Weighted ORAM, with Applications to Searchable Symmetric Encryption

Léonard Assouline   Brice Minaud

Eurocrypt 2023
April 25th

Motivation (from https://signal.org/blog/building-faster-oram/)

Alice downloads the Signal app.
Wants to check with Signal's server if her contact Bob uses Signal.

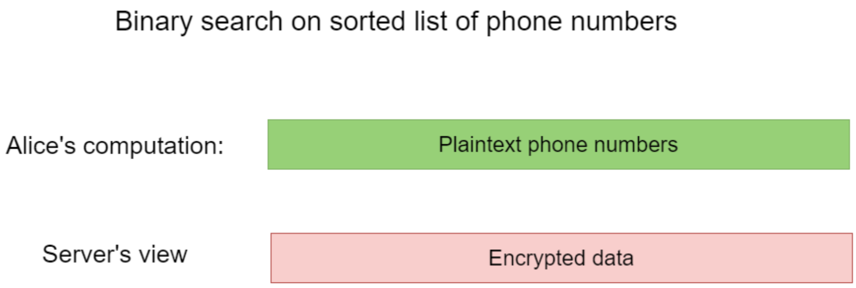


Figure: Looking for Bob's phone number: 212-555-2368

Motivation (from https://signal.org/blog/building-faster-oram/)

Alice downloads the Signal app.
Wants to check with Signal's server if her contact Bob uses Signal.

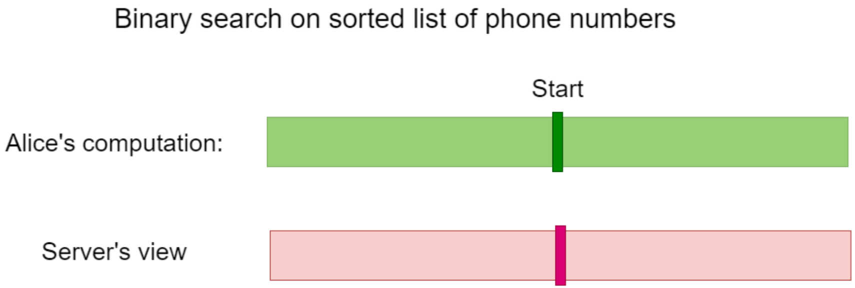


Figure: Looking for Bob's phone number: 212-555-2368

Motivation (from https://signal.org/blog/building-faster-oram/)

Alice downloads the Signal app.
Wants to check with Signal's server if her contact Bob uses Signal.

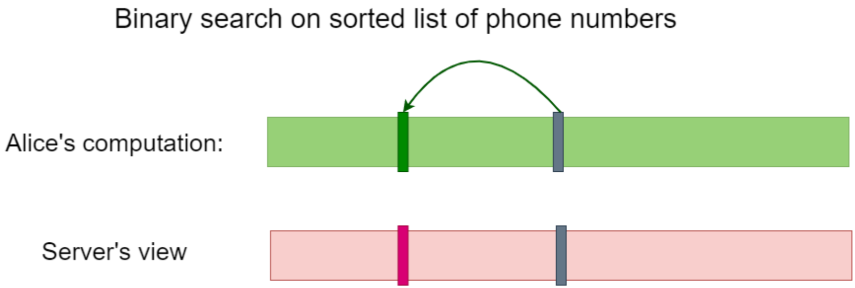


Figure: Looking for Bob's phone number: 212-555-2368

Motivation (from https://signal.org/blog/building-faster-oram/)

Alice downloads the Signal app.
Wants to check with Signal's server if her contact Bob uses Signal.

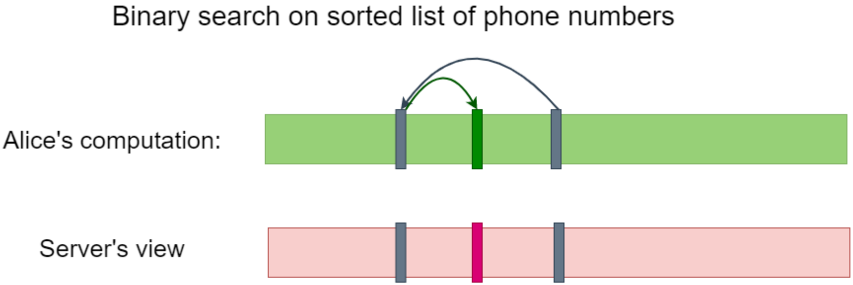


Figure: Looking for Bob's phone number: 212-555-2368

Motivation (from https://signal.org/blog/building-faster-oram/)

Alice downloads the Signal app.
Wants to check with Signal's server if her contact Bob uses Signal.

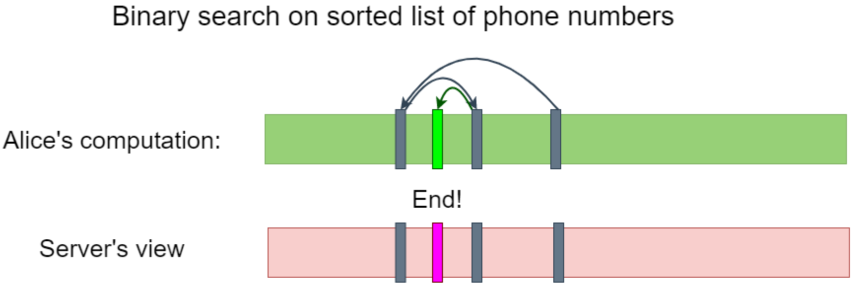


Figure: Looking for Bob's phone number: 212-555-2368

Despite encryption, information leaks

Motivation (from https://signal.org/blog/building-faster-oram/)

Alice downloads the Signal app.
Wants to check with Signal's server if her contact Bob uses Signal.

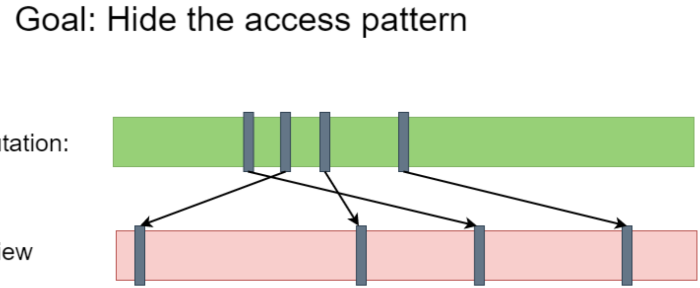


Figure: Looking for Bob's phone number: 212-555-2368

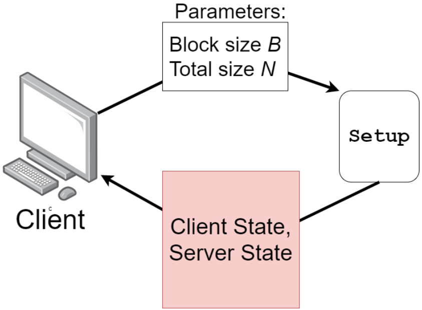
Despite encryption, information leaks

Oblivious RAM framework

Introduced by Goldreich in 1987: obfuscate the access pattern.
ORAM protocol: tuple (`Setup`, `Access`):

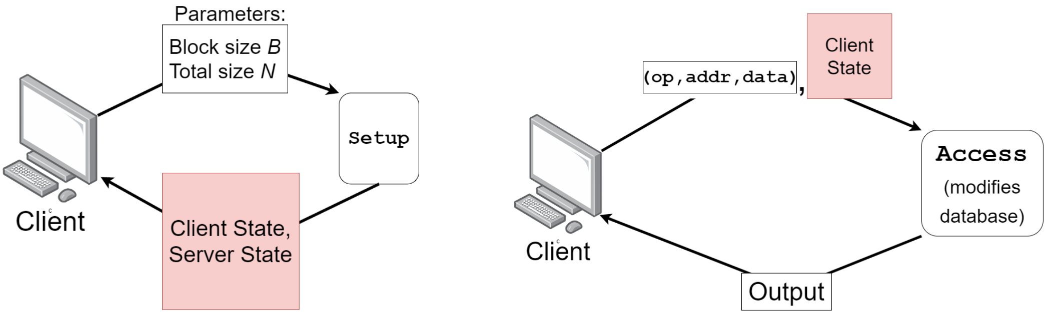
Oblivious RAM framework

Introduced by Goldreich in 1987: obfuscate the access pattern.
ORAM protocol: tuple (`Setup`, `Access`):



Oblivious RAM framework

Introduced by Goldreich in 1987: obfuscate the access pattern.
ORAM protocol: tuple (`Setup`, `Access`):



ORAM Security

Each access:

- $\texttt{op} \in \{\texttt{write}, \texttt{read}\}$: type of operation.
- $\texttt{addr}$: address of object.
- $\texttt{data}$: new value (if $\texttt{op} = \texttt{read}$, $\mathit{data} = \bot$).

ORAM Security

Each access:

- $\texttt{op} \in \{\texttt{write}, \texttt{read}\}$: type of operation.
- $\texttt{addr}$: address of object.
- $\texttt{data}$: new value (if $\texttt{op} = \texttt{read}$, $data = \perp$).

Definition: Security

Let: $\mathbf{s} = (op_i, addr_i, data_i)_{i \in [m]}$, $\mathbf{t} = (op'_i, addr'_i, data'_i)_{i \in [m]}$: sequences of accesses of same size.
ORAM secure iff, in server's view: $\mathbf{s} \approx \mathbf{t}$.

ORAM Security

Each access:

- $\mathtt{op} \in \{\mathtt{write}, \mathtt{read}\}$: type of operation.
- $\mathtt{addr}$: address of object.
- $\mathtt{data}$: new value (if $\mathtt{op} = \mathtt{read}$, $data = \perp$).

Definition: Security

Let: $\mathbf{s} = (op_i, addr_i, data_i)_{i\in[m]}$, $\mathbf{t} = (op'_i, addr'_i, data'_i)_{i\in[m]}$: sequences of accesses of same size.
ORAM secure iff, in server's view: $\mathbf{s} \approx \mathbf{t}$.

Proving security: easy part (access patterns are information theoretically hidden).

ORAM Security

Each access:

- $\texttt{op} \in \{\texttt{write}, \texttt{read}\}$: type of operation.
- $\texttt{addr}$: address of object.
- $\texttt{data}$: new value (if $\texttt{op} = \texttt{read}$, $data = \perp$).

Definition: Security

Let: $\mathbf{s} = (op_i, addr_i, data_i)_{i \in [m]}$, $\mathbf{t} = (op'_i, addr'_i, data'_i)_{i \in [m]}$: sequences of accesses of same size.
ORAM secure iff, in server's view: $\mathbf{s} \approx \mathbf{t}$.

Proving security: easy part (access patterns are information theoretically hidden).

Challenge: prove correctness (does the ORAM run without failure?)

This Work

Goal: ORAM that handles many objects of different sizes, without changing communication cost

This Work

Goal: ORAM that handles many objects of different sizes, without changing communication cost

Naïve solutions:

- padding (to largest object size) $\rightarrow$ inefficient

This Work

Goal: ORAM that handles many objects of different sizes, without changing communication cost

Naïve solutions:

- padding (to largest object size) $\rightarrow$ inefficient
- divide into regular small chunks $\rightarrow$ too many accesses

This Work

Goal: ORAM that handles many objects of different sizes, without changing communication cost

Naïve solutions:

- padding (to largest object size) $\rightarrow$ inefficient
- divide into regular small chunks $\rightarrow$ too many accesses

Our solution: Build ORAM for total size N, handles $m > N$ objects, each of weight w_i

Constraint: $\sum_{i=1}^{m} w_i \leq N$ and $\forall i \in [m], w_i \leq 1$

As long as constraint is respected, w_i can change after a `Write`.

This Work

Goal: ORAM that handles many objects of different sizes, without changing communication cost

Naïve solutions:

- padding (to largest object size) $\rightarrow$ inefficient
- divide into regular small chunks $\rightarrow$ too many accesses

Our solution: Build ORAM for total size N, handles $m > N$ objects, each of weight w_i

Constraint: $\sum_{i=1}^{m} w_i \leq N$ and $\forall i \in [m], w_i \leq 1$

As long as constraint is respected, w_i can change after a `Write`.

We call that a

Weighted Oblivious RAM (wORAM)

Existing ORAM paradigms

- Trivial: download everything
- Hierarchical ORAM
- Tree-ORAM (focus of this work)

Existing ORAM paradigms

- Trivial: download everything
- Hierarchical ORAM
- Tree-ORAM (focus of this work)

Also results (with complexity blowup) for arbitrary ORAM protocols.

Existing ORAM paradigms

- Trivial: download everything
- Hierarchical ORAM
- Tree-ORAM (focus of this work)

Also results (with complexity blowup) for arbitrary ORAM protocols.

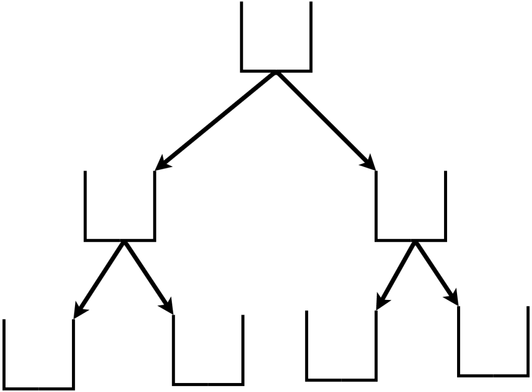
Next:

- Path-ORAM
- Generic Criterion
- Proof

Path-ORAM (Stefanov et al, 2013)

Idea

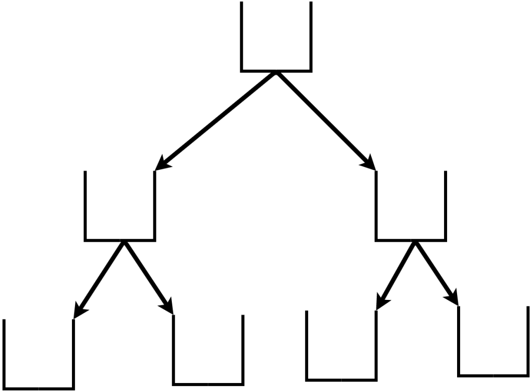
- Store N objects (blocks) in several buckets, each with max Z blocks
- Store buckets in complete binary tree of depth $\approx log(N)$
- Associate block with leaf: block is in a bucket along path to leaf



Path-ORAM (Stefanov et al, 2013)

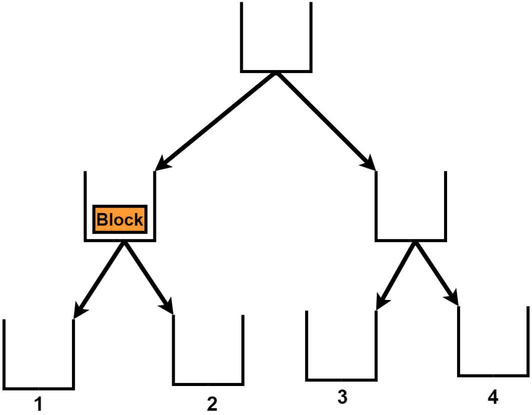
Idea

- Store N objects (blocks) in several buckets, each with max Z blocks
- Store buckets in complete binary tree of depth $\approx log(N)$
- Associate block with leaf: block is in a bucket along path to leaf



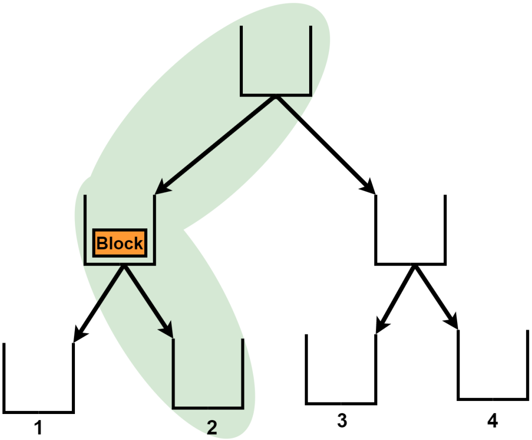
Path-ORAM (Stefanov et al, 2013)

`Access`(orange block), associated with leaf 2



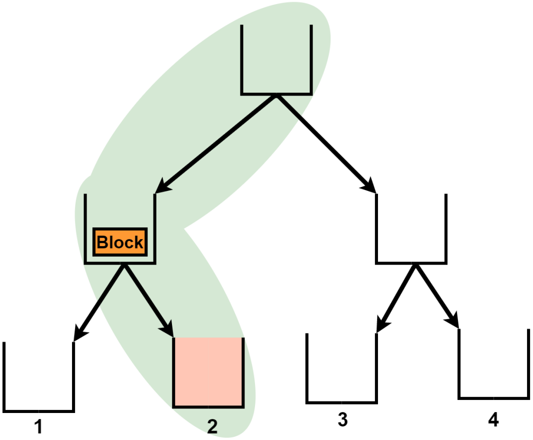
Path-ORAM (Stefanov et al, 2013)

Identify associated path (leaf 2)



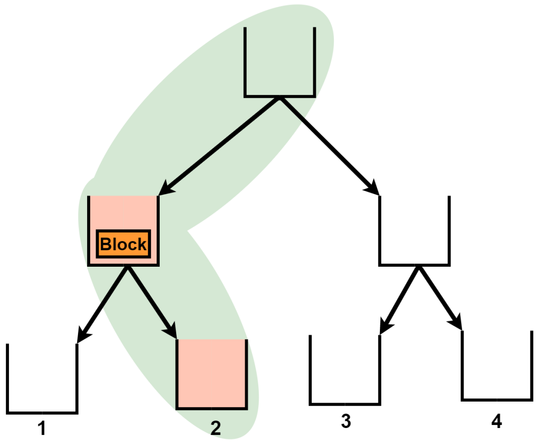
Path-ORAM (Stefanov et al, 2013)

Download each bucket in path



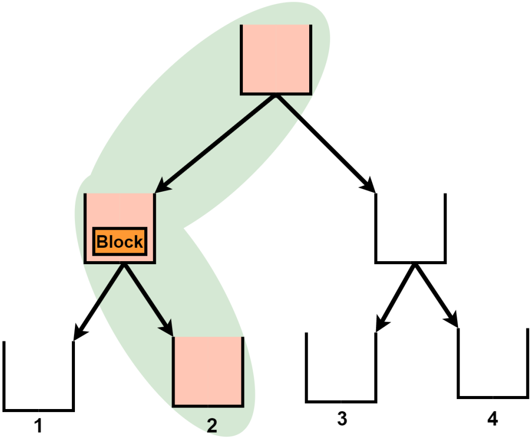
Path-ORAM (Stefanov et al, 2013)

Download each bucket in path



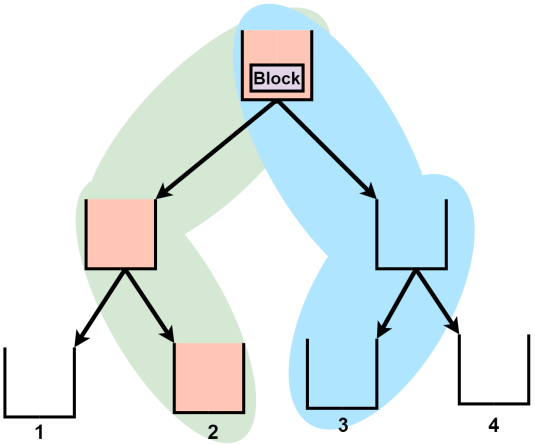
Path-ORAM (Stefanov et al, 2013)

Download each bucket in path



Path-ORAM (Stefanov et al, 2013)

Modify block's content and reencrypt it (orange → grey)
Sample new leaf randomly (leaf 3)
Write back at intersection of paths

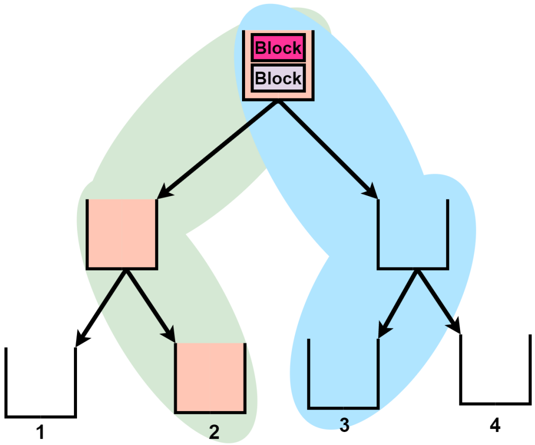


Path-ORAM (Stefanov et al, 2013)

Modify block's content and reencrypt it (orange $\rightarrow$ grey)

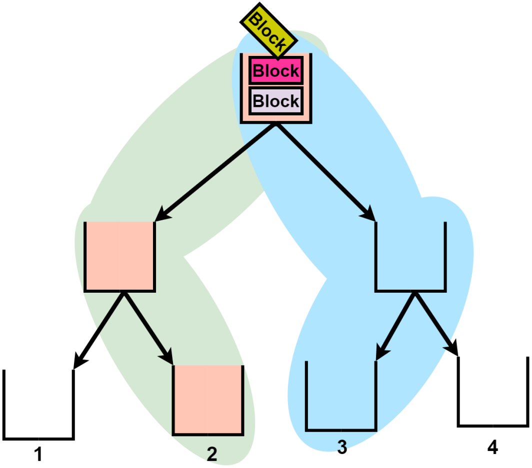
Sample new leaf randomly (leaf 3)

Write back at intersection of paths



Path-ORAM (Stefanov et al, 2013)

What if I run out of space?



Offline memory: client stash

Client has stash of size $\omega(log(N))$, stores blocks when unable to write them online.

Offline memory: client stash

Client has stash of size $\omega(log(N))$, stores blocks when unable to write them online.

Offline memory: client stash

Client has stash of size $\omega(log(N))$, stores blocks when unable to write them online.

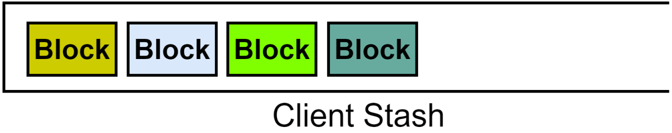
Security: new random leaf (i.e. path) every access.

Offline memory: client stash

Client has stash of size $\omega(log(N))$, stores blocks when unable to write them online.

Security: new random leaf (i.e. path) every access.

Correctness: Stash never overflows (with overwhelming probability).



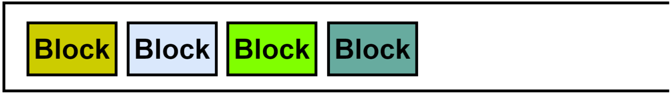
Offline memory: client stash

Client has stash of size $\omega(log(N))$, stores blocks when unable to write them online.

Security: new random leaf (i.e. path) every access.

Correctness: Stash never overflows (with overwhelming probability).

From Path-ORAM paper: $\mathbb{P}(|\text{stash}| > R) \leq 14 \cdot (0.6002)^R$



Client Stash

Offline memory: client stash

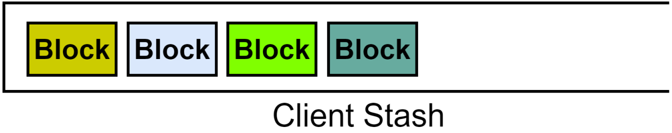
Client has stash of size $\omega(log(N))$, stores blocks when unable to write them online.

Security: new random leaf (i.e. path) every access.

Correctness: Stash never overflows (with overwhelming probability).

From Path-ORAM paper: $\mathbb{P}(|\text{stash}| > R) \leq 14 \cdot (0.6002)^R$

Our contribution: Transformation to handle blocks of variable sizes.



Standard Tree-ORAM protocol → Weighted Tree-ORAM

- m blocks, each of size $w_i \leq B$

Standard Tree-ORAM protocol $\rightarrow$ Weighted Tree-ORAM

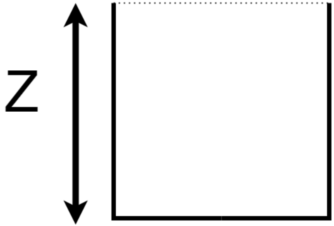
- m blocks, each of size $w_i \leq B$
- $\sum w_i = N \cdot B$

Standard Tree-ORAM protocol $\rightarrow$ Weighted Tree-ORAM

- m blocks, each of size $w_i \leq B$
- $\sum w_i = N \cdot B$ (Consider $B = 1$)

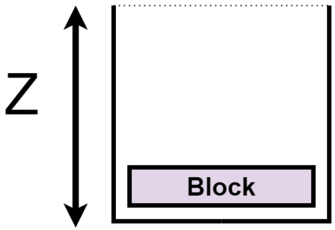
Standard Tree-ORAM protocol $\rightarrow$ Weighted Tree-ORAM

- m blocks, each of size $w_i \leq B$
- $\sum w_i = N \cdot B$ (Consider $B = 1$)
- Buckets: Can store objects until threshold Z is reached (total capacity $Z + 1$)



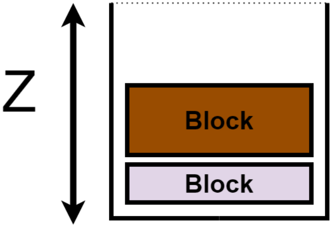
Standard Tree-ORAM protocol → Weighted Tree-ORAM

- m blocks, each of size $w_i \leq B$
- $\sum w_i = N \cdot B$ (Consider $B = 1$)
- Buckets: Can store objects until threshold Z is reached (total capacity $Z + 1$)



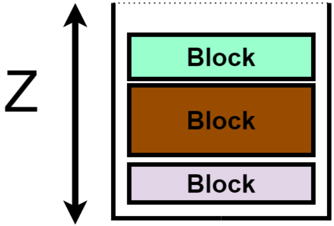
Standard Tree-ORAM protocol → Weighted Tree-ORAM

- m blocks, each of size $w_i \leq B$
- $\sum w_i = N \cdot B$ (Consider $B = 1$)
- Buckets: Can store objects until threshold Z is reached (total capacity $Z + 1$)



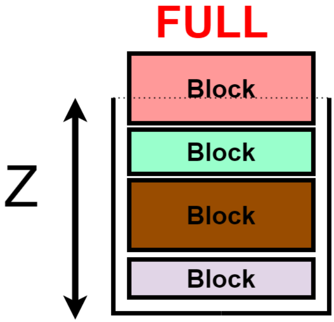
Standard Tree-ORAM protocol → Weighted Tree-ORAM

- m blocks, each of size $w_i \leq B$
- $\sum w_i = N \cdot B$ (Consider $B = 1$)
- Buckets: Can store objects until threshold Z is reached (total capacity $Z + 1$)



Standard Tree-ORAM protocol $\rightarrow$ Weighted Tree-ORAM

- m blocks, each of size $w_i \leq B$
- $\sum w_i = N \cdot B$ (Consider $B = 1$)
- Buckets: Can store objects until threshold Z is reached (total capacity $Z + 1$) , remaining blocks stay in the stash.



Main Theorem

Consider an ORAM protocol. If:

1. Reading a bucket is done via a Trivial ORAM
2. Stash load comes from collection of subsets of buckets in ∞-ORAM
3. For any subset in this collection, overflow is negligible

Then this ORAM can be turned into a weighted ORAM.

Proof of correctness

From Path-ORAM paper

Given a sequence of accesses $\mathbf{s} = (op_i, addr_i, data_i)_{i \in [m]}$,

1. Consider execution of $\mathbf{s}$ on the ∞-*ORAM* $(Z = \infty)$

Proof of correctness

From Path-ORAM paper

Given a sequence of accesses $\mathbf{s} = (op_i, addr_i, data_i)_{i \in [m]}$,

1. Consider execution of $\mathbf{s}$ on the ∞-*ORAM* ($Z = \infty$)
2. Apply post-processing algorithm G_Z

Proof of correctness

From Path-ORAM paper

Given a sequence of accesses $\mathbf{s} = (op_i, addr_i, data_i)_{i \in [m]}$,

1. Consider execution of $\mathbf{s}$ on the ∞-*ORAM* ($Z = \infty$)
2. Apply post-processing algorithm G_Z
3. Prove that : normal ORAM's stash load $=$ ∞-ORAM's stash load after applying G_Z

Proof of correctness

From Path-ORAM paper

Given a sequence of accesses $\mathbf{s} = (op_i, addr_i, data_i)_{i \in [m]}$,

1. Consider execution of $\mathbf{s}$ on the ∞-*ORAM* ($Z = \infty$)
2. Apply post-processing algorithm G_Z
3. Prove that : normal ORAM's stash load $=$ ∞-ORAM's stash load after applying G_Z
4. For ∞-*ORAM*, prove $\mathbb{P}$(stash overflow) is negligible.

Proof of correctness

From Path-ORAM paper

Given a sequence of accesses $\mathbf{s} = (op_i, addr_i, data_i)_{i \in [m]}$,

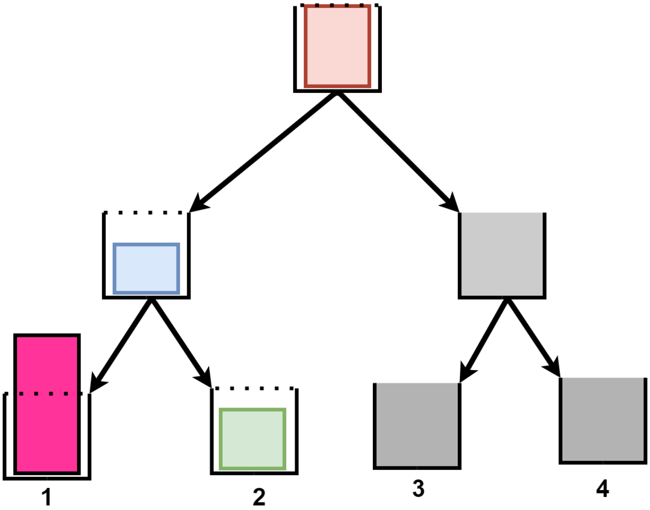
1. Consider execution of $\mathbf{s}$ on the ∞-*ORAM* ($Z = \infty$)
2. Apply post-processing algorithm G_Z
3. Prove that : normal ORAM's stash load $=$ ∞-ORAM's stash load after applying G_Z
4. For ∞-*ORAM*, prove $\mathbb{P}$(stash overflow) is negligible.

For weighted objects (this work):

5. Prove that standard ORAM size distribution is the worst case. Thus, $\mathbb{P}$(stash overflow) is negligible in our case too.

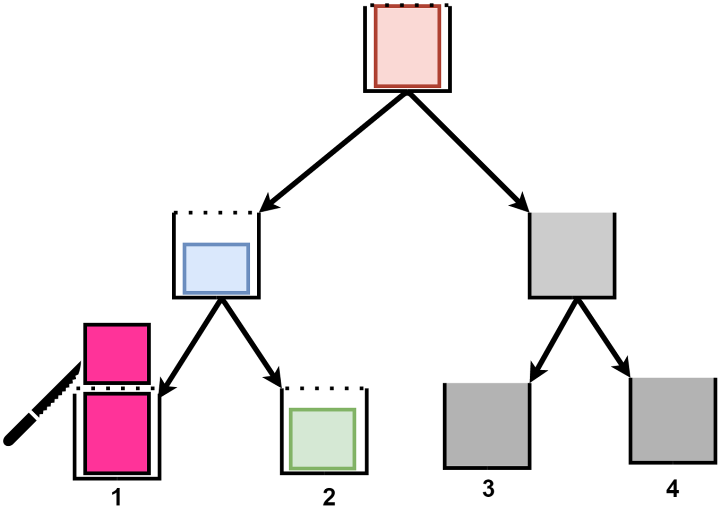
Post-processing algorithm

State of the ∞-ORAM after execution:



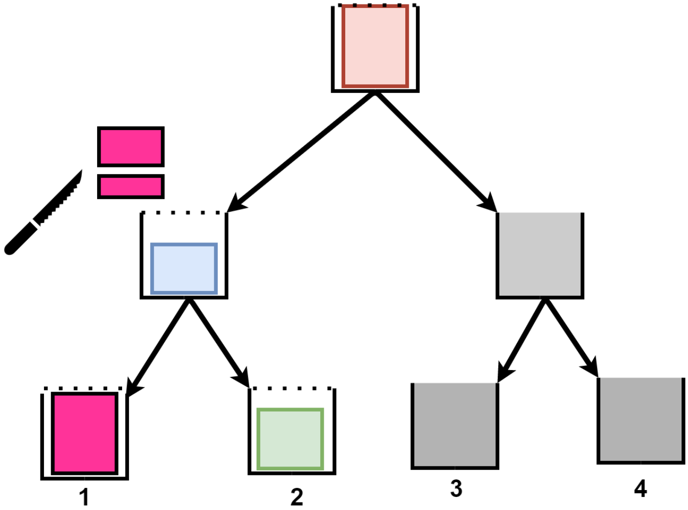
Post-processing algorithm

Application of G_Z:



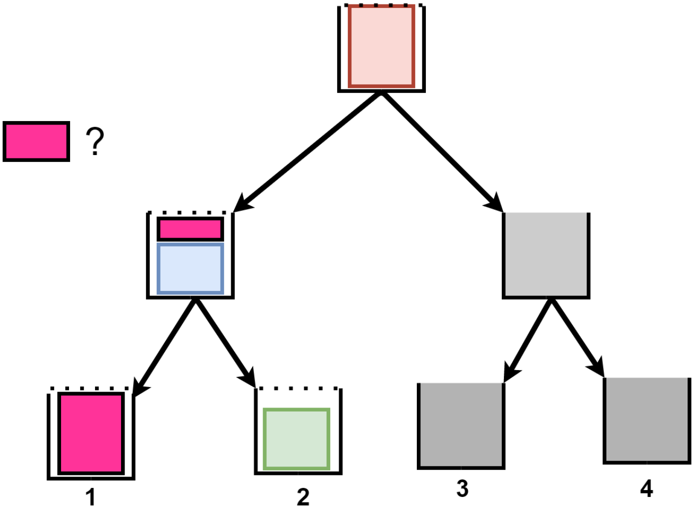
Post-processing algorithm

Application of G_Z:



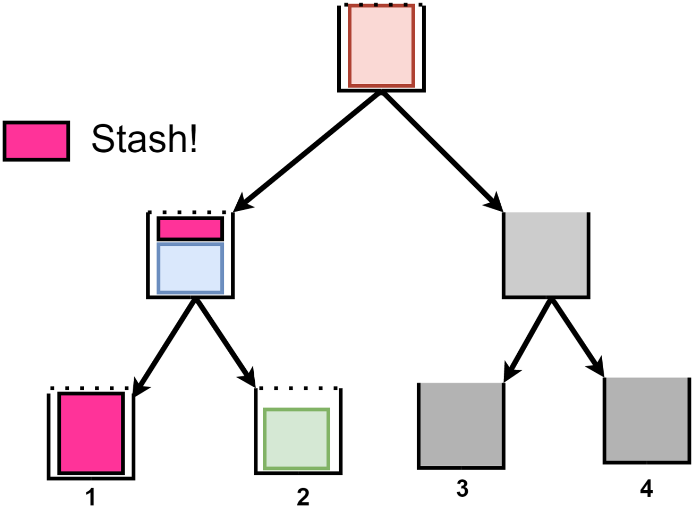
Post-processing algorithm

Application of G_Z:



Post-processing algorithm

Application of G_Z:



Reduction to the standard case

- Notice that stash load of $G_Z(\infty\text{-ORAM}) \geq$ stash load of ORAM.

Reduction to the standard case

- Notice that stash load of $G_Z(\infty\text{-ORAM}) \geq$ stash load of ORAM.
- We have m objects, with weights $\mathbf{w} \in [0,1]^m$ s.t. $\sum w_i \leq N$.

Reduction to the standard case

- Notice that stash load of $G_Z(\infty\text{-ORAM}) \geq$ stash load of ORAM.
- We have m objects, with weights $\mathbf{w} \in [0,1]^m$ s.t. $\sum w_i \leq N$.
- For a given access sequence $\mathbf{s}$, let $X(\mathbf{w})$ be the random variable of max stash load in post-processed ∞-ORAM for any permutation of $\mathbf{w}$.

Reduction to the standard case

- Notice that stash load of $G_Z(\infty\text{-ORAM}) \geq$ stash load of ORAM.
- We have m objects, with weights $\mathbf{w} \in [0,1]^m$ s.t. $\sum w_i \leq N$.
- For a given access sequence $\mathbf{s}$, let $X(\mathbf{w})$ be the random variable of max stash load in post-processed ∞-ORAM for any permutation of $\mathbf{w}$.
- We show that $\forall \mathbf{w}$, $\mathbb{E}(X(\mathbf{w})) \leq \mathbb{E}(X(\mathbf{u}))$ where $\mathbf{u} = (\underbrace{1,\ldots,1}_{N},\underbrace{0,\ldots,0}_{m-N})$.
(Corresponds to standard case, where correctness is proven)

Majorization argument

For a vector $\mathbf{v}$, define $\mathbf{v}^{\downarrow}$ as $\mathbf{v}$ with components sorted in decreasing order.

Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^m$ such that $\sum_{i=1}^{m} v_i = \sum_{i=1}^{m} w_i$

Majorization argument

For a vector $\mathbf{v}$, define $\mathbf{v}^{\downarrow}$ as $\mathbf{v}$ with components sorted in decreasing order.

Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^m$ such that $\sum_{i=1}^{m} v_i = \sum_{i=1}^{m} w_i$

v majorizes w ($\mathbf{w} \prec \mathbf{v}$) if: $\forall k \in [m]$, $\sum_{i=1}^{k} v_i^{\downarrow} \geq \sum_{i=1}^{k} w_i^{\downarrow}$.

Majorization argument

For a vector $\mathbf{v}$, define $\mathbf{v}^{\downarrow}$ as $\mathbf{v}$ with components sorted in decreasing order.

Let $\mathbf{v}, \mathbf{w} \in \mathbb{R}^m$ such that $\sum_{i=1}^{m} v_i = \sum_{i=1}^{m} w_i$

v majorizes w ($\mathbf{w} \prec \mathbf{v}$) if: $\forall k \in [m]$, $\sum_{i=1}^{k} v_i^{\downarrow} \geq \sum_{i=1}^{k} w_i^{\downarrow}$.

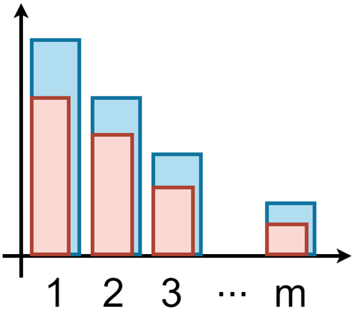


Figure: $\mathbf{w} \prec \mathbf{v}$

Proof

Lemma

If:

- $f : \mathbf{v} \mapsto f(\mathbf{v})$ *is convex*
- $\forall \mathbf{v}, \forall$ *permutation* P, $f(\mathbf{v} \cdot P) = f(\mathbf{v})$

(We say that f *is Schur-convex)*

Then, $\mathbf{w} \prec \mathbf{v} \implies f(\mathbf{w}) \leq f(\mathbf{v})$

Proof

Lemma

If:

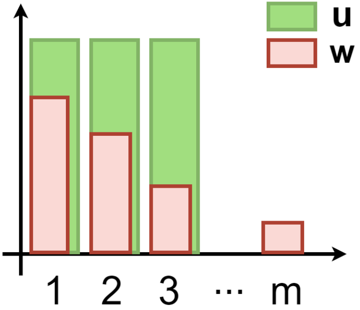
- $f : \mathbf{v} \mapsto f(\mathbf{v})$ *is convex*
- $\forall \mathbf{v}$, $\forall$ *permutation* P, $f(\mathbf{v} \cdot P) = f(\mathbf{v})$

(We say that f *is Schur-convex)*

Then, $\mathbf{w} \prec \mathbf{v} \implies f(\mathbf{w}) \leq f(\mathbf{v})$

Notice:

1. Random variable X is Schur-convex
2. Expectation function is convex
3. $\forall$ weight distribution $\mathbf{w}$, $\mathbf{w} \prec \mathbf{u}$



Proof

Lemma

If:

- $f : \mathbf{v} \mapsto f(\mathbf{v})$ *is convex*
- $\forall \mathbf{v}, \forall$ *permutation* P, $f(\mathbf{v} \cdot P) = f(\mathbf{v})$

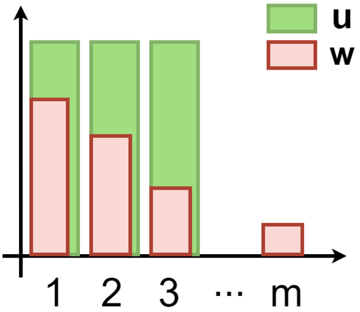
(We say that f is Schur-convex)

Then, $\mathbf{w} \prec \mathbf{v} \implies f(\mathbf{w}) \leq f(\mathbf{v})$

Notice:

1. Random variable X is Schur-convex
2. Expectation function is convex
3. $\forall$ weight distribution $\mathbf{w}$, $\mathbf{w} \prec \mathbf{u}$

Thus $\mathbb{E}(X(\mathbf{w})) \leq \mathbb{E}(X(\mathbf{u}))$



Proof

Lemma

If:

- $f : \mathbf{v} \mapsto f(\mathbf{v})$ *is convex*
- $\forall \mathbf{v}, \forall$ *permutation* P, $f(\mathbf{v} \cdot P) = f(\mathbf{v})$

(We say that f is Schur-convex)

Then, $\mathbf{w} \prec \mathbf{v} \implies f(\mathbf{w}) \leq f(\mathbf{v})$

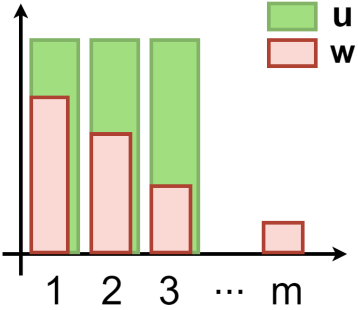
Notice:

1. Random variable X is Schur-convex
2. Expectation function is convex
3. $\forall$ weight distribution $\mathbf{w}$, $\mathbf{w} \prec \mathbf{u}$

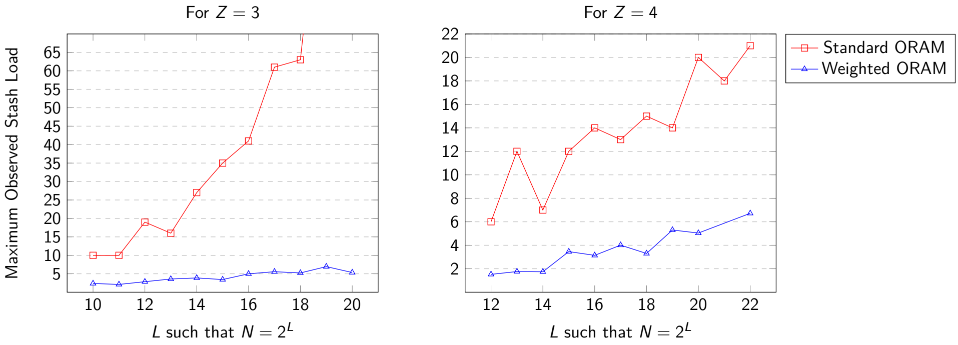
Thus $\mathbb{E}(X(\mathbf{w})) \leq \mathbb{E}(X(\mathbf{u}))$

$\mathbb{E}(X(\mathbf{u}))$ is negligible (cf Path-ORAM)

$\implies$ expected overflow negligible. □



Experimental results



Takaway

- Tree-ORAMs are powerful enough to naturally (no added cost) support items of variable sizes (variable in time too)
- Criterion to judge of an ORAM's ability to handle weighted objects.

Takaway

- Tree-ORAMs are powerful enough to naturally (no added cost) support items of variable sizes (variable in time too)
- Criterion to judge of an ORAM's ability to handle weighted objects.
- Any ORAM can handle them with small blowup ($O(log(N))$)
- Weighted ORAM can be used to build Searchable Symmetric Encryption

Takaway

- Tree-ORAMs are powerful enough to naturally (no added cost) support items of variable sizes (variable in time too)
- Criterion to judge of an ORAM's ability to handle weighted objects.
- Any ORAM can handle them with small blowup ($O(log(N))$)
- Weighted ORAM can be used to build Searchable Symmetric Encryption

Thank You!

ia.cr/2023/350