Integrity Verification for Path Oblivious-RAM (in Ascend)

Ling Ren, Christopher Fletcher, Xiangyao Yu, Marten van Dijk, Srinivas Devadas

Massachusetts Institute of Technology

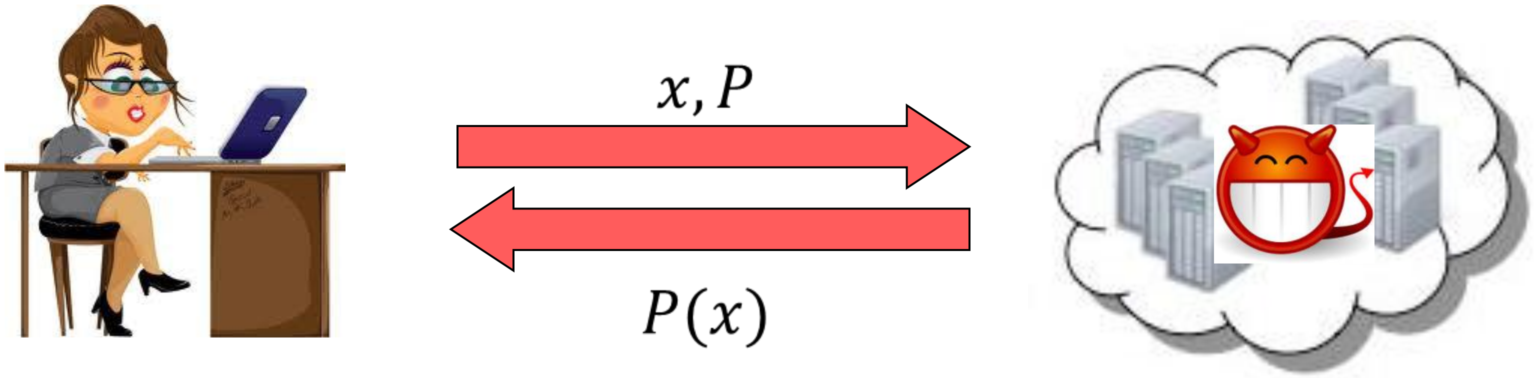
HPEC'13

Outline

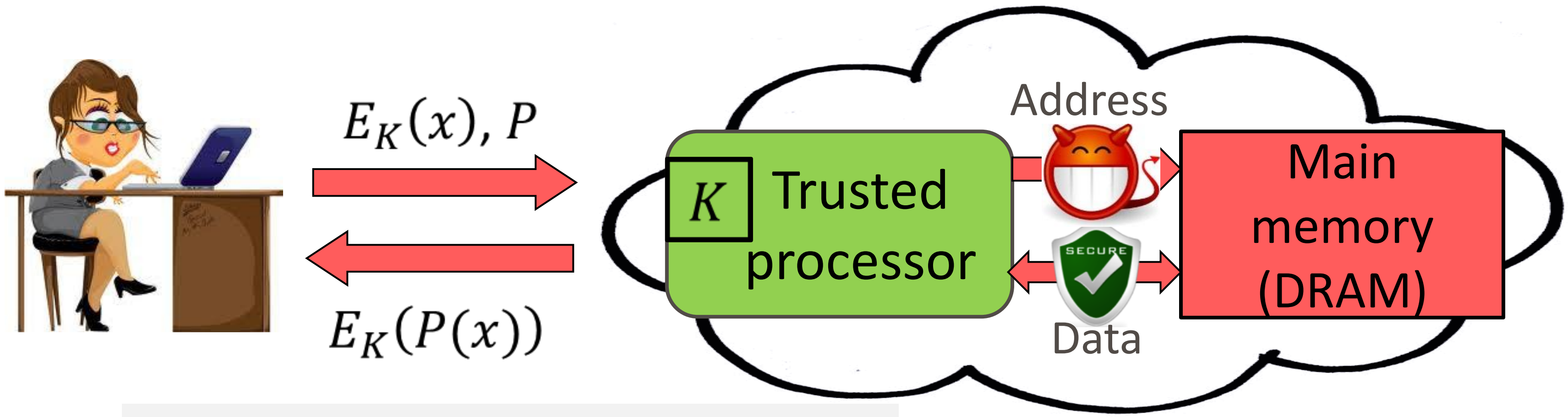
- **Background**
  - Ascend secure processor
  - Path ORAM
- **Motivation**
- **Integrity verification for Path ORAM**

Privacy & Integrity in Cloud

- **Context**: cloud computing
- **Privacy**: user's data not leaked to anyone
- **Integrity:** computation is done correctly (user gets P(x))



Secure Processors



User data decrypted inside and computed in the clear

Data can be encrypted but address cannot

+ Integrity (e.g. Aegis) — **Integrity?**

– Leakage through address/timing/power — **Privacy?**

Leakage through Addresses

```
for i = 1 to N
    if (x == 0)
        sum += A[i]
    else
        sum += A[0]
```

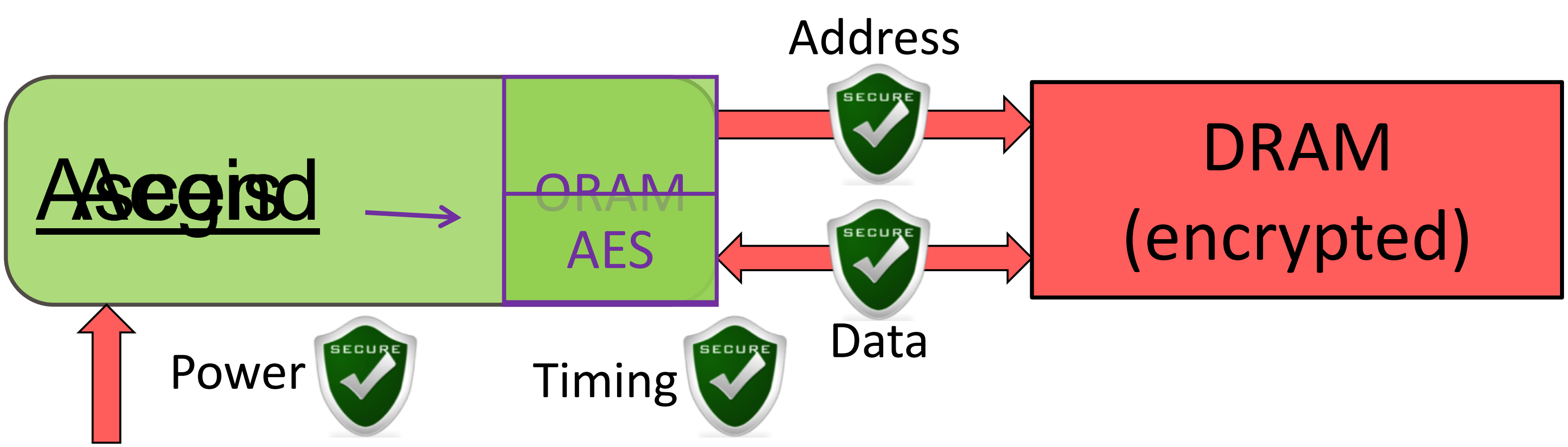
⇨ Address sequence: 0x00, 0x01, 0x02 …

⇨ Address sequence: 0x00, 0x00, 0x00 …

- **Previous work** [HIDE, NDSS12] **has shown access pattern leakage in practical applications**
- **Addresses can be monitored by software**

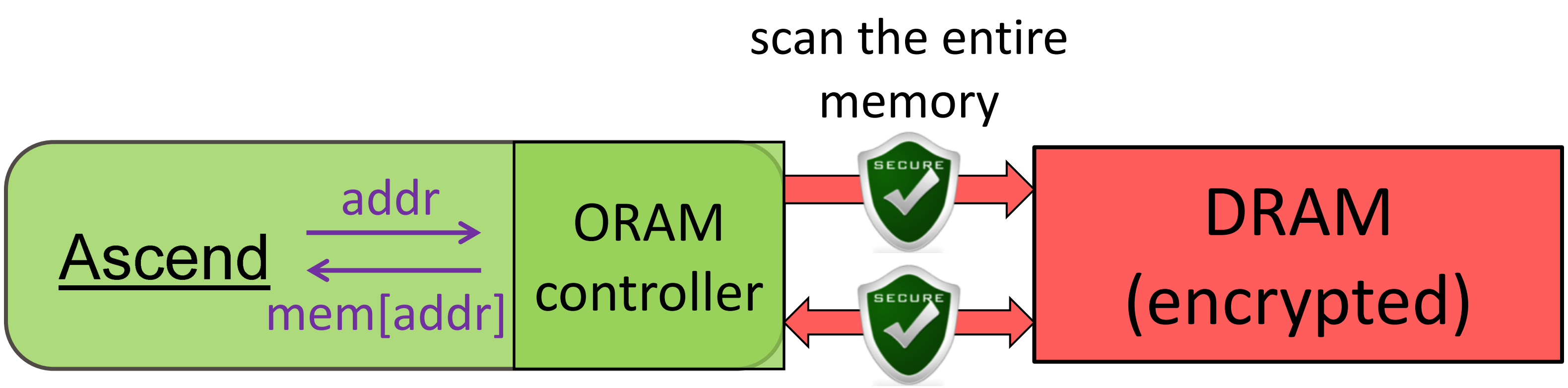
Ascend secure processor

- **Existing secure processors (e.g., XOM, Aegis)**
  - \+ Can provide integrity
  - ~~– Leakage through address/timing/power, or trust the program~~
- **Ascend: terminate leakage over above channels**
  - – I/O channel: Oblivious RAM
  - – Timing and power channel …



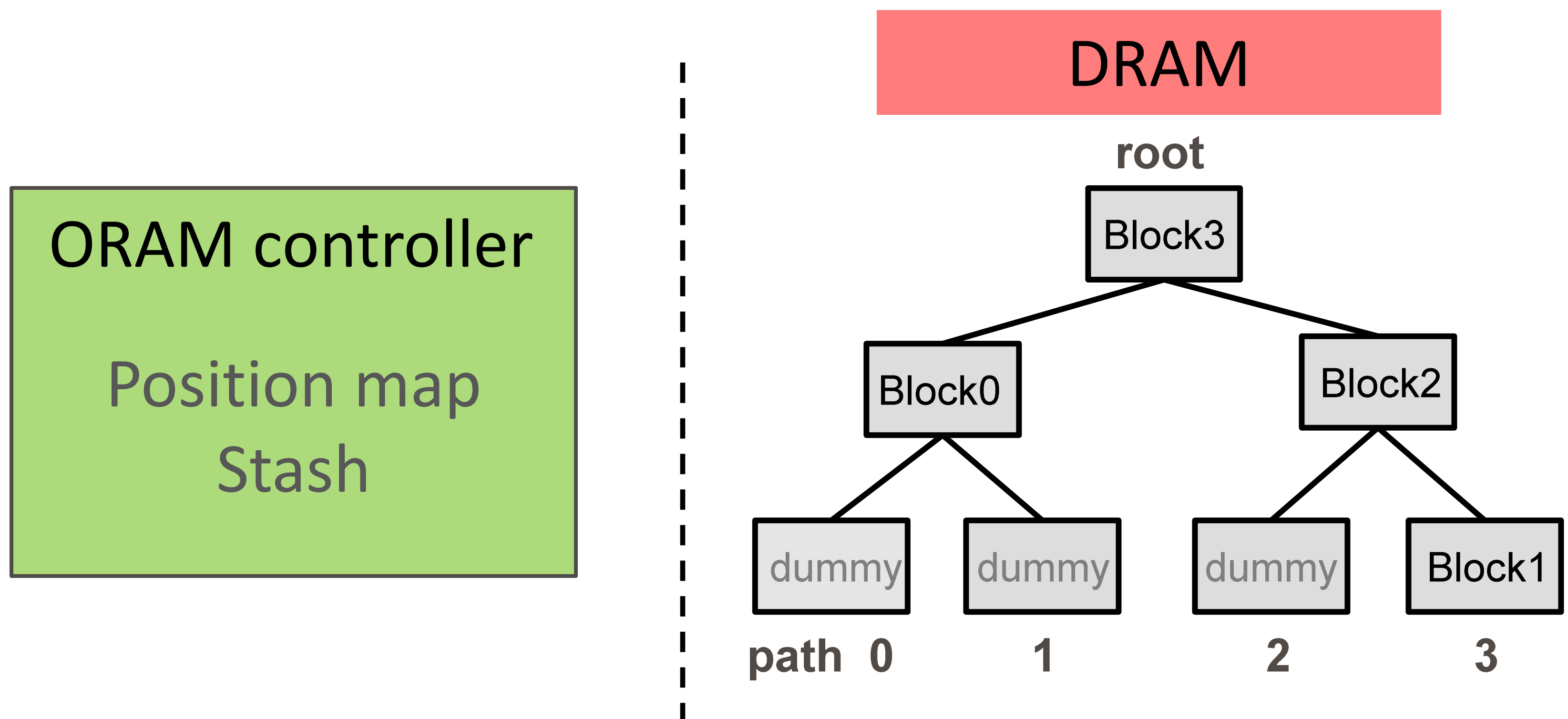
Oblivious RAM (ORAM)

- **Hide access pattern**
  - Read vs. write
  - Make all address sequences indistinguishable
- **Naïve ORAM**
  - Read/write the entire memory on each access
  - Probabilistic encryption → everything always changes
  - $O(N)$ overhead, N = # of data blocks (cache lines) in the memory



Path ORAM

- **Path ORAM**
  - One of the most efficient ORAMs, simple
- **External DRAM structured as a binary tree**
  - Each node contains Z blocks (Z=1 in the example below)



Path ORAM

- **Position Map: map each block to a random leaf**
- **Invariant: if a block is mapped to a path, it must be on that path or in the stash**
  - Stash: temporarily hold some blocks

ORAM controller

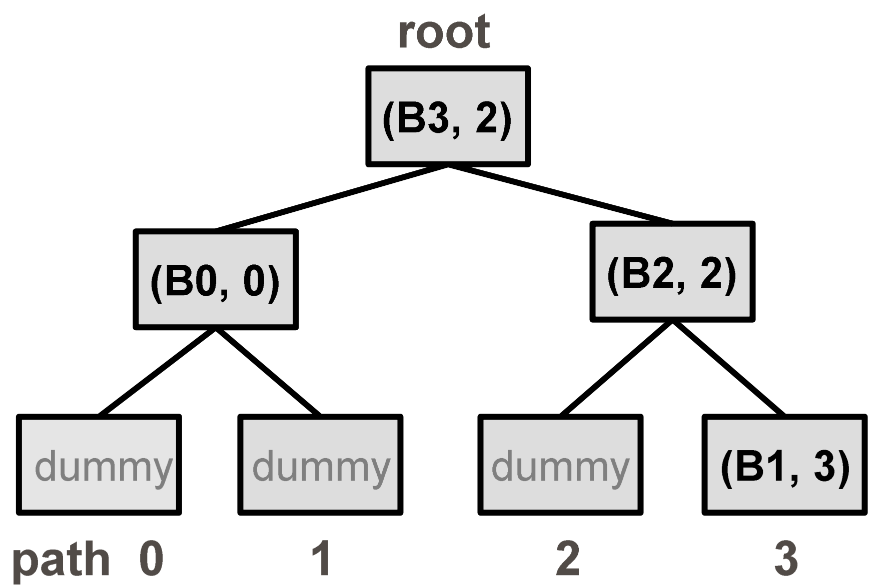
Stash

			(B4, 1)

Position Map

Block	Path
B0	0
B1	3
B2	2
B3	2
B4	1

DRAM



Path ORAM Operation

- **Access Block 1** PosMap(B1) = 3
  - Read all blocks on path 3
  - Remap B1 to a new random path
  - Write as many blocks as possible back to path 3

$$O(L) = O(\log(N))$$

ORAM controller

Stash

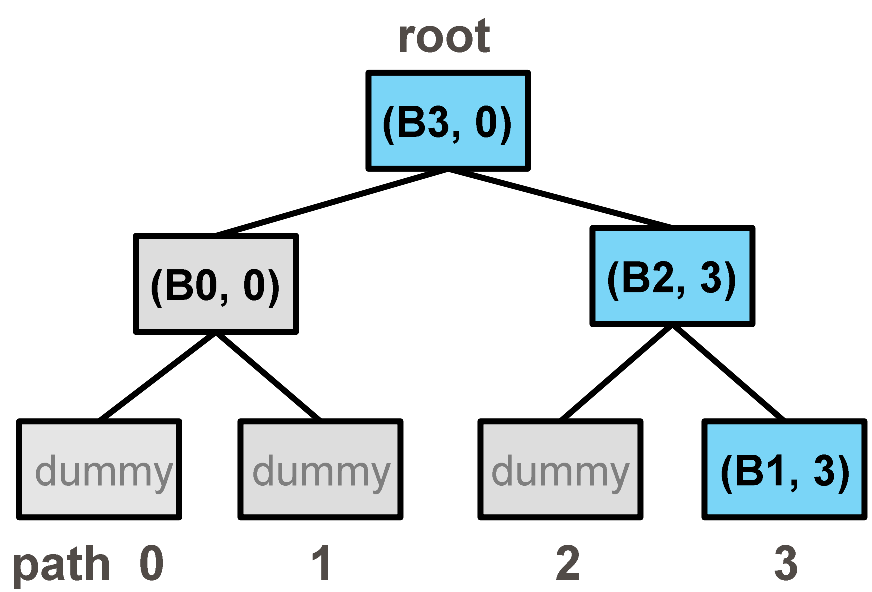
dummy

		(B1, 1)	(B4, 1)

Position Map

Block	Path
B0	0
B1	3
B2	3
B3	0
B4	1

DRAM

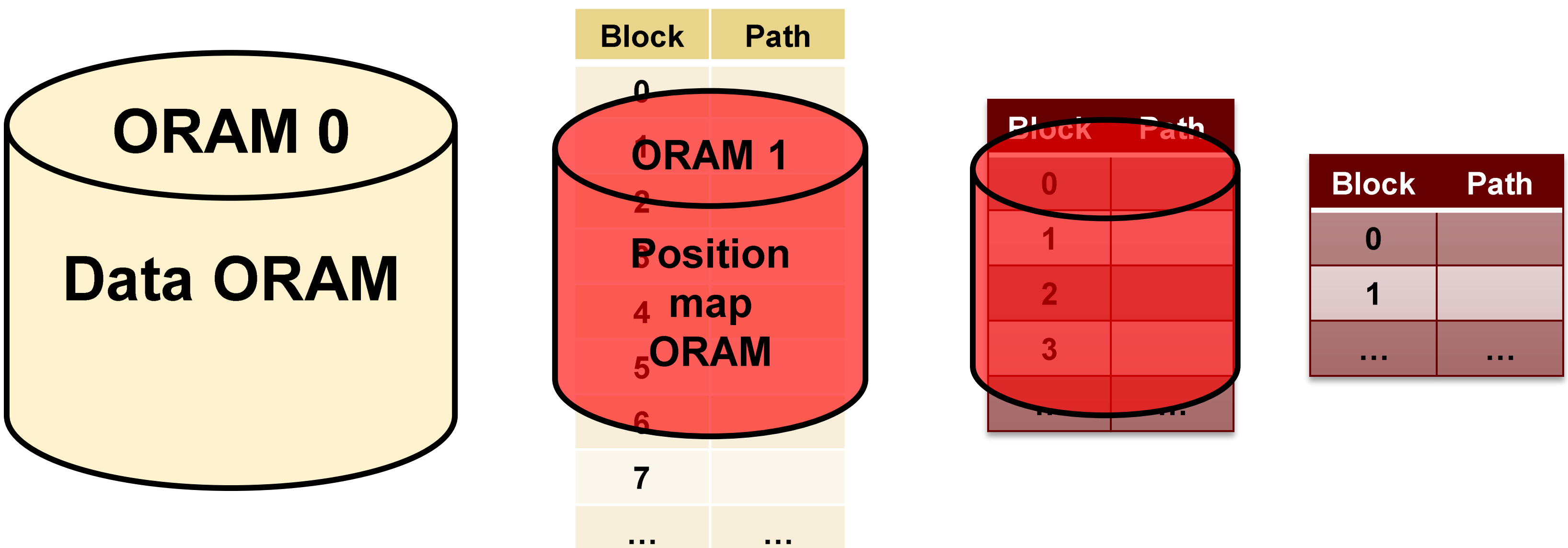


Path ORAM Security

- **A random path is read/written on every access**
  - Extracted from PosMap, which is always random and fresh due to remapping
- **All ciphertexts on the path always change**
  - Due to probabilistic encryption

Recursive Path ORAM

- **Problem: Position map too large**
- **Solution: Recursion**
  - Trade off latency for smaller position map
- **Ascend has 3~4 ORAMs in the recursion**

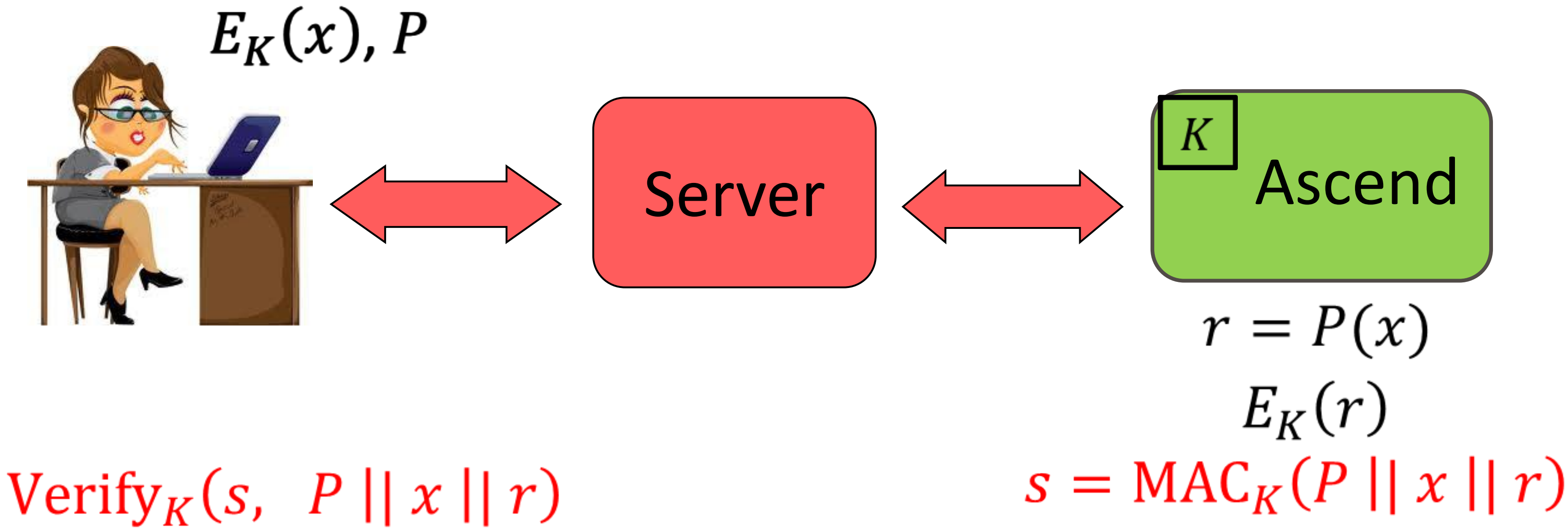


4 GB **93 MB** **2 MB** **43 KB**

Outline

- **Background**
  - Ascend secure processor
  - Path ORAM
- **Motivation for Path ORAM integrity**
- **Integrity verification for Path ORAM**

Motivation: Ascend Integrity



- **Certified execution protocol: message authentication code (MAC) for** P, x, r
- **Verify the integrity (freshness, authenticity) of external memory**
  - Aegis verifies DRAM. Ascend has to verify Path ORAM

Another Motivation

- **Recursive Path ORAM's privacy is broken without integrity verification when attackers can modify ORAM**
  - Revert PosMap ORAMs to force reuse of old leaf labels

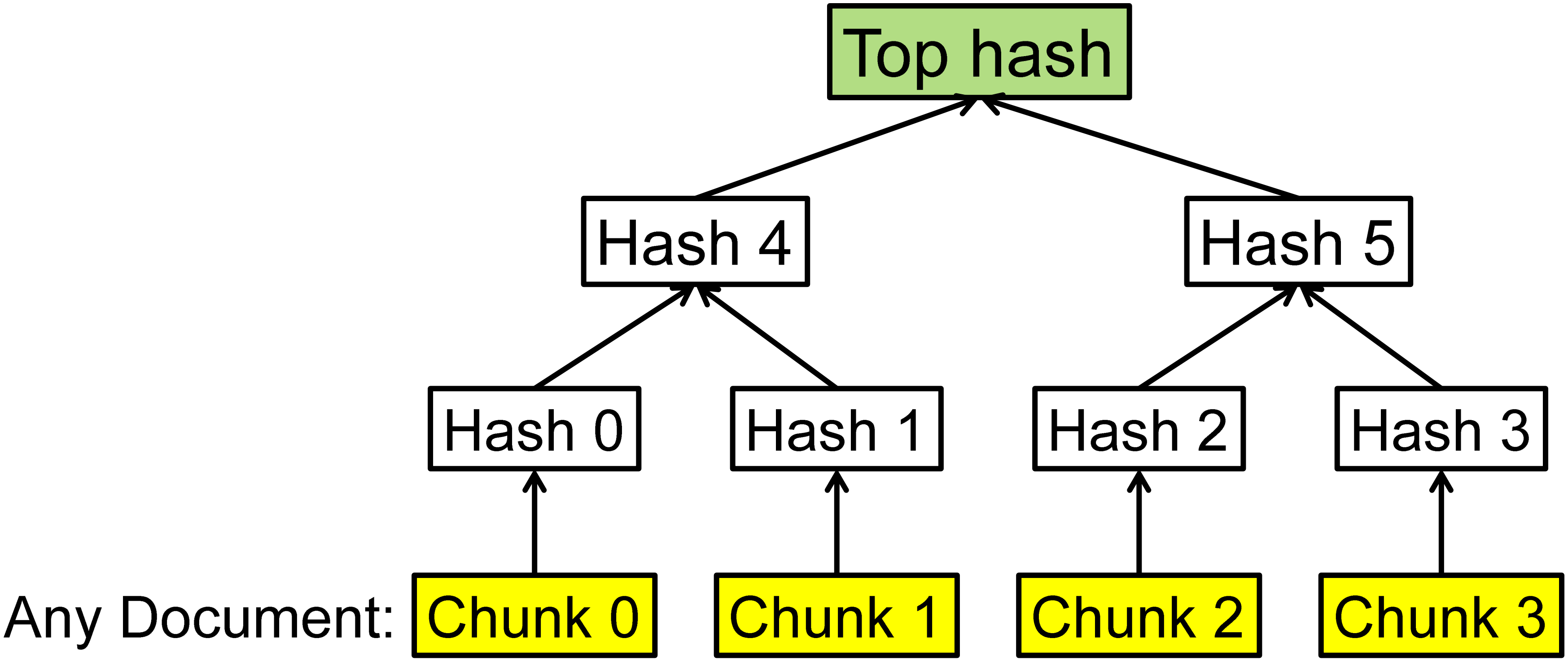
- **So we need to verify Path ORAM integrity**
  - To maintain privacy of recursive Path ORAM
  - To achieve integrity in Ascend

Outline

- Background
- Motivation for Path ORAM integrity
- **Integrity verification for Path ORAM**
  - Verify one Path ORAM
  - Verify recursive Path ORAM

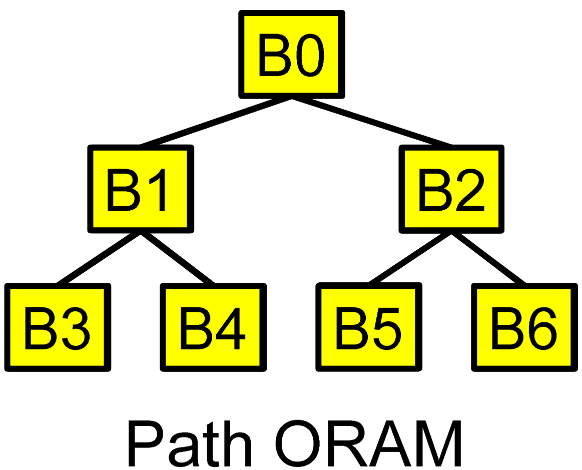
Background – Merkle Signature

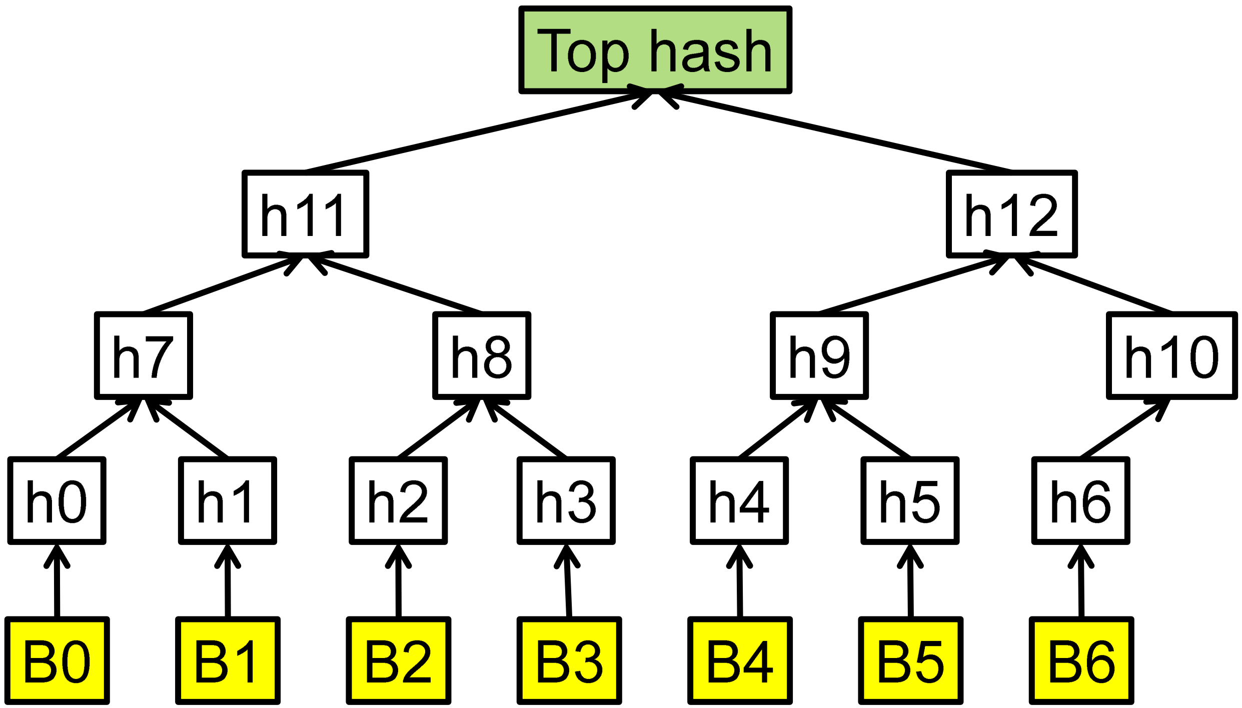
- General, can be used for any document, any ORAM
- Efficient $O(L) = O(\log(N))$
- Security reduced to collision-resistant hash function



Merkle Signature for Path ORAM?

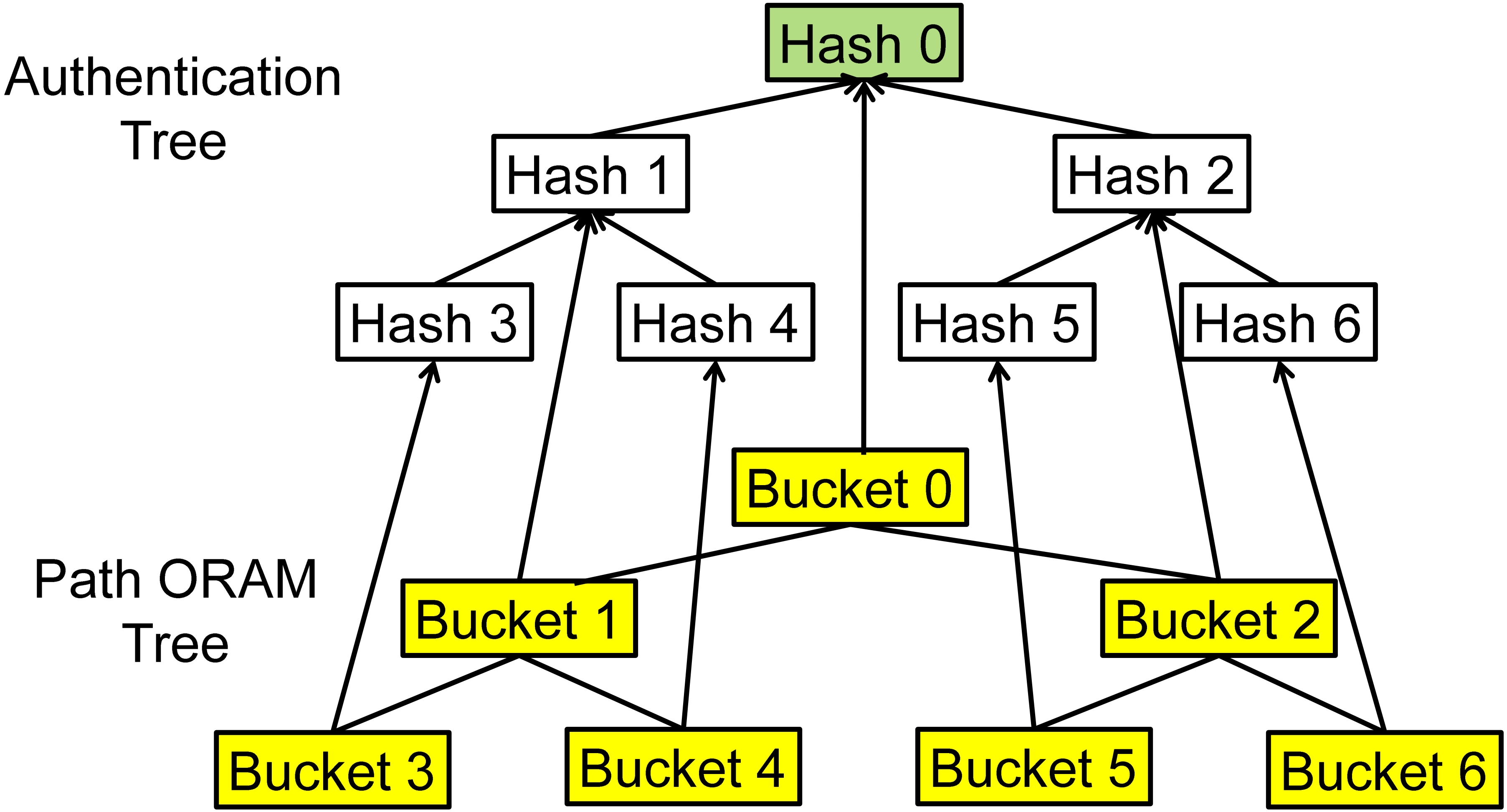
- **ORAM hides access pattern**
  - **→ (pretend to) verify all buckets on a path**
  - **→ $O(L^2)$ complexity**
    - – Path ORAM $O(L)$ complexity





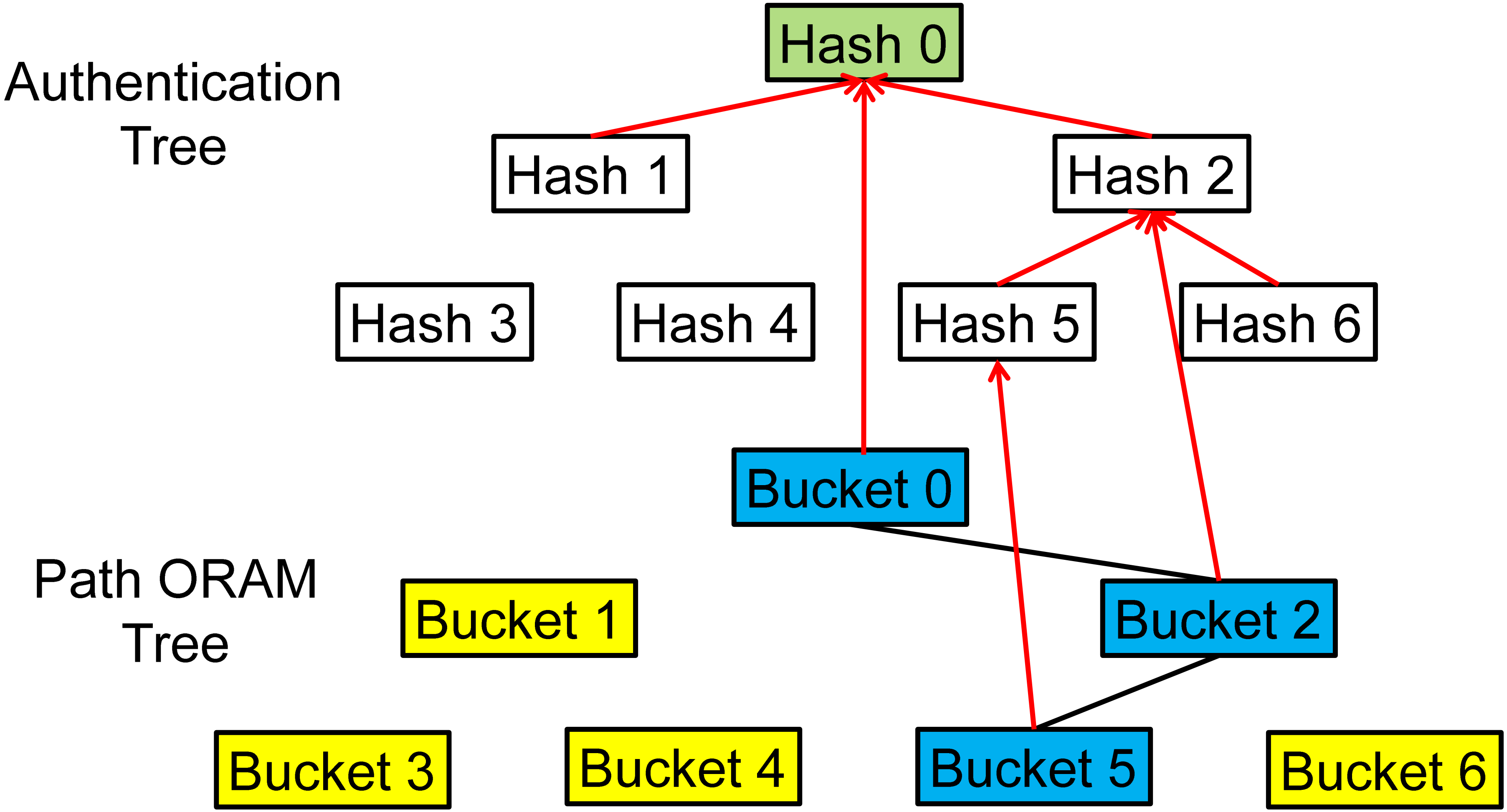
Verify one Path ORAM

- **Combine Merkle tree and Path ORAM tree**



Verify one Path ORAM

$O(L)$



Verify Recursive Path ORAMs

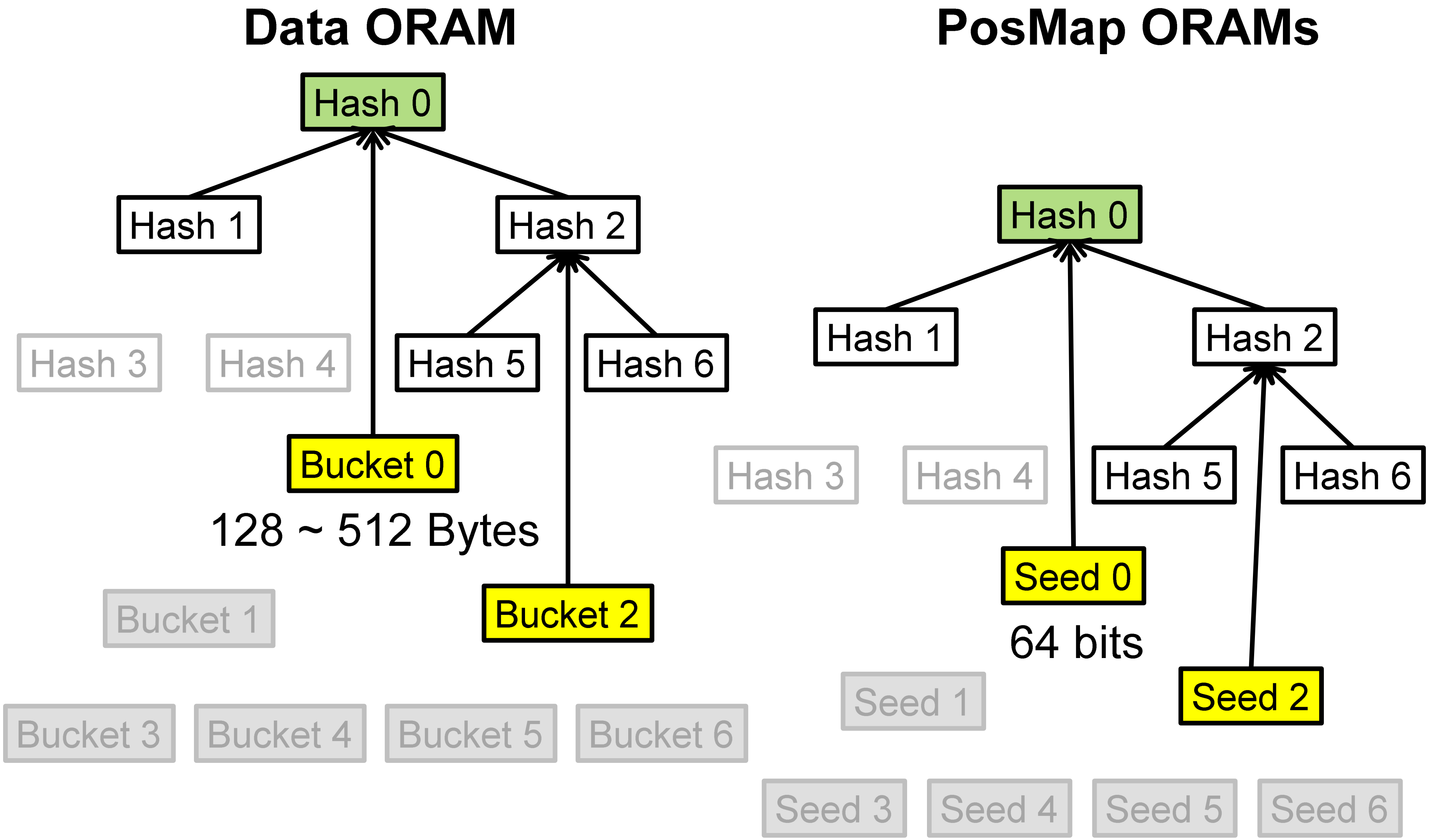
- Apply the scheme to every ORAM in the recursion
- Can we do better?
  - Hash latency $\propto$ hash input. Reduce hash input?
- Yes, we only need to integrity-verify data ORAM and the seeds in position map ORAMs.

- Pseudorandom generator (PSRG) $r = G_K(s)$
  - Seed s Secret key K
  - Output r looks random to anyone who does not know K

- Probabilistic encryption based on PSRG
  - To encrypt X, choose new s
  - $Y = G_K(s) \oplus X$ ciphertext (s, Y)

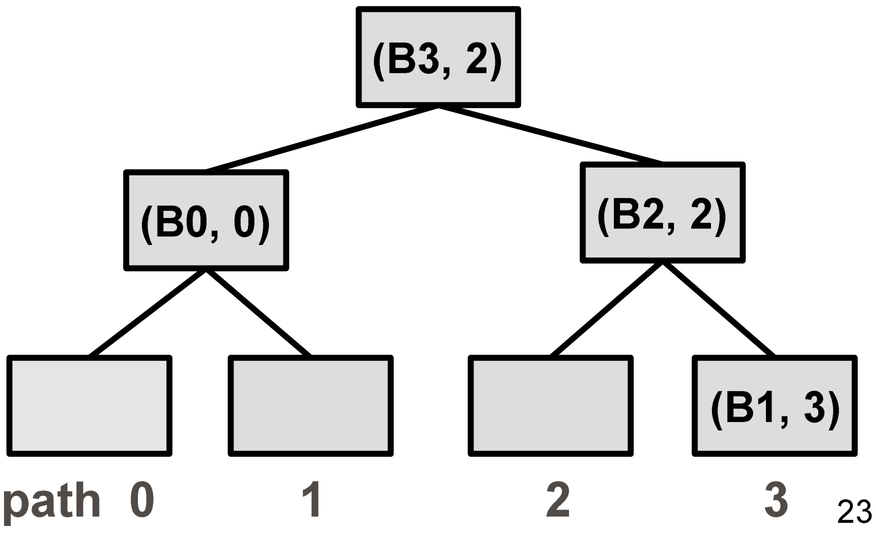
e.g. AES counter mode

Final Scheme



Proof

- **Only intuition here, details in paper.**
- **PosMap ORAMs just yield a leaf label for data ORAM**
  - (block, leaf label) tuple
  - If PosMap ORAM returns a wrong leaf label for data ORAM, it will be detected if compared with the verified leaf in data ORAM
- **Verify seeds to thwart the replay attack**



Evaluation

- **Setup**
  - 4 GB ORAM, 128 Byte block, three ORAMs in recursion
  - SHA-1 hash and AES-128 encryption
  - Built on commodity DDR3
- **Our integrity verification adds 17% latency on top of recursive Path ORAM**
  - 35% if verifying everything in PosMap ORAMs
  - 3x worse if directly using Merkle signature

Contributions

- Recursive Path ORAM is insecure w/o integrity verification
- An integrity verification scheme with only 17% overhead
- Ascend + verified Path ORAM + certified execution → privacy and integrity in cloud computing by trusting only hardware (not trusting any software)

Thank you! Questions?

Backup

Another Motivation

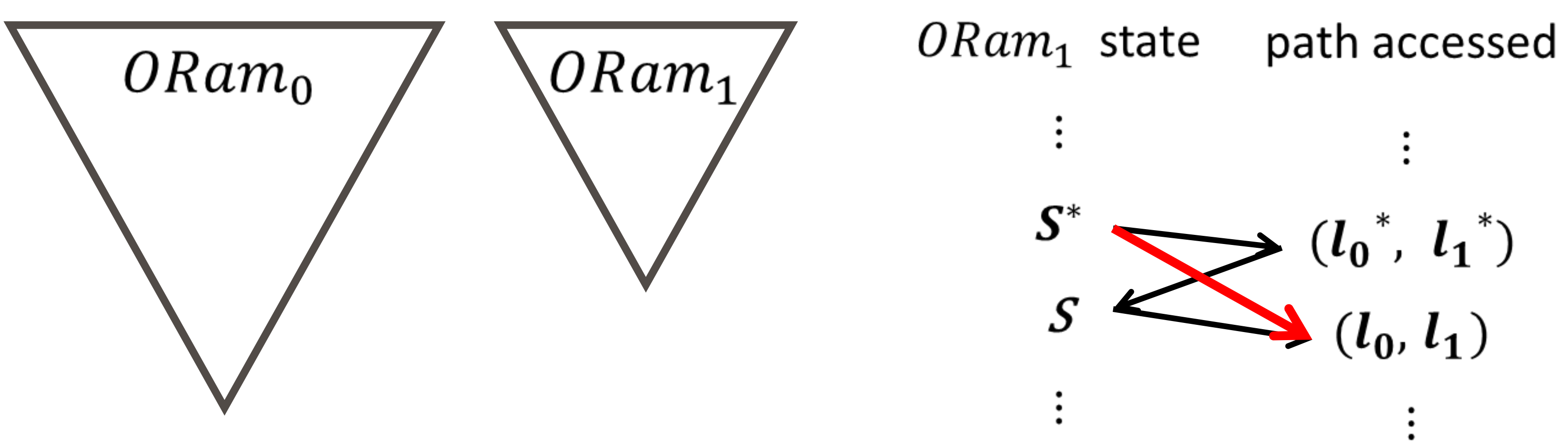
- **Recursive Path ORAM is broken when attackers can modify ORAM**
- **Replay attack to distinguish**

  * Access pattern (1) 0x00, 0x01, 0x02 … (2) 0x00, 0x00, 0x00

  – Find consecutive accesses such that $l_1 = l_1^*$

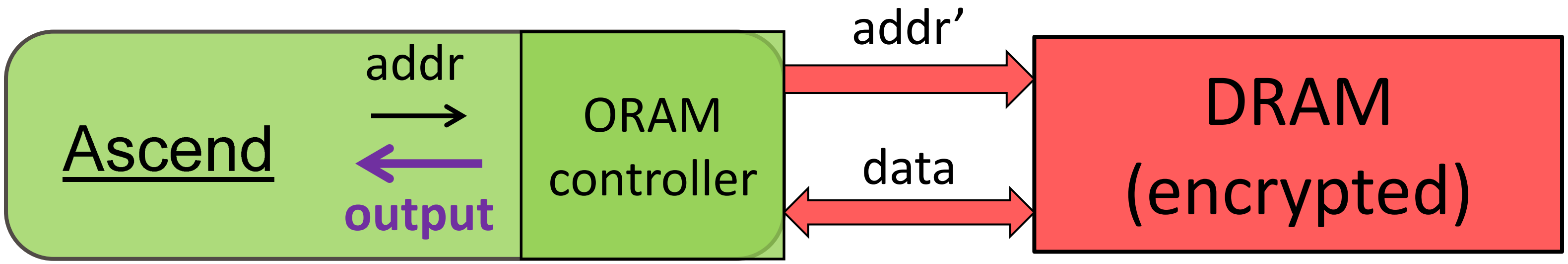
  – Revert $ORam_1$ from S to S^*

  – If $l_0 = l_0^*$, guess access pattern (2); otherwise guess (1)



Verify Recursive Path ORAMs

- Apply the scheme to every ORAM in the recursion
- Can we do better?
  - Hash latency ∝ hash input. Reduce hash input?
- Yes, if we follow a slightly relaxed security definition
  - **An integrity verification** for ORAMs **is secure, if no** computationally bounded adversaries with the ability to modify ORAMs can with non-negligible probability (1) **change the output** of the ORAM interface without being detected, or (2) **learn anything about the access pattern**.



Verify Recursive Path ORAMs

Theorem 1. *To integrity-verify a recursive Path ORAM, it suffices to integrity-verify data ORAM and the random seeds for position map ORAMs.*

$$\mathsf{encrypt}_K(X) = (s, G_K(s) \oplus X)$$

- **Proof outline**
  - l_j^i the path read and written for ORam$_i$ on the j-th ORAM access

    $1^0 = \mathsf{PosMap}(\mathbf{u})$
  - [Correctness] Data ORAM stores (address, data, leaf) triplets.
  - [Privacy] Modified ciphertexts decrypt into random bits → still access random paths

$$X' = G_K(s) \oplus Y'$$

Proof

Lemma 1. *Given* ORam_0 *is authentic and fresh, if* $\exists j$ *where* PosMap' *yields* $l_j^{0\prime} \neq l_j^0$*, then the ORAM interface can detect this when accessing* ORam_0.

a triplet (b_j, u_j, l_j^0) must be stored somewhere

$\mathsf{access}(\mathsf{ORam}_0, l_j^{0\prime})$, then either:

1) block b_j is not found along path $l_j^{0\prime}$ or the stash, and the ORAM interface knows $l_j^{0\prime}$ is wrong;
2) block b_j is found in the stash or on the common subpath of path $l_j^{0\prime}$ and path l_j^0, the ORAM interface compares $l_j^{0\prime}$ with the leaf label stored in the triplet and finds $l_j^{0\prime} \neq l_j^0$.

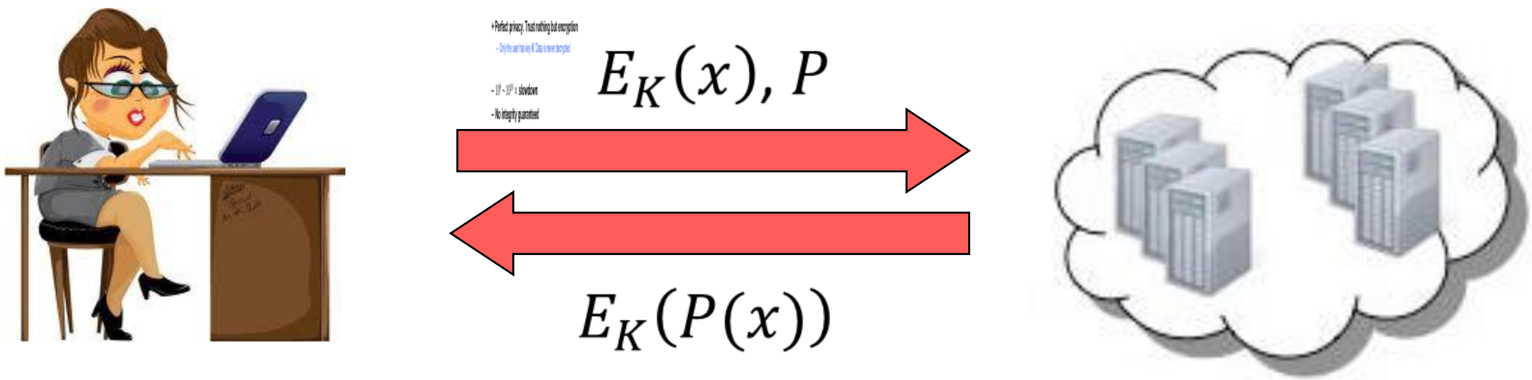
In either case, the ORAM interface can detect that position map ORAMs are tampered with.

Proof

Lemma 2. *Given the random seeds are authentic and fresh, whichever way an adversary tampers with any* ORam_i, $l_j^{i\prime}$ *is indistinguishable from uniformly random for any* i, j.

$$Y = G_K(s) \oplus X \qquad X' = G_K(s) \oplus Y'$$

Fully Homomorphic Encryption



- **+ Perfect privacy. Trust nothing but encryption**
  - Only the user has key ***K***. Data is never decrypted

- **–** $10^9 \sim 10^{18} \times$ **slowdown**
- **– No integrity guaranteed**