

CSci 8271  
Security and Privacy in Computing  
Day 3: Path ORAM

Stephen McCamant  
University of Minnesota

### Hiding access pattern leakage

- When outsourcing data, encryption can hide the data itself
- But the sequence of accesses might also reveal information
  - Including locations, read vs. write, and repeated accesses
- An ORAM is a randomized data structure where the access pattern does not reveal information
  - Fixed, or random from a fixed distribution

### A trivial solution

- On every read or write, access and re-encrypt every block of data
- ORAMs use randomized encryption that does not reveal equalities
- Secure, but very inefficient

### Position map and shuffling

- The client can remember the location of blocks held on the server
  - Space usage not bad if blocks are large relative to block pointers ( $x$ )
- Blocks can be shuffled in an oblivious way
- This would be enough if we never wanted to access the same block again

### Local caching

- One way to deal with repeated accesses is to save blocks on the client
  - Make dummy access to another location when using from cache
  - Reshuffle everything when the client cache is full
- Called "square root" ORAM based on the size of the client cache

### Tradeoffs in relocation

- Need to move blocks when accessing them
  - To not reveal duplicate accesses
- Need to move more than one block per update
  - Otherwise, still reveals identity
- Accessed block should move to one of many locations
- But want to not do too many moves, for efficiency

### Tree structure for Path ORAM

- Organize the storage like a complete binary tree
- Each node is a bucket holding a handful (4-7) of blocks
- Position map maps each block to a leaf of the tree, randomly
- A block is stored somewhere on the path from the root to its leaf
  - Or in an overflow client "stash"

### Update rules

- When accessing a block, process the entire path where it might be found, and choose a new leaf for it
- Rules when writing blocks back to the path:
  - Each block must stay on the path to its leaf
  - Subject to (1), move blocks closer to the leaves if possible
- On every update, opportunistically moves unrelated blocks towards the leaves

## Position map recursion

- Can trade-off access steps to get lower client storage
- Instead of keeping the whole position map on the client, store it in its own, smaller, ORAM
  - Can repeat until the client position map is constant size

## Adding integrity checking

- A standard simplifying assumption is that the server is "honest but curious"
  - Tries to glean information but still follows the protocol
- If you also need to guard against server changes, make the tree also a Merkle tree
  - I.e., each node includes a cryptographic hash of its children

## Bucket and stash sizing

- Buckets should hold at least 4 blocks to empirically avoid overflow
  - 7, or 6 with more nodes, is provably sufficient
- Stash size: doesn't depend on number of blocks
  - Linear stash increase ensures exponentially decreased failure rate

## Proof techniques

- Effect of full buckets is tricky to reason about
  - Pretend they are unlimited in operation, then post-processed to fit size
- Separately bound the risk of overflow out of any partial subtree including the root