A Practical Oblivious Map Data Structure with Secure Deletion and History Independence

Daniel S. Roche  Adam J. Aviv  Seung Geol Choi

Computer Science Department
United States Naval Academy
Annapolis, Maryland, USA

San Jose, California
Goal: A remote key/value store with...

**Strong privacy**
- Hidden keys, values, and access patterns (Obliviousness)
- Secure against powerful attackers (Secure Deletion and History Independence)

**Practical utility**
- No computation on server
- Poly-logarithmic local storage, bandwidth, computation
- Low round complexity
Oblivious RAM

Oblivious RAM (ORAM) hides access patterns as well as data. (Goldreich & Ostrovsky JACM’96, and many more since then!)
Oblivious RAM (ORAM) hides access patterns as well as data. (Goldreich & Ostrovsky JACM’96, and many more since then!)
Problem 1

What if the size of data is not fixed?
ORAM reveals the number of operations, and therefore data size.

**Insecure solution**
Send **multiple blocks** depending on the data size

**Inefficient solution**
**Pad** all data up to the maximum size
Problem 1

What if the size of data is not fixed?
ORAM reveals the number of operations, and therefore data size.

Insecure solution
Send **multiple blocks** depending on the data size

Inefficient solution
**Pad** all data up to the maximum size

Our approach: Oblivious RAM with variable blocks (vORAM)
Hide large data in the overhead of Path ORAM,
Large data blocks are stored across multiple ORAM “buckets”.

Roche, Aviv, & Choi (USNA)
Oblivious Data Structures (ODS)

Storing a data structure in ORAM (Wang et. al, CCS’14)
Oblivious Data Structures (ODS)

Storing a data structure in ORAM (Wang et. al, CCS’14)

Pieces of data structure (i.e., nodes) are stored in ORAM blocks.
Problem 2

What if your data structure has varying running time?
The number of memory accesses in each operation are leaked by ORAM.

Insecure solution
Let the number of operations vary by access

Inefficient solution
Perform dummy operations up to the worst-case cost
Problem 2

What if your data structure has varying running time?
The number of memory accesses in each operation are leaked by ORAM.

Insecure solution
Let the number of operations vary by access

Inefficient solution
Perform dummy operations up to the worst-case cost

Our approach: History-Independent Randomized B Tree (HIRB)
Use a fixed-height tree data structure, so that no padding is necessary.
“Catastrophic” Attacks

An attacker may be able to coerce the private key.
“Catastrophic” Attacks

An attacker may be able to **coerce the private key**.
Problem 3

What if your private key is compromised?

- Some leakage is inevitable
- ORAM reveals entire history, including prior deletions
- Most data structures also leak history information

Inefficient solution

Re-encrypt and transfer entire data set on every access
Problem 3

What if your private key is compromised?

- Some leakage is inevitable
- ORAM reveals entire history, including prior deletions
- Most data structures also leak history information

Inefficient solution
Re-encrypt and transfer entire data set on every access

Our approach (vORAM+HIRB)

HIRB data structure leaks no history nor prior deletions.
vORAM leaks minimal history and no prior deletions.
Outline and Related Work

1. Problem Statement and Goals

2. vORAM: Oblivious RAM with variable-sized blocks
   - Path ORAM (Stefanov et al., CCS’13)
   - Secure deletion B-tree (Reardon et al., CCS’13)

3. HIRB: History Independent Randomized B-Tree
   - Oblivious Data Structures (Wang et al., CCS’14)
   - History-Independent Data Structures (Naor & Teague ’01, Hartline et al. ’02, Golovin ’08)

4. Experimental Results
Path ORAM with Variable-Sized Blocks: vORAM

General idea: Large items are rare; distribute their bits along an ORAM path.

Terminology: Each tree node is a bucket stored on the server. The user stores blocks of data. Each block may be broken up into chunks of bytes.

Crucial restrictions:
- All chunks of the same block are on the same path
- Chunks of the same block are always in order
vORAM Example: Setup

Stored blocks: [ Yellow, Purple, Blue, Green ]

**Color** represents data, **Width** = size, **Number** = position.
vORAM Example: Update

Stash:

UPDATE (yellow): Evict, Re-assign, Writeback
vORAM Example: Update

UPDATE([1, 3]): Evict, Re-assign, Writeback

Stash: [6, 3]
vORAM Example: Update

**UPDATE** (Evict, Re-assign, Writeback)
More details on vORAM

- Identifiers are chosen randomly, and the position (leaf node index) is a prefix of the identifier.

- The entire path is fetched and returned in parallel, resulting in 2 rounds per operation.

- Each node encrypted with a key stored in the parent node that is refreshed on each operation — implies secure deletion.

- No history beyond the most recent $O(n/\log n)$ operations is revealed, matching an asymptotic lower bound.
How big should the buckets be?

An crucial parameter is **bucket size**: number of bytes per bucket.

As with Path ORAM, if this is too small, the root node (or stash) will “overflow”.

**Theorem**

*The vORAM stash will overflow with only negligible probability if:*

- Block sizes are **bounded by a geometric distribution**
- Bucket size is **20 times the expected block size**

**Note:** In practice, the constant can be only 6, not 20.
Recall the identifiers in vORAM: 4 6

These identifiers are random; where do we store them?
Oblivious Data Structures

Recall the identifiers in vORAM: 4 6

These identifiers are random; where do we store them?

- **Standard solution**: Store a position map in recursively smaller ORAMs

- **ODS (Wang et al. ‘14)**: If you’re storing a data structure, store each node’s identifier in its parent node!

  To store a key/value map, use an AVL tree.
We want to store a key/value data structure within the vORAM.

But most data structures leak history information!

**Example: AVL Tree Leakage**

- Were you browsing reddit or youtube?
Example: AVL Tree Leakage

- We want to store a key/value data structure within the vORAM.
- But most data structures leak history information!

Were you browsing reddit or youtube?
Overview:

- B-tree structure, but the height of each element is uniquely determined.

- Heights determined from a *randomly-selected hash function*.

- The keys of key/value pairs are not stored, only their hashes.

- **Strong history independence** (Naor & Teague, STOC’01): The contents of the tree uniquely determine its structure.
**Over-simplification:** Height = number of trailing zeros in hash
Over-simplification: Height = number of trailing zeros in hash

Example: Insert HELLO

hash(HELLO) = 510
**Over-simplification**: Height = number of trailing zeros in hash

**Example**: Insert HELLO

\[ \text{hash(HELLO)} = 510 \]
Choosing the heights

At tree creation: Choose a random hash function.

Crucial parameter: $\beta$, the expected block size

Given an element: Compute its hash, to seed a PRNG. Sample from a geometric distribution with probability $\frac{\beta - 1}{\beta}$ to determine the height.
The HIRB is perfectly suited for vORAM:

- Node sizes follow a geometric distribution
- Identifiers can be stored in parent nodes
- Height is fixed — no padding necessary
- Combination still provides secure deletion
- HIRB leaks no operation history beyond what vORAM inevitably leaks
Comparison baselines

- **vORAM+HIRB**: Good performance, near-best security.

- Path ORAM with AVL tree: Poor performance, no secure deletion.
  Uses padding for obliviousness.

- Secure deletion B-tree: Best performance, no obliviousness.
  A normal B-tree, re-encrypting nodes on each access.

- Naïve baseline: Worst performance, best security.
  Re-encrypt and transfer the entire dataset on each access.

All implemented by us, in Python3, and tested using Amazon AWS.
Biggest Factors of Performance Improvement

- **Height** of HIRB compared to AVL tree
- **Larger nodes** in HIRB to take advantage of block size
- **Efficient block packing** in vORAM
- **Parallel fetching** of paths from vORAM
- All leads to significantly reduced *round complexity*
Experimental Timings

Median Access Time (s) (logscale)
Number of Entries (logscale)
vORAM+HIRB
Path ORAM with AVL tree
Secure deletion B-tree
Naive Baseline

Roche, Aviv, & Choi (USNA)
May 23, 2016 23 / 24
Take-Aways

- **ORAMs don’t (have to) suck.**
  Our construction has practical utility in a real cloud setting.

- **We can get more flexibility and privacy from ORAMs.**
  We support variable-size blocks, secure deletion, and (limited) history independence.

- **Specialized data structures are needed to work well in ORAMs**
  Our HIRB tree is ideally suited for vORAM.
Take-Aways

- **ORAMs don’t (have to) suck.**
  Our construction has practical utility in a real cloud setting.

- **We can get more flexibility and privacy from ORAMs.**
  We support variable-size blocks, secure deletion, and (limited) history independence.

- **Specialized data structures are needed to work well in ORAMs**
  Our HIRB tree is ideally suited for vORAM.

Thank You!

Daniel S. Roche, Adam Aviv, and Seung Geol Choi (U.S. Naval Academy)
A Practical Oblivious Map Data Structure with Secure Deletion and History Independence