van Dijk, Srini Devadas

Current art and where to go next

- **State of the art schemes**
  - Bandwidth: O(log N)
  - Client storage: O(1) (Path ORAM = O(log N))
  - Server storage: O(N)

- **Is "optimal" ORAM possible?**

  O(1) bandwidth, O(1) client storage, O(N) server storage

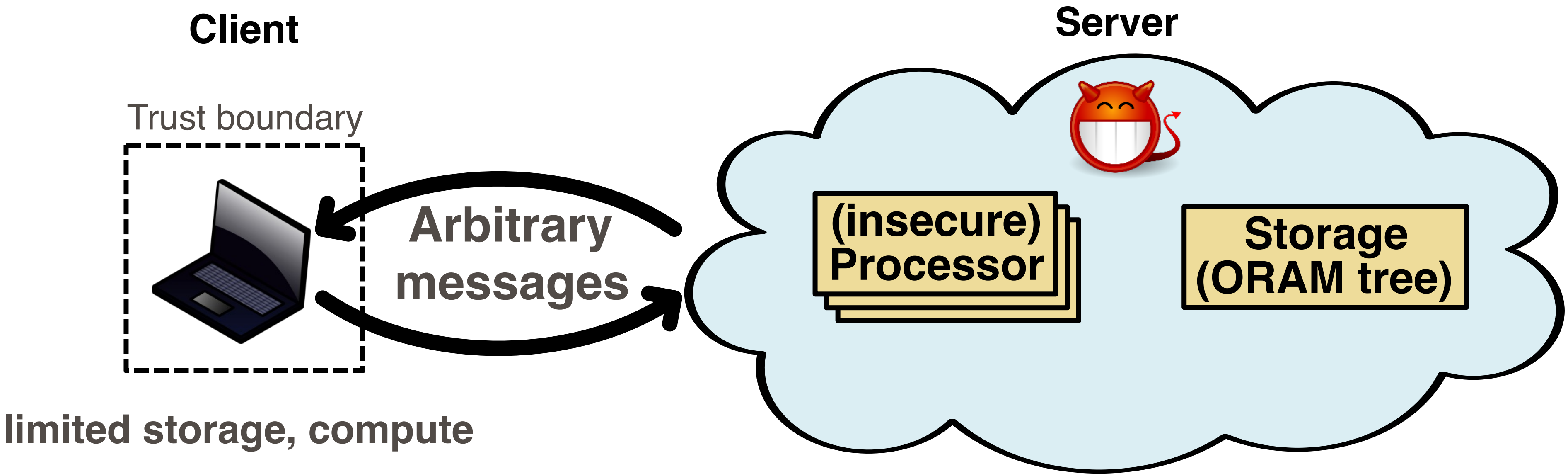
- **Goldreich-Ostrovsky lower bound [1987, 1996]**

  Given a program that runs in T time and an N block ORAM with O(1) client storage, the program+ORAM must run in $\Omega(T \log N)$ time

- **$\Omega(T \log N)$ doesn't mean $\Omega(T \log N)$ bandwidth!**

ORAM with Server Computation

- Example: Outsourced storage (Honest but curious)



- "Read X, Y, Z, return F(X, Y, Z)"
- Message stream must be oblivious

Server comp. in previous ORAMs

- **XORing reads [Dautrich et al.], PIR+ORAM [Mayberry et al.]**

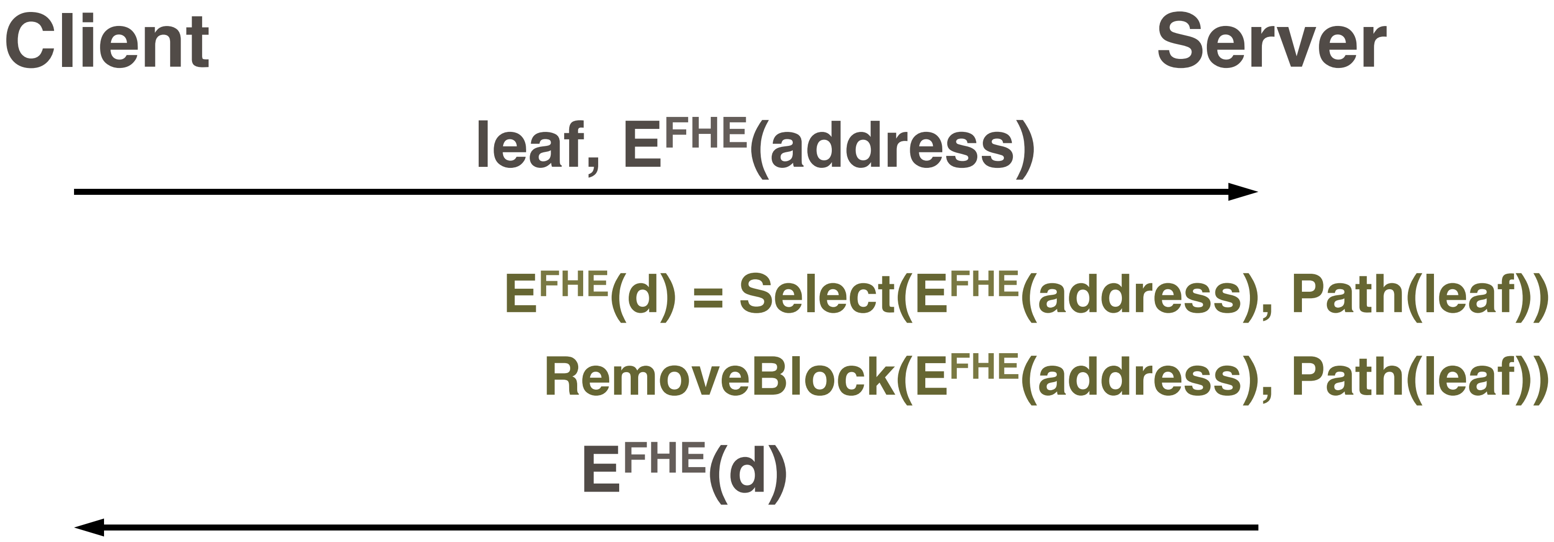
- **XOR + Ring ORAM**
  - Permuted buckets → one real block touched / read
  - **B**, d1, d2, d3, …
  - **E(B, r)**, E(0, r1), E(0, r2), E(0, r3) …
  - **Server sends: E(B, r)** ⊕ E(0, r1) ⊕ E(0, r2) ⊕ E(0, r3) ⊕ ...
  - **Client computes:** E(0, r1) ⊕ E(0, r2) ⊕ E(0, r3) ⊕ ...

- **Both schemes make read bandwidth O(log N) → O(1)**
- **Does not help on evictions!**

Can we make evictions O(1) Bandwidth?

FHE + ORAM

- **Example: Ring ORAM**
  - ORAM on server is encrypted under FHE scheme E^{FHE}
  - Reads



Client

Server

leaf, E^{FHE}(address)

$E^{FHE}(d)$ = Select(E^{FHE}(address), Path(leaf))

RemoveBlock(E^{FHE}(address), Path(leaf))

$E^{FHE}(d)$

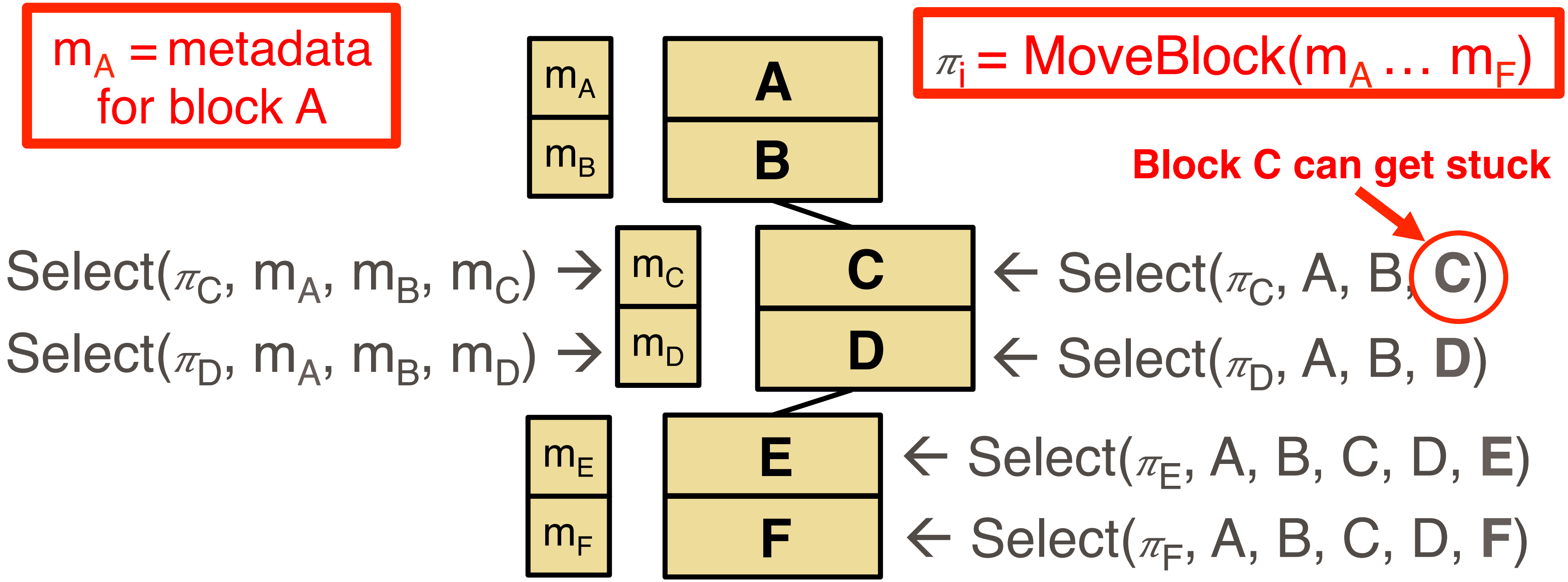
  - Evictions

Path(leaf_g)' = EvictPath(Path(leaf_g))

Read bandwidth is O(1), no bandwidth for evictions!

Eviction circuit for FHE

$$\text{Path}(\text{leaf}_g)' = \text{EvictPath}(\text{Path}(\text{leaf}_g))$$



m_A = metadata for block A

π_i = MoveBlock(m_A … m_F)

Block C can get stuck

Select(π_C, m_A, m_B, m_C) → m_C

Select(π_D, m_A, m_B, m_D) → m_D

C ← Select(π_C, A, B, **C**)

D ← Select(π_D, A, B, **D**)

E ← Select(π_E, A, B, C, D, **E**)

F ← Select(π_F, A, B, C, D, **F**)

- **Only Select() touches *blocks***

Server computation: polylog(N)

Bootstrap to manage noise [Apon et al., Mayberry et al.]

*Discuss later: Does the previous scheme achieve optimal Bandwidth/storage?

Do we need bootstrapping?
Do we need FHE?

Do we need FHE?

- **Additive-HE (e.g., Paillier)**
  - Addition: $E^{AHE}(a) \oplus E^{AHE}(b) = E^{AHE}(a + b)$
  - Scalar multiplication: $E^{AHE}(a) \otimes c = E^{AHE}(ca)$

- **Select from (X, Y):**

$$E^{AHE}(0) \otimes X \;\oplus\; E^{AHE}(1) \otimes Y \;=\; E^{AHE}(0+Y) = E^{AHE}(Y)$$

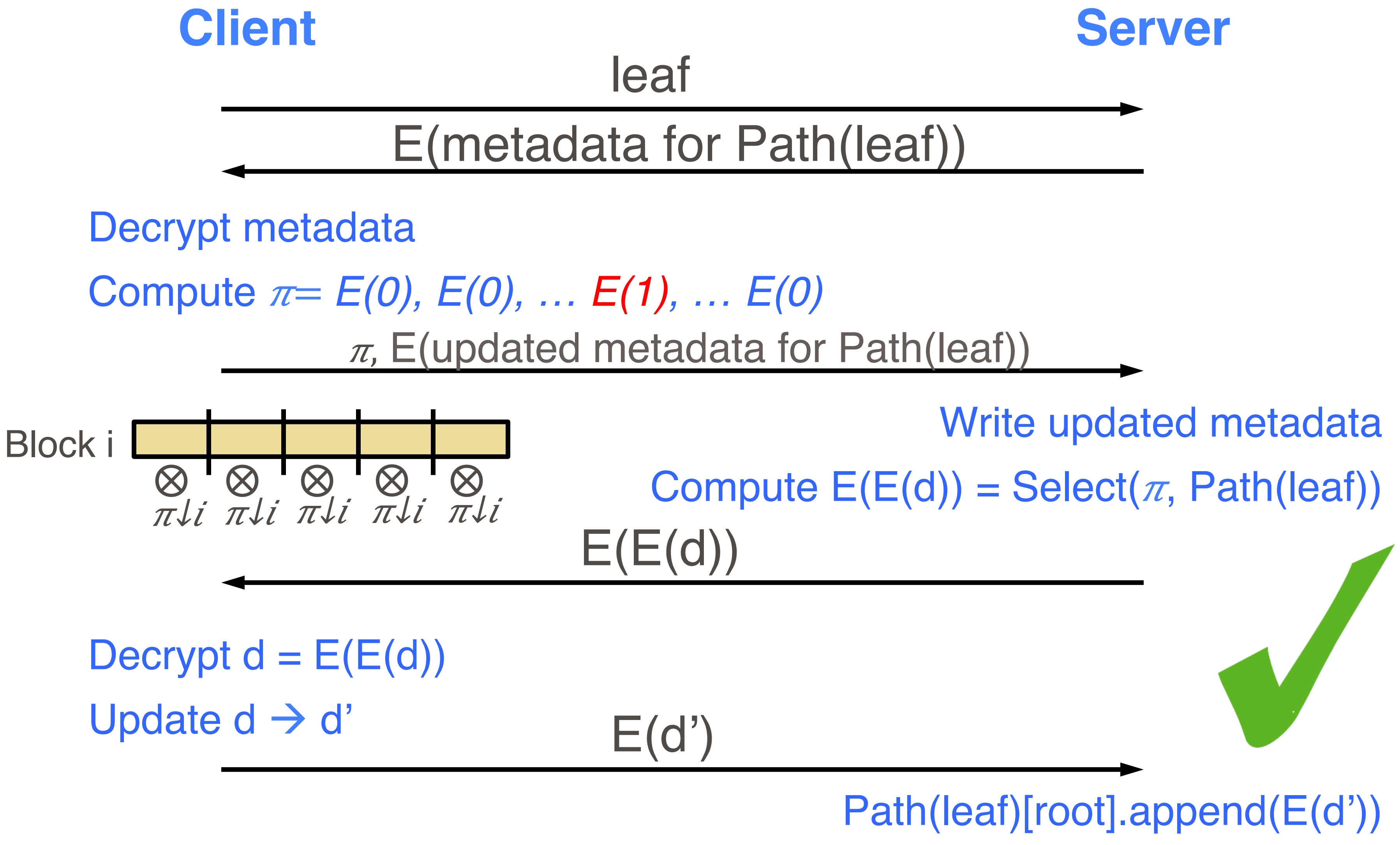
- **$Y = E^{AHE}(\text{plaintext})$**
- **Select op → $E^{AHE}(E^{AHE}(\text{plaintext}))$**
  - Client decrypts twice
  - (Possible) ciphertext blowup per layer

Block gets extra layer of encryption

- **$\text{Layers}(\text{output}) = \mathit{max}(\ \text{Layers}(\text{Block}_i) : \text{Blocks}\ ) + 1$**

ORAM Read + Additive-HE

- **ORAM encrypted using 1 layer of E^{AHE} (abbreviated E)**



Client **Server**

leaf →

← E(metadata for Path(leaf))

Decrypt metadata

Compute $\pi = E(0), E(0), \ldots$ $E(1)$, $\ldots$ $E(0)$

π, E(updated metadata for Path(leaf)) →

Write updated metadata

Compute E(E(d)) = Select(π, Path(leaf))

← E(E(d))

Decrypt d = E(E(d))

Update d → d'

E(d') →

✓

Path(leaf)[root].append(E(d'))

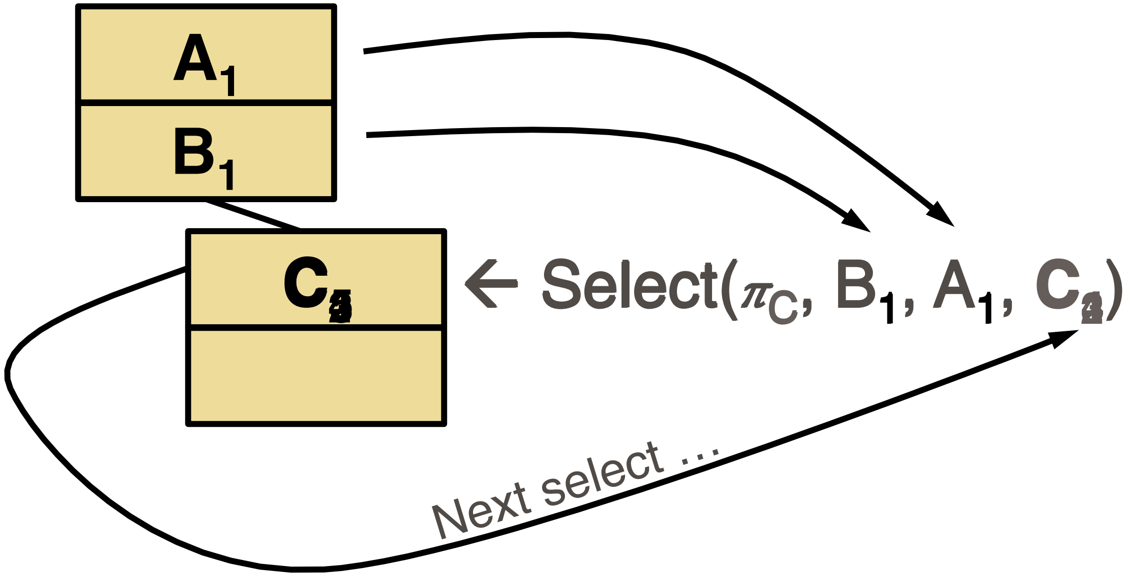
ORAM Evict + Additive-HE

- **Problem: Continuous reshuffling → Unbounded layers**

- **Reason: Blocks can get stuck in buckets after evictions**

Layers(output) = max(Layers(Block_i) : Blocks) + 1

A_1 = Block A with 1 layer



← Select(π_C, B_1, A_1, C_4)

O(T) evictions → Slot with C gets O(T) layers

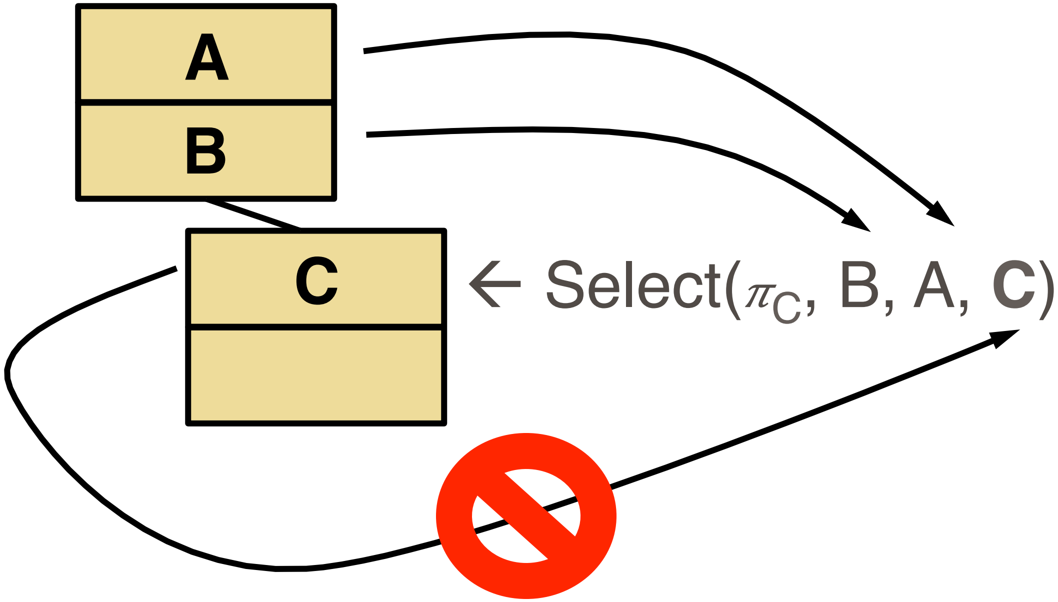
ORAM with $O(1)$ bandwidth,
$O(1)$ client storage,
$O(N)$ server storage

…with only additive-HE

Onion ORAM

Design our ORAM eviction algorithm such that

buckets are guaranteed to be empty regularly



← Select(π_C, B, A, **C**)

Onion ORAM

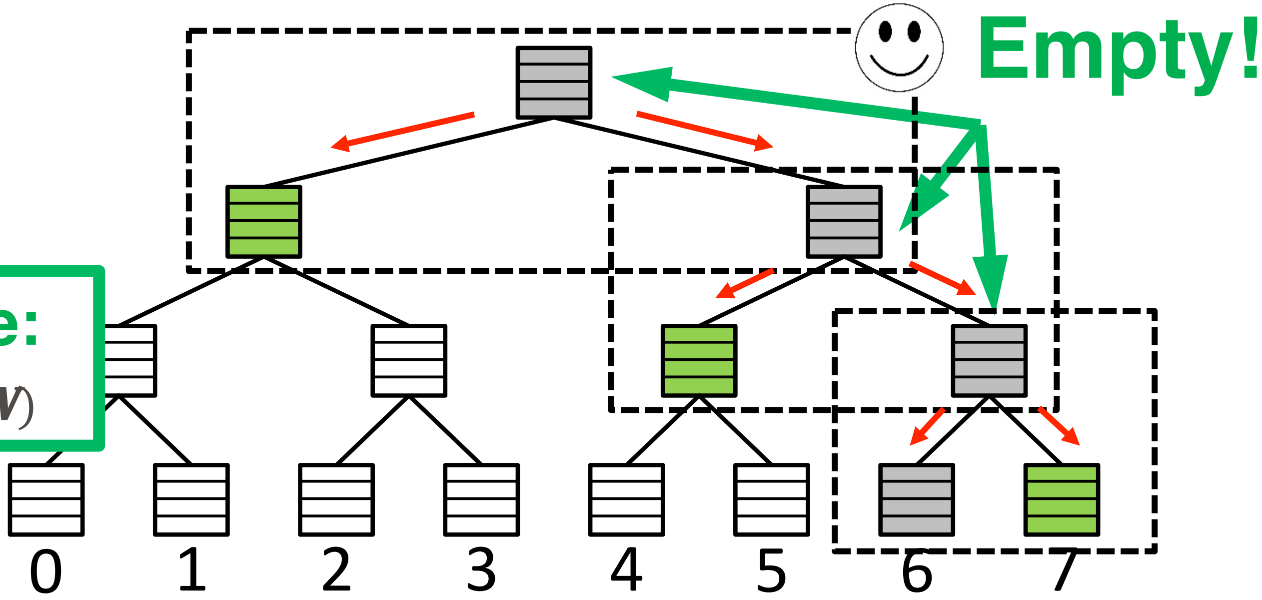
Design our ORAM eviction algorithm such that

buckets are guaranteed to be empty regularly

1. **Evict over reverse-lexicographic order of paths**
2. **Also evict to sibling buckets**
3. **Set Z, A s.t. Pr[bucket overflow] = negl(security parameter)**
4. Evict to 1 bucket triplet at a time

Example:
evict to leaf 6

Informal guarantee:
Max layers = $O(\log N)$



Which A, Z work?

Theorem: $Z \geq A$, $N \leq A * 2^{L-1}$

→ Pr[bucket overflow] = $e^{-(2Z-A)^2/6A}$

- $Z = A = \theta(\log N)\omega(1)$ → Pr[bucket overflow] = $N^{-\omega(1)}$

  Note: $N = \mathrm{poly}(\mathit{security\ parameter})$

- **Asymptotics w/o server computation**
  - Bandwidth = $O(\log^2 N)\omega(1)$ blocks ☹
  - Client storage = $O(\log N)\omega(1)$ blocks ☹
  - Server storage = $O(N)$ blocks ☺

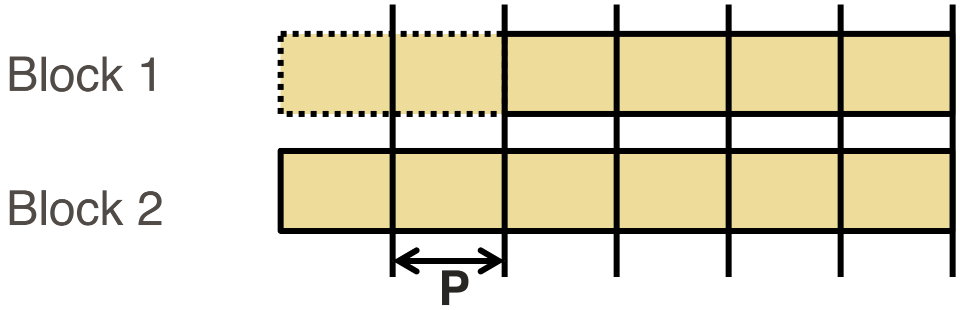
- **Not competitive w/o server computation**

Onion ORAM read w/ Additive-HE

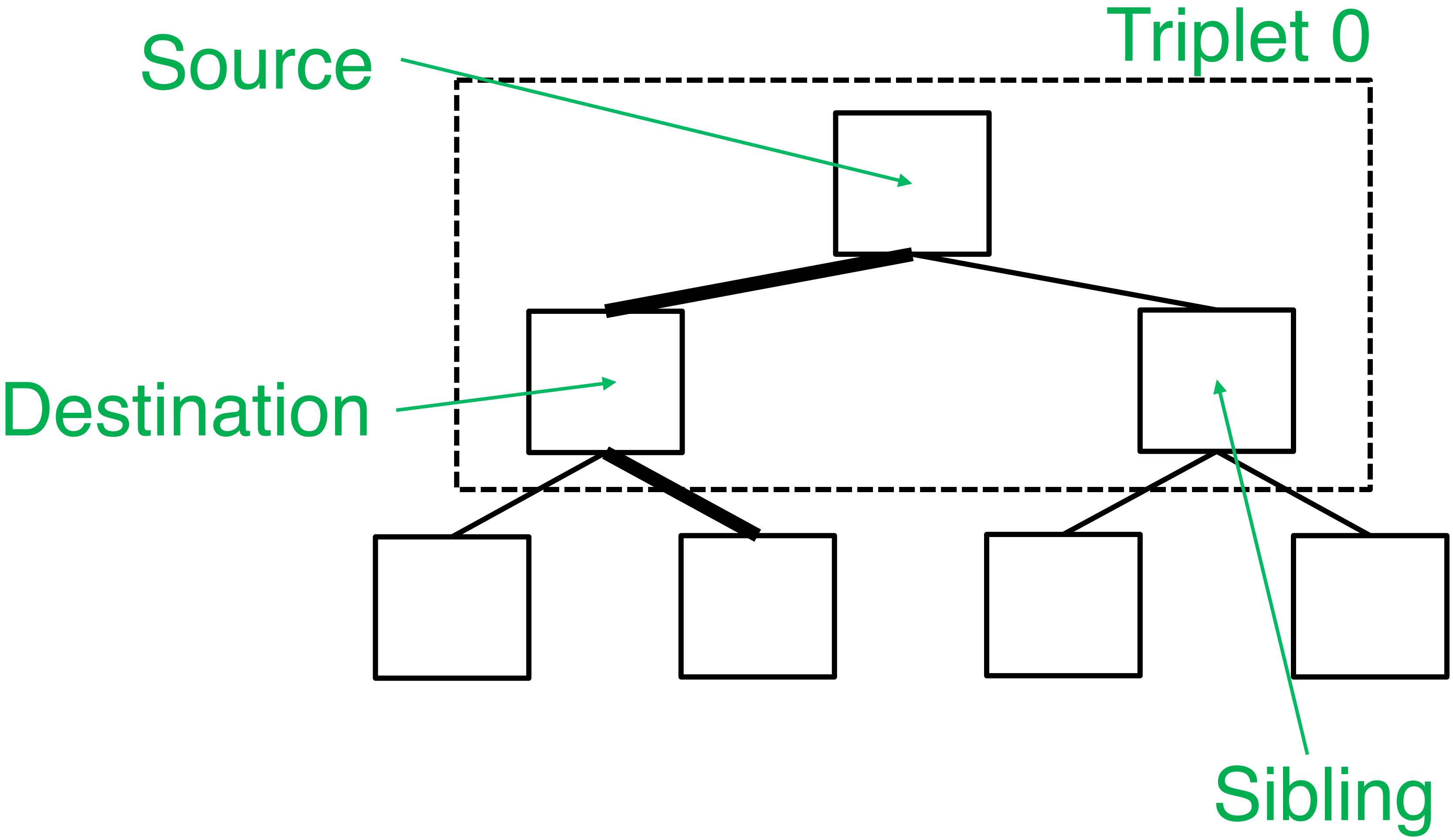
- **Same as previous proposal**
  - Client sends leaf
  - Server sends metadata
  - Client sends $\pi = E(0), E(0), \ldots E(1), \ldots E(0)$
  - Server sends block

Assume layers → ciphertext blowup

- **Simple scheme factoring in layers**
  - Elements of π have 1 layer
  - Pad blocks on path to S = Max($|Block_i|$: Blocks) bits
  - Split each padded block into C chunks s.t. S / C = Plaintext($\pi\downarrow i$) = P



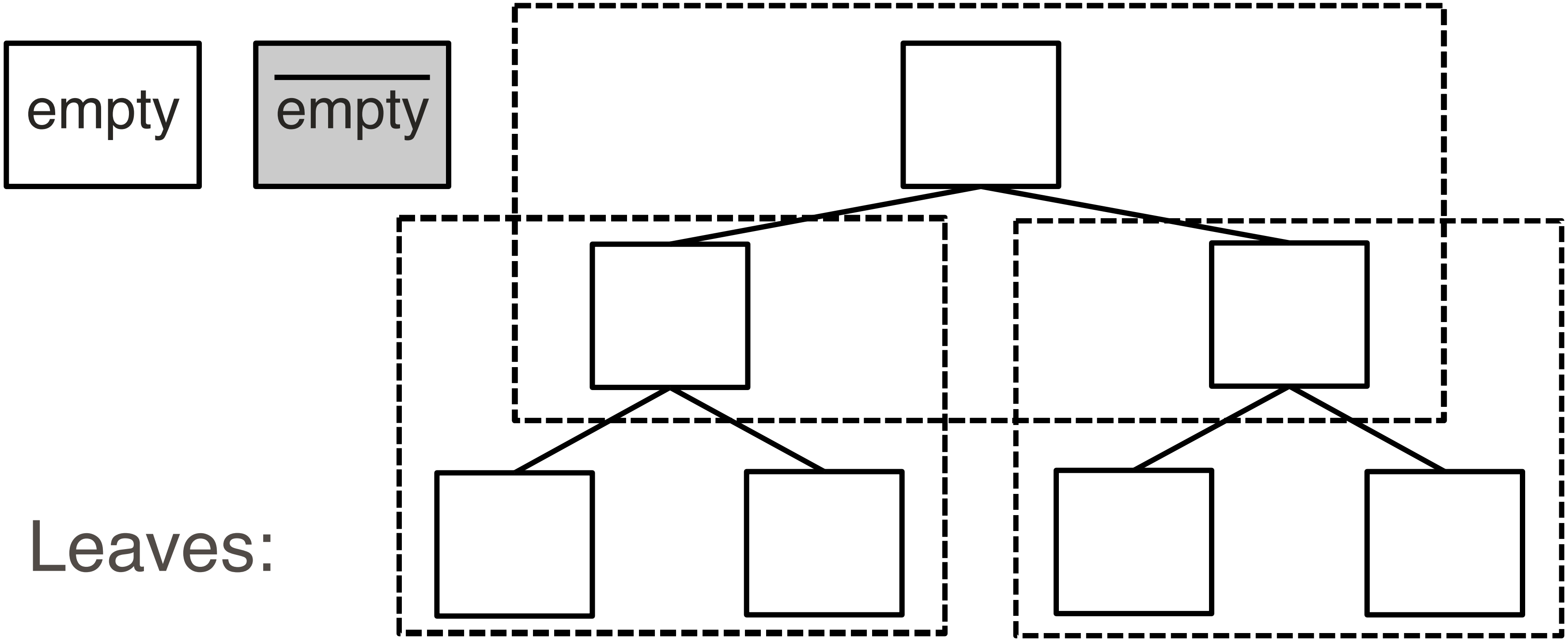
Eviction Terminology



Path(leaf)[*i*] =	i^{th} triplet on path
Path(leaf)[*i*].dest[*j*] =	j^{th} block in i^{th} triplet's dest. bucket

Layer Analysis

- **Useful properties:**
  1. At eviction start: non-leaf sibling buckets are empty
  2. At eviction end: non-leaf destination buckets are empty

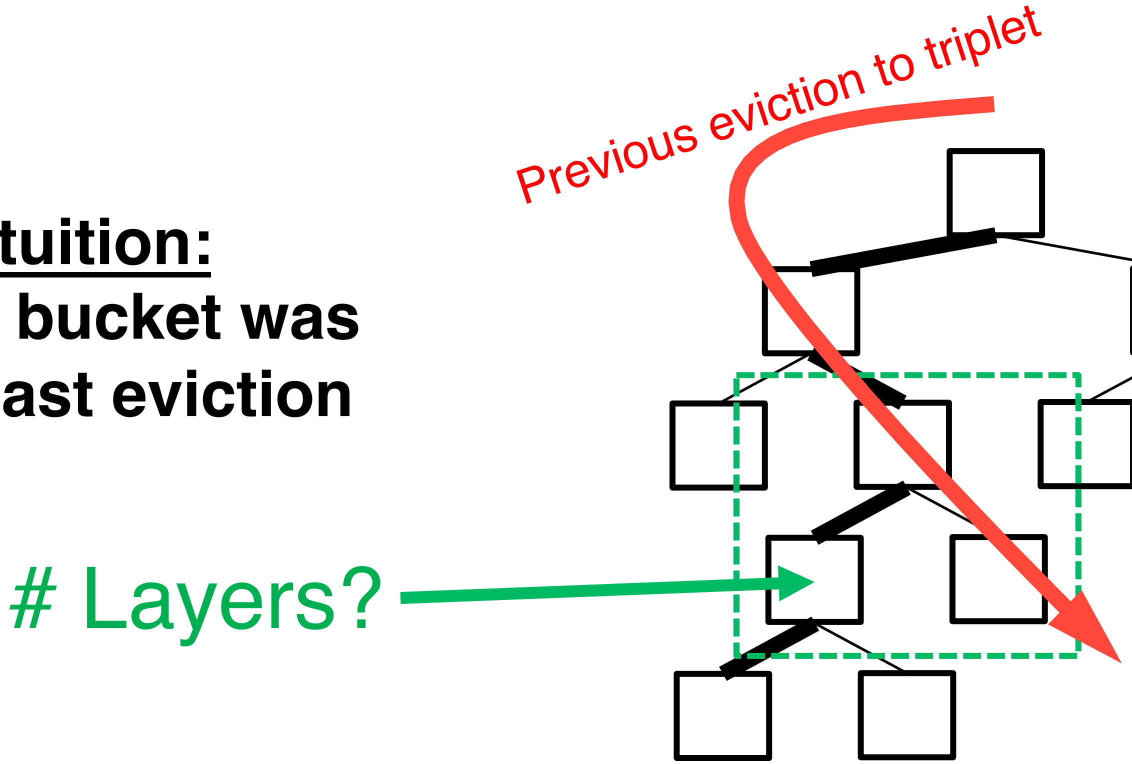


  - Blocks get stuck in the leaves
  - Non-leaves empty at regular intervals

Layer Analysis

- **Analyze: Layers on destination bucket at start of select**

Key intuition:
destination bucket was sibling on last eviction



Theorem: buckets at level $k<L$ have $\leq c*k+1$ layers

- **c is constant, $c=1$ in our final scheme**

Onion ORAM evict w/ Additive-HE

Client **Server**

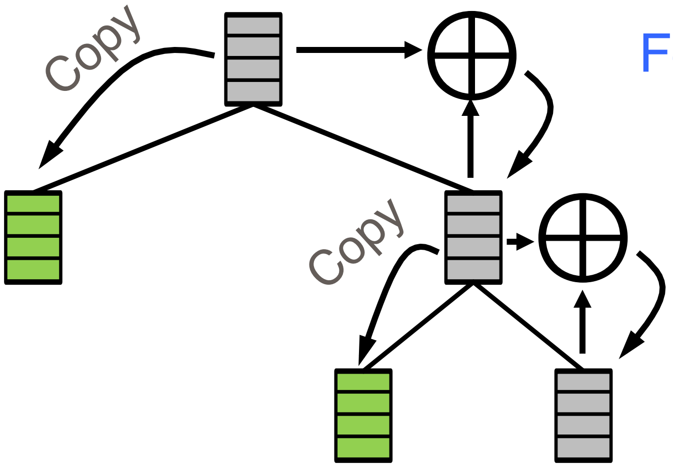
leaf_g (eviction path) known by server

E(metadata for Path(leaf_g))

Compute $\Pi=\{\pi_0 \ldots \pi_{Z*L}\}$

($|\pi_i|=O(Z)$ encrypted coefficients)

Π, E(updated metadata for Path(leaf_g))



For triplet *i*:

Path(leaf_g)[i].sibling = Path(leaf_g)[i].src

For slot *j*:

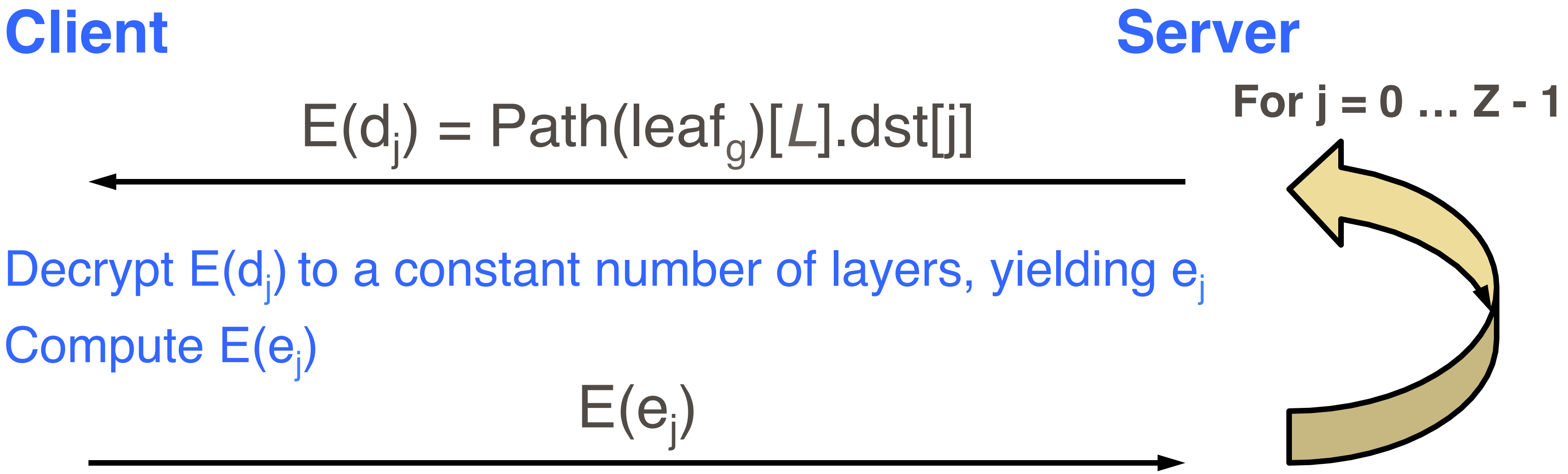
args = {Path(leaf_g)[*i*].dst[j] , Path(leaf_g)[*i*].src}

Path(leaf_g)[i].dst[*j*] = Select(π_{Z*i+j}, args)

Eviction Post-Processing

Problem: layers in leaves are not bounded

- **At end of each eviction…**



Client

Server

For j = 0 … Z - 1

$E(d_j) = \text{Path}(\text{leaf}_g)[L].\text{dst}[j]$

Decrypt $E(d_j)$ to a constant number of layers, yielding e_j

Compute $E(e_j)$

$E(e_j)$

- **Layer theorem now applies to all levels**
- **Adds constant amortized bandwidth if Z ~ A**

Setting parameters

Which Additive-HE scheme?

Problem: each layer can add ciphertext blowup

- Layer bound = $O(\log N)$
- Paillier (1999): $n \rightarrow n^2$ (n = RSA modulus) ☹
- Damgård-Jurik (2001): $n^s \rightarrow n^{s+1}$ ☺
  - s = free parameter
  - Strategy: set $s_0 = \log N$, $\log N$ layers → $n^{s_0 + \log N}$ = $n^{O(\log N)}$
- Operations are like Paillier:

$$E(a) \oplus E(b) = E(a)E(b) \qquad E(a) \otimes b = E(a)^b$$

- Best attack: factor n, complexity $\exp\left(|n|^{1/3} (\log |n|)^{2/3}\right)$

  $\therefore |n| = \theta(\log^3 N)$ → defeat attacks w/ complexity $N^{\omega(1)}$

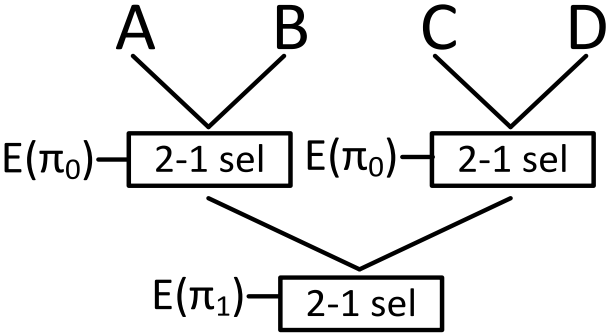
Optimization: Hierarchical PIR

- **So far … Select = $\bigoplus_{i} \pi{\downarrow}i \otimes Block{\downarrow}i$ "trivial linear PIR"**
  - Each select adds 1 layer

    $\text{layer bound}=\log N$
  - Z inputs → $|\pi|=Z*\text{layer bound}*|n|= log{\uparrow}5\ N\omega(1)$
- **Hierarchical PIR [Lipmaa 2005]**
  - Multiplexer tree
  - Z inputs → $|\pi|=\log Z$ coefficients

    → select adds $\log Z$ layers
  - $\therefore \text{layer bound}{\uparrow}' = \log M o g \log N$
  - Z inputs → $|\pi|=\log Z \ *\text{layer bound}{\uparrow}' * |n|$

    $=\log{\uparrow}4\ N\ \log{\uparrow}2\ \log N$



At least better in theory

Parameterization

- **Strategy: set $|\Pi| = |\{\pi_0 \ldots \pi_{Z*L}\}| = O(B)$**
- **I.e., Π contributes constant (amortized) bandwidth**

- **Let $Z = A = \log N \, \omega(1)$**
- $|\pi_{read}| = |n| * \log^2 N \omega(1)$ **(n = RSA modulus)**
- $|\pi_{evict}| = |n| * \log N \, \log^2 \log N$ **(mux tree)**
- $|\Pi_{evict}| = |n| * \log^2 N \, \log^2 \log N = \log^5 N \, \log^2 \log N$

- **Final asymptotics:**
  - Block size $B = \Omega(\log^5 N \, \log^2 \log N)$
  - Bandwidth = $O(B)$
  - Client storage = $O(B)$
  - Server storage = $O(BN)$

Ongoing/Future work

- **Decrease block size** $B = \Omega(k * \log^2 N \log^2 \log N)$
  - Modern schemes w/o computation: $B = O(\log^2 N)$
- **How?**
  - Server computation is $O(\log^2 N) \omega(1)$ blocks - is $O(\log N)$ possible?
  - Is there a suitable additive-HE scheme with $k = o(\log^3 N)$?

- **Protect against malicious servers**
  - Server performs select incorrectly
- **Improve Garbled RAM schemes?**
  - Use ORAM as a blackbox
- **Parameterization for SWHE for Onion ORAM**
  - No bootstrapping needed