Simple and Efficient Two-Server ORAM

Dov Gordon (George Mason U.)
Jonathan Katz (U. of Maryland)
Xiao Wang (MIT & Boston U.)

What is ORAM

- Outsourced memory storage, allowing oblivious memory access (read and write).
  - Any 2 sequences of operations are indistinguishable from the server(s) perspectives.
- Parameters of interest (per memory access):
  - communication complexity
  - rounds of interaction
  - storage requirements (server and client)
  - computational requirements.

Results

- Parameters of interest (per memory access):
  - communication complexity O(B log N)
  - rounds of interaction
  - storage requirements
  - computational requirements.

O(B log N) total, worst-case communication.

- Small constant: 10-20, depending on parameters. Compare to 160 in [LO].
- [A+] achieve O(log N / log log N) when B = $\boldsymbol{\Omega}(\lambda \log^2 N)$

[LO] Lu-Ostrovsky. Distributed oblivious RAM for secure two-party computation.
[A+] Abraham et al. Asymptotically tight bounds for composing ORAM with PIR.

Results

- Parameters of interest (per memory access):
  - communication complexity O(B log N)
  - rounds of interaction 2 rounds
  - storage requirements
  - computational requirements.

2 rounds!

- Can reduce to 1 if we moderately increase the client storage.
- Big open question in single server setting, while maintaining O(B log N) worst-case communication.

Results

- Parameters of interest (per memory access):
  - communication complexity O(B log N)
  - rounds of interaction 2 rounds
  - storage requirements 4N
  - computational requirements.

Servers store 4N encrypted blocks.
[LO] estimate server storage of $O(N\log^9 N)$ blocks.

- Most single server ORAMs require the server to store $O(N\log^2 N)$ blocks.

[LO] Lu-Ostrovsky. Distributed oblivious RAM for secure two-party computation.

Results

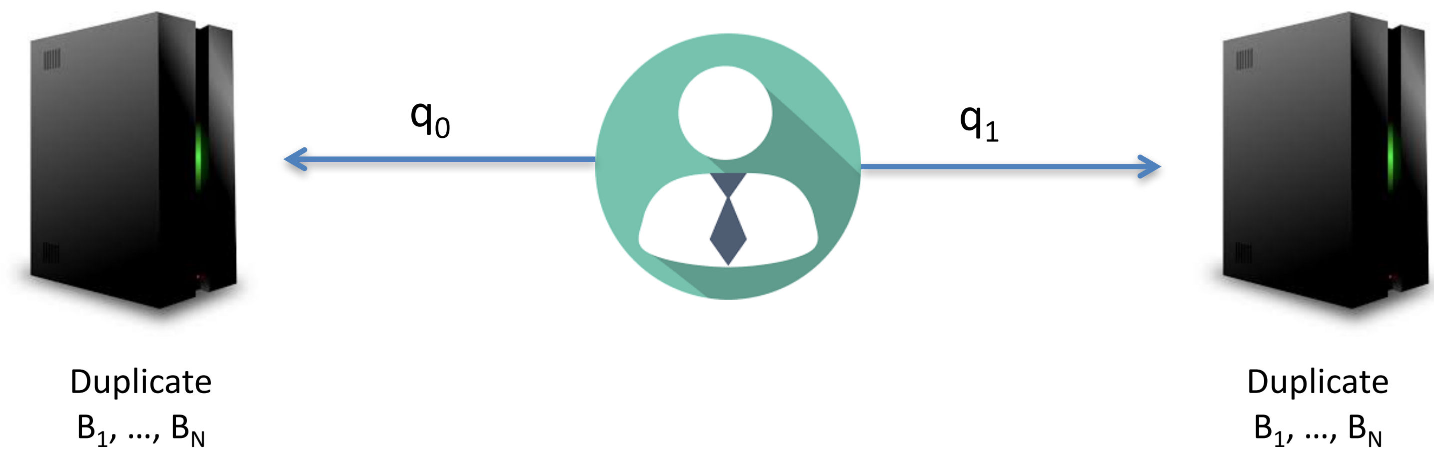
- Parameters of interest (per memory access):
  - communication complexity — O(B log N)
  - rounds of interaction — 2 rounds
  - storage requirements — 4N
  - computational requirements. — O(N)

Our servers have to do O(N) symmetric key operations for each query, which is a drawback to our construction.

- Computation time is not likely to be the bottleneck on reasonable data sizes.
- [DS] Demonstrate this empirically in another O(N) protocol.

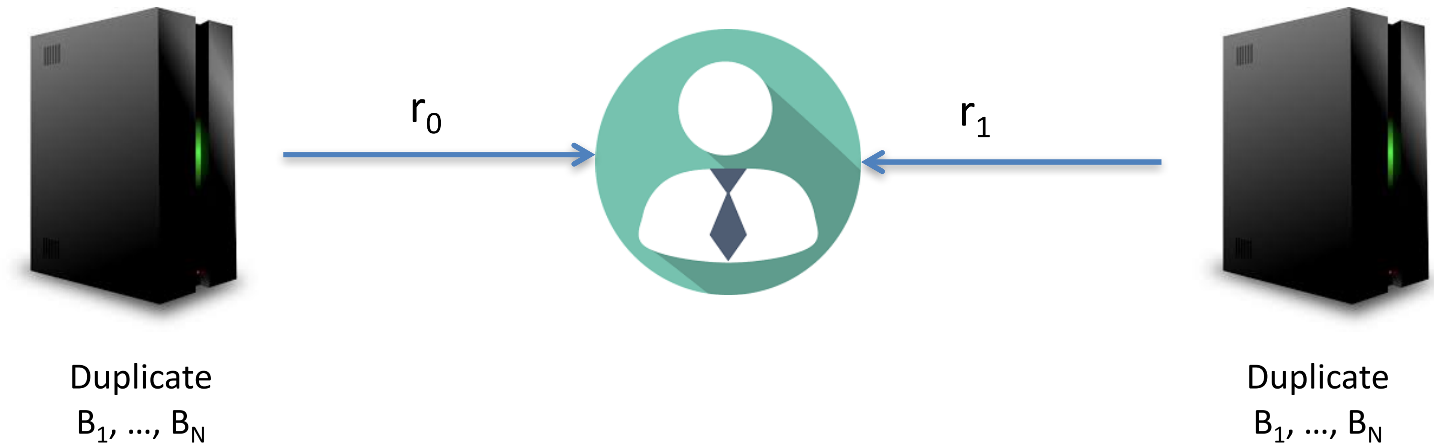
[DS] Doerner and Shelat. Scaling ORAM for secure computation, 2017.

Private Information Retreival



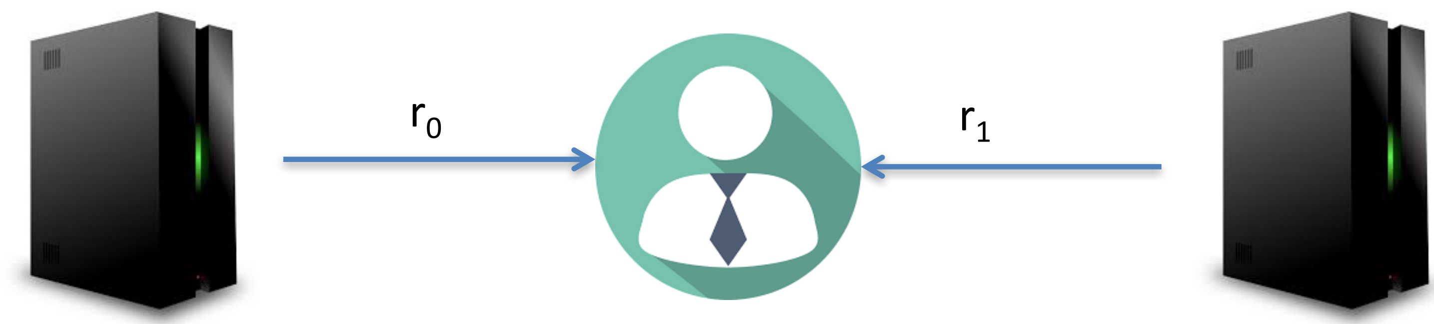
1. Client wants to read data block B_i in data array $B_1, \ldots, B_N$
   $(q_0, q_1) \leftarrow$ PIR.C $(1^\kappa, B, |D|, i)$

Private Information Retreival



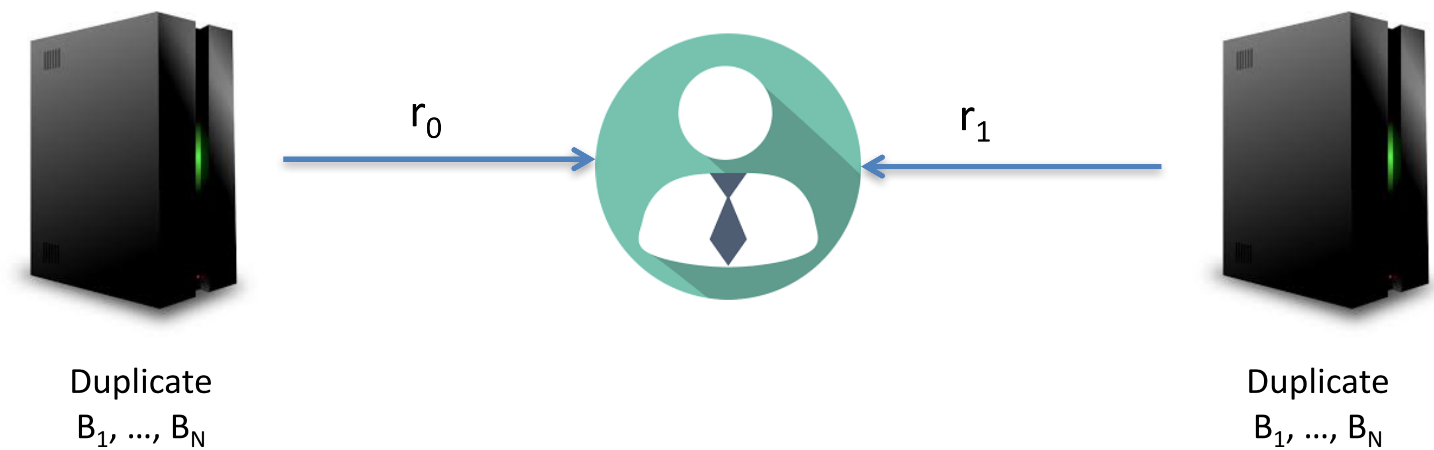
1. Client wants to read data block B_i in data array $B_1, \ldots, B_N$
   $(q_0, q_1) \leftarrow \mathrm{PIR.C}\ (1^{\kappa}, B, |D|, i)$
2. Servers reply with r_0 and r_1, such that $r_0 \oplus r_1 = B_i$
   $(r_b) \leftarrow \mathrm{PIR.S}(B_1, \ldots, B_N, q_b)$

Private Information Retreival



1. Client wants to read data block B_i in data array $B_1, \ldots, B_N$
   $(q_0, q_1) \leftarrow \text{PIR.C}\ (1^\kappa, B, |D|, i)$
2. Servers reply with r_0 and r_1, such that $r_0 \oplus r_1 = B_i$
   $(r_b) \leftarrow \text{PIR.S}(B_1, \ldots, B_N, q_b)$

Private Information Retreival



1. Client wants to read data block B_i in data array $B_1, \ldots, B_N$
   $(q_0, q_1) \leftarrow \text{PIR.C}(1^\kappa, B, |D|, i)$
2. Servers reply with r_0 and r_1, such that $r_0 \oplus r_1 = B_i$
   $(r_b) \leftarrow \text{PIR.S}(B_1, \ldots, B_N, q_b)$

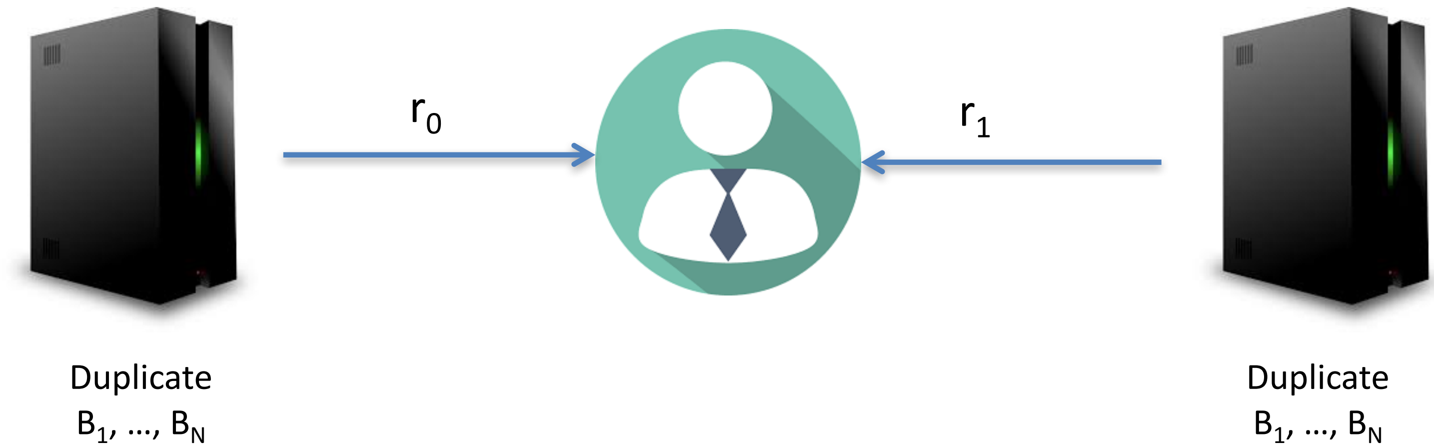
Security requirement: Servers learn nothing about i.

Structural Assumption ([BGI]): $r_b = \bigoplus_j \lambda_b[j] \cdot B_j$, where

$$\lambda_0[j] \oplus \lambda_1[j] = 1 \leftrightarrow j = i$$

[BGI] Boyle, Gilboa, Ishai. Function secret sharing: Improvements and extensions

Private Path Retreival



1. Client wants to read data **path** to leaf node **i** in data **tree T.**
   $(q_0, q_1) \leftarrow \text{PIR.C}(1^\kappa, B, |T|, i)$
2. Servers reply with $r_0[1] \ldots r_0[L]$ and $r_1[1] \ldots r_1[L]$, one random value for each layer, such that $r_0[j] \oplus r_1[j] = T_j \leftrightarrow j$ lies on the path to leaf i.
   $(r_b[1] \ldots r_b[L]) \leftarrow \text{PPR.S}(T, q_b)$

Security requirement: Servers learn nothing about i.

Private Path Retreival
(Naïve Solution)

Each layer of the tree can be treated as its own, independent instance of a PIR scheme.

To query the path to leaf node B_i, the client makes L independent PIR queries, one for each layer of the tree.

The cost of [BGI] for a single query:	$(2\|B\|) + O(\kappa \log n)$
The cost is (log n) (PIR),	$(2\|B\| \log n) + O(\kappa \log^2 n)$
We will show how to achieve	$(2\|B\| \log n) + O(\kappa \log n)$

[BGI] Boyle, Gilboa, Ishai. Function secret sharing: Improvements and extensions

Private Path Retreival

To query the path to leaf node B_i, the client makes a **single** PIR query for index i, over leaf nodes $B_1 \ldots B_n$.

Private Path Retreival

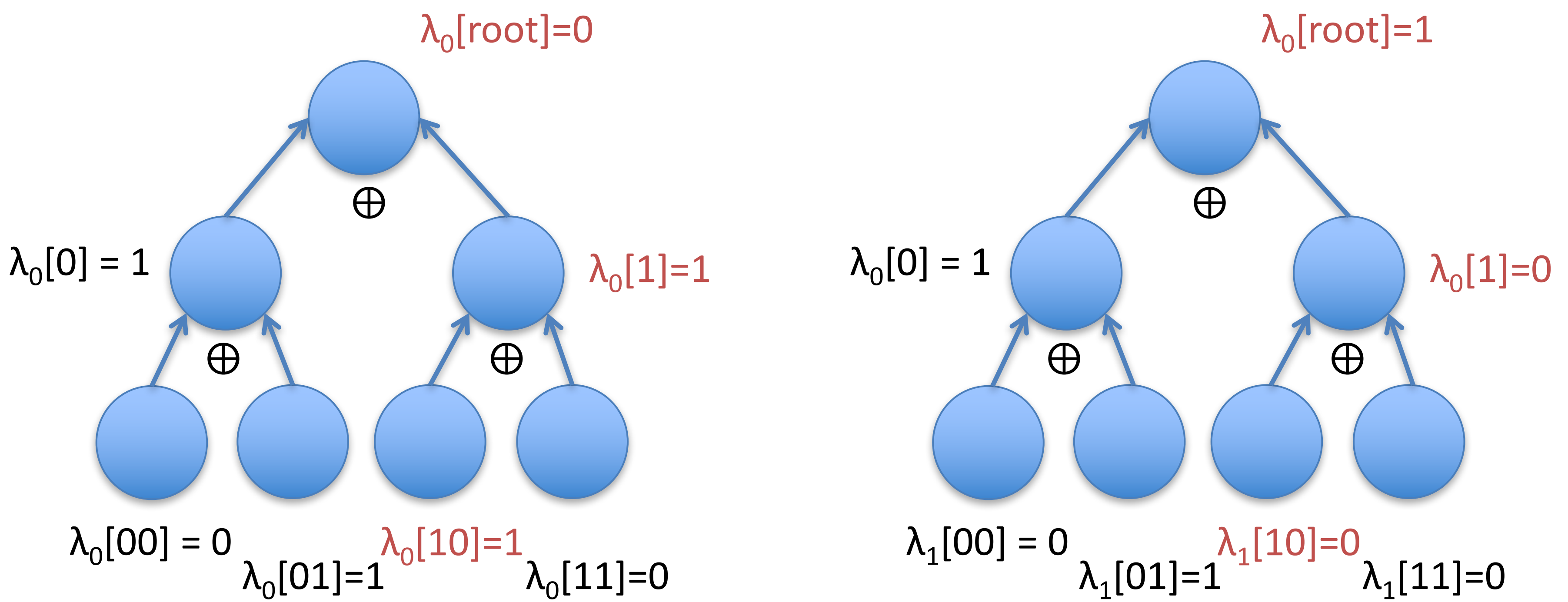
To query the path to leaf node B_i, the client makes a **single** PIR query for index i, over leaf nodes $B_1 \ldots B_n$.

In PIR scheme, server responses are: $r_b = \bigoplus_j \lambda_b[j] \cdot B_j$

Private Path Retreival

To query the path to leaf node B_i, the client makes a **single** PIR query for index i, over leaf nodes $B_1 \dots B_n$.
In the PPR scheme, server sends $r_b[j]$ for layer j, where

$$r_b[j] = \bigoplus_k \lambda_b[k] \cdot T[k],\ \forall k: |k| = j.$$



ORAM

Oblivious RAM
(structure)

- Data is stored in a tree of depth $L = \log N$.
- Each node in the tree contains a "bucket" of size Z.
- Root node is special: stash stored at client.
- Records are of the form $\mathrm{Enc}(\mathrm{flag}, i, F_k(i), B_i)$
  - Flag indicates real or dummy.
  - $F_k(\cdot)$ is a PRF held by the client.

Oblivious RAM

(structure)

- Data is stored in a tree of depth L = log N.
- Each node in the tree contains a "bucket" of size Z.
- Root node is special: stash stored at client.
- Records are of the form $Enc(flag,i,F_k(i),B_i)$
  - Flag indicates real or dummy.
  - $F_k(\cdot)$ is a PRF held by the client.

Invariants:

1. B_i is always along the path to leaf node $F_k(i)$.
2. The most up-to-date copy of B_i is closest to the root.

Oblivious RAM

(read/write)

To read B_i:

- PPR.C(F_k(i)), and return the real record closest to the root.
  - We do **NOT** assign B_i a new leaf node!

To write B_i:

- Remove records of form (1, i, F_k(i), *) from stash.
- Write record (1, i, F_k(i), B_i) to stash.

Invariants:

1. B_i is always along the path to leaf node F_k(i).
2. The most up-to-date copy of B_i is closest to the root.

Oblivious RAM

(static leaf assignments)

We do **NOT** assign B_i a new leaf node!

In most prior ORAM constructions, the path to B_i is requested in the clear:

- To ensure security, every time B_i is accessed, it must lie on a new, random path.
- Requires a **dynamic** mapping between records and their leaf nodes. Typically stored recursively, requiring log N overhead.
- In contrast, we can use a **static**, pseudo-random mapping.

Invariants:

1. B_i is always along the path to leaf node $F_k(i)$.
2. The most up-to-date copy of B_i is closest to the root.

Oblivious RAM

(eviction)

As in prior work, our root node (stash) fills up.

Every *A* operations, we choose a path, and push items down the path as far as they can go.

- Path is chosen deterministically, as in [G+], using reverse lexicographic ordering.
- Both invariants are maintained:
  - We remove duplicate real records when not the closest to root.
  - We push remaining records down, subject to the first invariant.

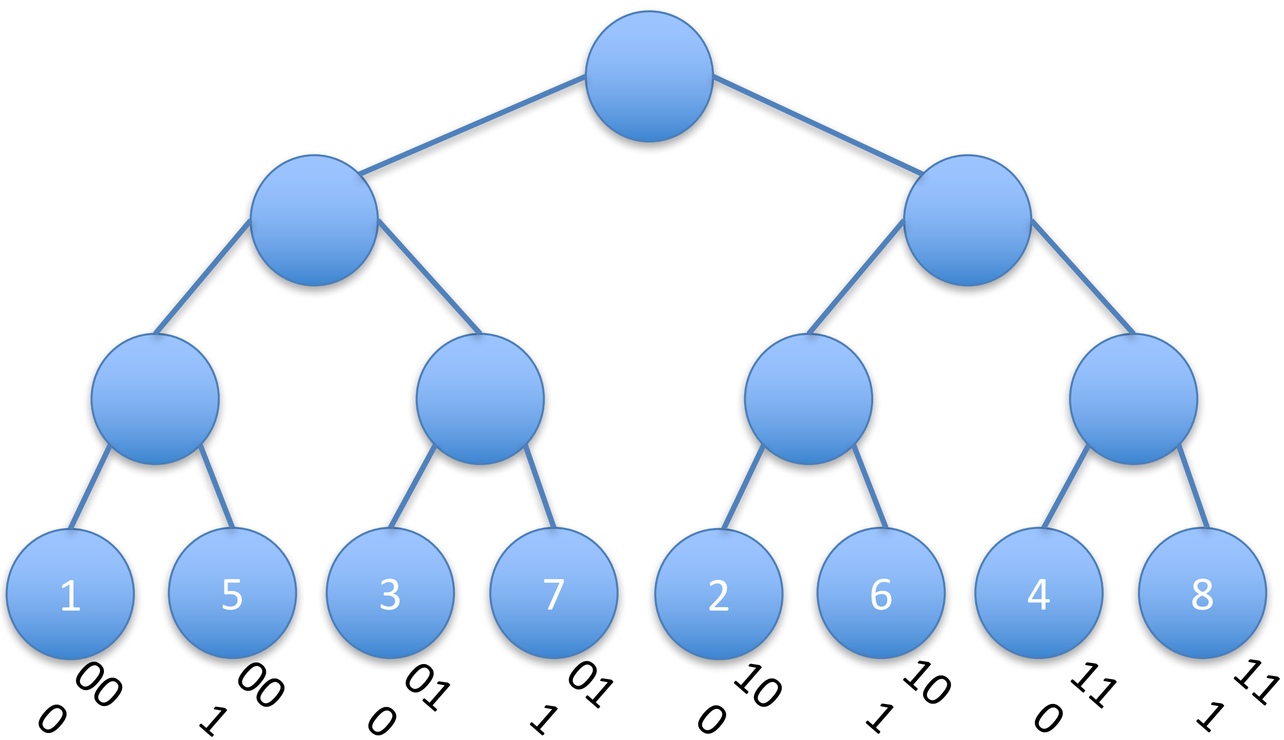
[G+] Gentry et al. Optimizing ORAM and using it efficiently for secure computation.

Invariants:

1. B_i is always along the path to leaf node $F_k(i)$.
2. The most up-to-date copy of B_i is closest to the root.

Oblivious RAM
(eviction)

- Path is chosen deterministically, as in [G+], using reverse lex. ordering.
- Easy analysis: each node at level j is on an eviction path exactly every 2^j evictions. Expected load on each node is ½.



[G+] Gentry et al. Optimizing ORAM and using it efficiently for secure computation.

Security / Stash

- The security follows immediately from the security of the PPR scheme.
- We can bound the stash size as done in [R+]. Communication is minimized when 3Z/A is minimized.

	$Z = 3$	$Z = 4$	$Z = 5$	$Z = 6$	$Z = 7$
$A = 1$	16	14	13	12	11
$A = 2$	-	21	18	16	15
$A = 3$	-	32	24	21	19
$A = 4$	-	-	33	26	23
$A = 5$	-	-	-	34	28

Table 1: **Bounds on the number of blocks in the client's stash.** These bounds hold except with probability 2^{-40} (per operation).

[R+] Ren et al. Constants Count: Practical Improvements to Oblivious RAM, 2014.

Further Features

- Public read/write is cheaper than oblivious operations. Simply request the whole path in the clear.
- Initialization is for free: we can share $F_k()$ with the servers, as long as the accesses don't depend on F_k.
- Only write operations fill the root node, so eviction can be less frequent if you can reveal read vs. write.

THANKS!