

Data-X Talk

Path ORAM: An Extremely Simple Oblivious RAM Protocol

by *Emil Stefanov, Elaine Shi, Marten van Dijk, Christopher Fletcher, Ling Ren, Xiangyao Yu and Srinivas Devadas*

[[Ste+13](#)]

Dmytro Bogatov

dmytro@bu.edu

Built from [946fdbb7](#) on December 17, 2017

Boston University

Table of Contents

Oblivious Memory

Overview of other ORAMs

Problem definition

Path ORAM protocol

Overview

Server storage

Client storage

The algorithm

Example

Recursion and
parametrization

Bounds on stash usage

Evaluation

Applications and extensions

Conclusion



OBLIVIOUS MEMORY

Dmytro Bogatov

BU Class of 2023

Problem statement

Untrusted server. Secure database — each record is encrypted.
What are we missing?

Dmytro Bogatov

BU Class of 2023



Problem statement

Untrusted server. Secure database — each record is encrypted.
What are we missing?

Security vulnerability

Adversary still sees the **access pattern** — an ordered sequence of read/write operations on data records.

Attack example

Compromising Privacy in Precise Query Protocols [DR13].

Observe range queries to recover order. Requires 10^4 queries to compromise privacy for database of over 10^6 records.



BU Class of 2023

Definition

A machine is *oblivious* if the sequence in which it accesses memory locations is equivalent for any two inputs with the same running time [[GO96](#)].

Dmytro Bogatov

BU Class of 2023



Theorems

Let $\text{RAM}(m)$ denote a **RAM** with m memory locations and access to a random oracle. Then t steps of an arbitrary $\text{RAM}(m)$ can be simulated by

- at most $\mathcal{O}(t \cdot (\log_2 t)^3)$ steps of an oblivious $\text{RAM}(m \cdot (\log_2 m)^2)$
- at least $\max\{m, (t - 1) \cdot \log_2 m\}$ steps of an oblivious $\text{RAM}(m)$

[G096]



OVERVIEW OF OTHER ORAMs

Dmytro Bogatov

BU Class of 2023

ORAMs Experimental Evaluation

ORAM	Computation	Communication	Server	Client
Basic-SR	$N \log N$	$N \log N$	N	1
IBS-SR	N	\sqrt{N}	N	\sqrt{N}
Basic-HR	$N \log^2 N$	$N \log^2 N$	$N \log N$	1^b
BB-ORAM	$\log^2 N$	$N \log^2 N$	$N \log N$	1
TP-ORAM	\sqrt{N}	1	N	$\sqrt{N} + \frac{N}{B}$
Path-ORAM	$\log N$	1	N	¹

Table 2 from [CXL16]. Worst-case scenarios shown.

¹ $\mathcal{O}(\log N) \cdot \omega(1) + \mathcal{O}\left(\frac{N}{B}\right)$



PROBLEM DEFINITION

Dmytro Bogatov
BU Class of 2023

- The client fetches/stores data in atomic units — *blocks* — of size B bytes each
- Let N be the working set — number of distinct data blocks stored in ORAM

Dmytro Bogatov

BU Class of 2023



We aim to provide an extremely simple ORAM construction in contrast with previous work

Can then be implemented in hardware [[Maa14](#)].

Dmytro Bogatov

BU Class of 2023



Formal security definition

Data request sequence of length M , where each op_i denotes a $\text{read}(a_i)$ or a $\text{write}(a_i, \text{data}_i)$ operation

$$\vec{y} := ((op_M, a_M, \text{data}_M), \dots, (op_1, a_1, \text{data}_1))$$

Let $A(\vec{y})$ denote the (possibly randomized) sequence of accesses to the remote storage given the sequence of data requests \vec{y} .



Formal security definition

An ORAM construction is said to be secure if

- for any two data request sequences \vec{y} and \vec{z} of the same length, their access patterns $A(\vec{y})$ and $A(\vec{z})$ are computationally indistinguishable by anyone but the client
- the ORAM construction is correct with probability $p \geq 1 - \text{negl}(|\vec{y}|)$.



PATH ORAM PROTOCOL

Dmytro Bogatov

BU Class of 2023

Diagram

Client

Position table

Block ID	Leaf ID
----------	---------

5	6
---	---

6	8
---	---

7	1
---	---

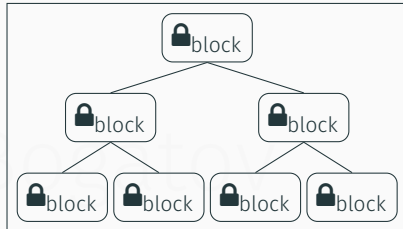
Stash

{3, 5, 7, 13, ...}

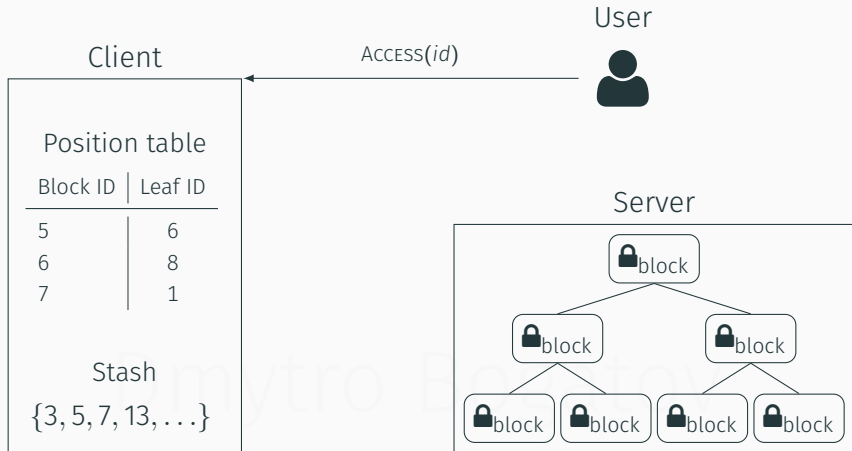
User



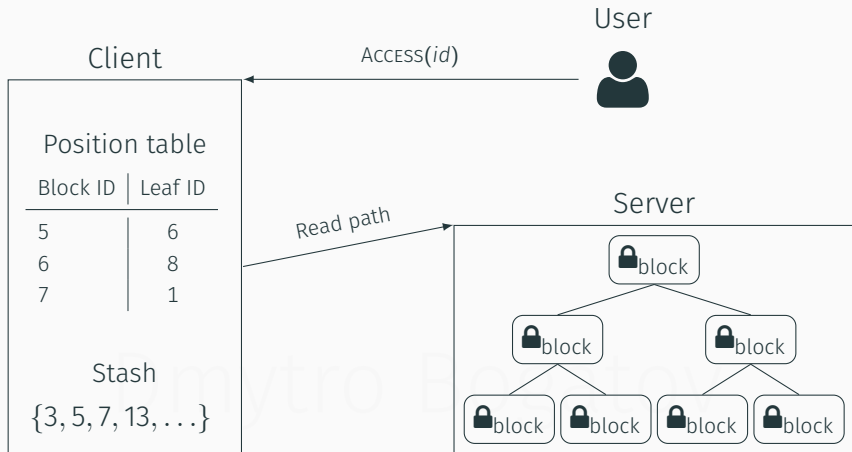
Server



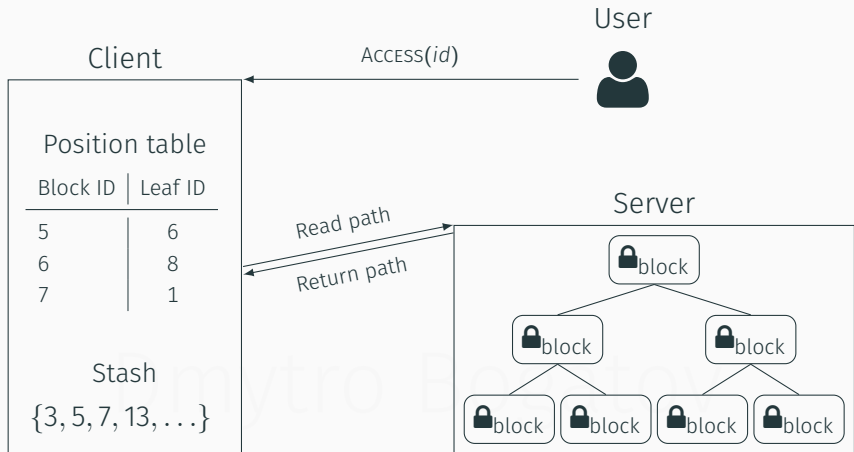
Diagram



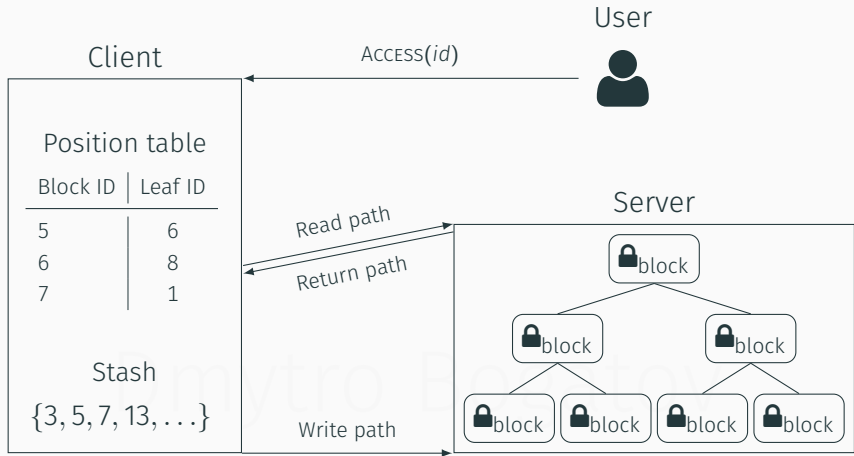
Diagram



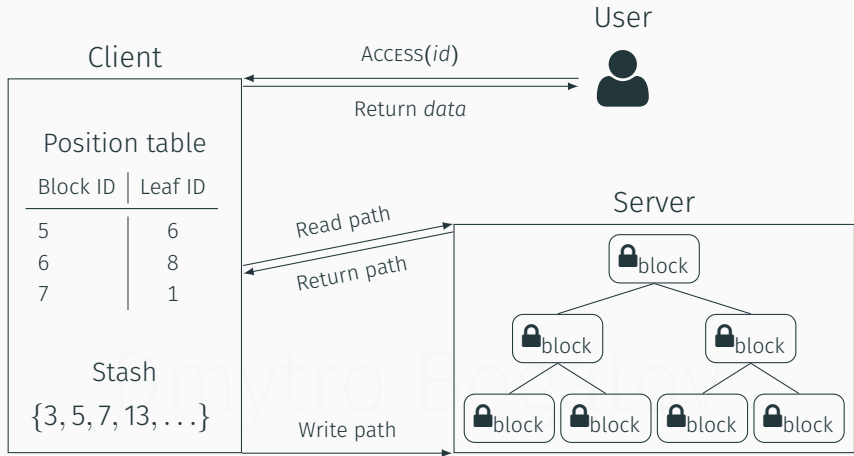
Diagram



Diagram



Diagram



Main invariant

The client stores a small amount of local data in a **stash**. The server-side storage is treated as a **binary tree** where each node is a **bucket** that holds an exact fixed number of **blocks**.

Invariant

At any time, each block is mapped to a uniformly random leaf bucket in the tree, and unstashed blocks are always placed in some bucket along the path to the mapped leaf.



Binary tree

The server stores a binary tree data structure of height L and 2^L leaves. We then need $L = \lceil \log_2 N \rceil$ levels. The levels of the tree are numbered 0 to L where level 0 denotes the root of the tree and level L denotes the leaves.

Dmytro Bogatov

BU Class of 2023



Bucket

Each node in the tree is called **bucket**. Each bucket can contain up to Z real blocks. If a bucket has fewer than Z real blocks, it is padded with dummy blocks to always be of size Z .

Dmytro Bogatov

BU Class of 2023



Path

Let $x \in \{0, 1, \dots, 2^L - 1\}$ denote the x_{th} leaf node in the tree. Any leaf node x defines a unique path from the root of the tree to the leaf x . We use $\mathcal{P}(x)$ to denote a set of buckets along the path from leaf x to the root. Additionally, $\mathcal{P}(x, l)$ denotes the bucket in $\mathcal{P}(x)$ at level l in the tree.



Server storage size

Total server storage used is about $Z \cdot N$ blocks. Since Z is a small constant, server storage is $\mathcal{O}(N)$.

Dmytro Bogatov

BU Class of 2023



Stash

The client locally stores overflowing blocks in a local data structure S called **stash**. The stash has a worst-case size of $\mathcal{O}(\log N) \cdot \omega(1)$ blocks with high probability. The stash is usually empty after each ORAM read/write operation completes.

Dmytro Bogatov

BU Class of 2023



Position map

The client stores a position map, such that $x := \text{position}[a]$ means that block a is currently mapped to the x_{th} leaf node — this means that block a resides in some bucket in path $\mathcal{P}(x)$, or in the stash. The position map changes over time as blocks are accessed and remapped.

Dmytro Bogatov

BU Class of 2023



Bandwidth

For each load or store operation, the client reads a path of $Z \log N$ blocks from the server and then writes it back, resulting in a total of $2Z \log N$ blocks bandwidth used per access. Since Z is a constant, the bandwidth usage is $\mathcal{O}(\log N)$ blocks.

Dmytro Bogatov

BU Class of 2023



Client storage size

The position map is of size $NL = N \log N$ bits, which is of size $\mathcal{O}(N)$ blocks when the block size $\Omega(\log N)$ bits.

The stash is at most $\mathcal{O}(\log N) \cdot \omega(1)$ blocks to obtain negligible failure probability. The recursive construction can achieve client storage of $\mathcal{O}(\log N) \cdot \omega(1)$.

Dmytro Bogatov

BU Class of 2023



$\text{ACCESS}(op, a, data^*)$

- 1 $x \leftarrow \text{position}[a]$
- 2 $\text{position}[a] \leftarrow \text{UNIFORMRANDOM}(0 \dots 2^L - 1)$

Dmytro Bogatov

BU Class of 2023



$\text{ACCESS}(op, a, data^*)$

```
3 | for  $\ell \in \{0, 1, \dots, L\}$  do  
4 |    $S \leftarrow S \cup \text{READBUCKET}(\mathcal{P}(x, \ell))$   
5 | end for
```

Dmytro Bogatov

BU Class of 2023



Update block

$\text{ACCESS}(op, a, data^*)$

```
6 | data  $\leftarrow$  Read block  $a$  from  $S$ 
7 | if  $op = \text{write}$  then
8 |    $S \leftarrow (S - \{(a, data)\}) \cup \{(a, data^*)\}$ 
9 | end if
```

Dmytro Bogatov

BU Class of 2023



Write path back

ACCESS($op, a, data^*$)

```
10 for  $\ell \in \{L, L-1, \dots, 0\}$  do  
11    $S' \leftarrow \{(a', data') \in S : \mathcal{P}(x, \ell) = \mathcal{P}(\text{position}[a'], \ell)\}$   
12    $S' \leftarrow \text{Select } \min(|S'|, Z) \text{ blocks from } S'$   
13    $S \leftarrow S - S'$   
14   WRITEBUCKET( $\mathcal{P}(x, \ell), S'$ )  
15 end for
```



$\text{ACCESS}(op, a, data^*)$

16 | **return** data

Dmytro Bogatov

BU Class of 2023



READBUCKET(*bucket*)

The client reads all Z blocks (including any dummy blocks) from the *bucket* stored on the server. Blocks are decrypted as they are read.

WRITEBUCKET(*bucket, blocks*)

The client writes the blocks *blocks* into the specified *bucket* on the server. When writing, the client *pads* blocks with dummy blocks to make it of size Z — note that this is important for security. All blocks (including dummy blocks) are re-encrypted, using a randomized encryption scheme, as they are written.



Computation

Client's computation is $\mathcal{O}(\log N) \cdot \omega(1)$ per data access. We treat the server as a network storage device, so it only needs to do the computation necessary to retrieve and store $\mathcal{O}(\log N)$ blocks per data access.

Dmytro Bogatov

BU Class of 2023



Full algorithm

ACCESS($op, a, data^*$)

```
1   $x \leftarrow \text{position}[a]$ 
2   $\text{position}[a] \leftarrow \text{UNIFORMRANDOM}(0 \dots 2^L - 1)$ 
3  for  $\ell \in \{0, 1, \dots, L\}$  do
4       $S \leftarrow S \cup \text{READBUCKET}(\mathcal{P}(x, \ell))$ 
5  end for
6   $\text{data} \leftarrow \text{Read block } a \text{ from } S$ 
7  if  $op = \text{write}$  then
8       $S \leftarrow (S - \{(a, \text{data})\}) \cup \{(a, \text{data}^*)\}$ 
9  end if
10 for  $\ell \in \{L, L - 1, \dots, 0\}$  do
11      $S' \leftarrow \{(a', \text{data}') \in S : \mathcal{P}(x, \ell) = \mathcal{P}(\text{position}[a'], \ell)\}$ 
12      $S' \leftarrow \text{Select } \min(|S'|, Z) \text{ blocks from } S'$ 
13      $S \leftarrow S - S'$ 
14      $\text{WRITEBUCKET}(\mathcal{P}(x, \ell), S')$ 
15 end for
16 return data
```



$$\mathbf{p} = (\text{position}_M[a_M], \text{position}_{M-1}[a_{M-1}], \dots, \text{position}_1[a_1])$$

$$\Pr[\mathbf{p}] = \left(\frac{1}{2^L}\right)^M$$

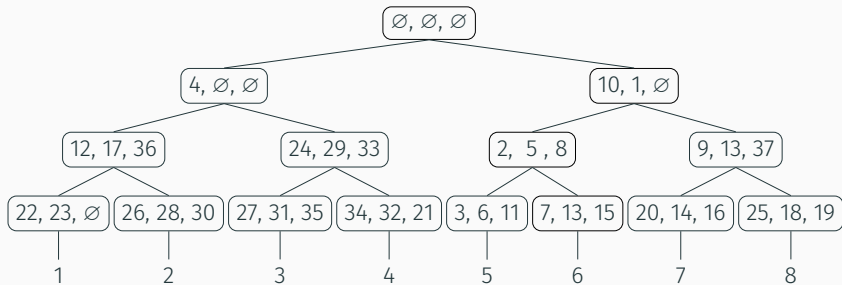
- The sequence of encrypted paths is computationally indistinguishable from a random sequence of bit strings
- $A(\vec{y})$ is computationally indistinguishable from a random sequence of bit strings



EXAMPLE

Dmytro Bogatov
BU Class of 2023

Initial state



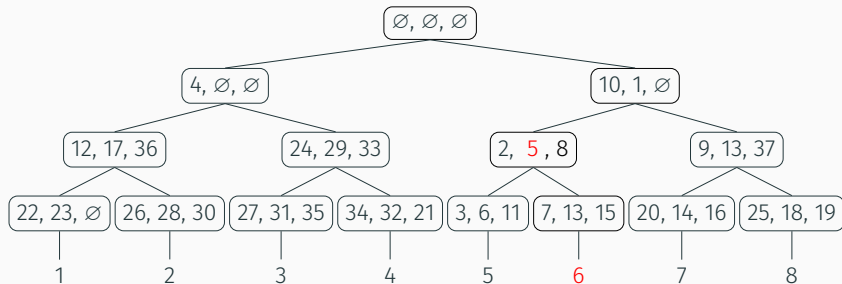
Initial underlying tree

Block ID	1	2	3	4	5	6	...	37
Leaf ID	6	5	5	1	6	5	...	7

Initial position map



ACCESS(5)



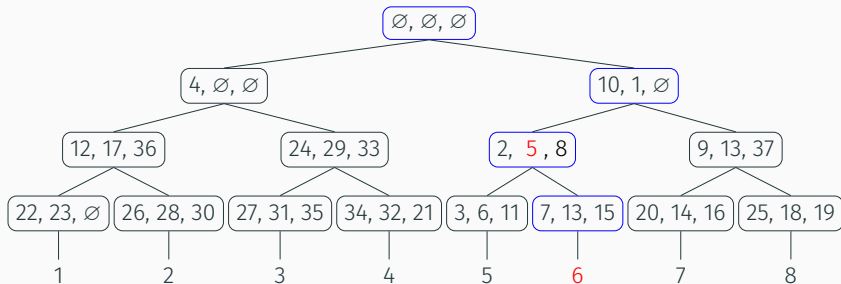
Select block with ID 5

Block ID	1	2	3	4	5	6	...	37
Leaf ID	6	5	5	1	6	5	...	7

Find leaf for block with ID 5



Read path



Select path $\mathcal{P}(6)$

Block ID	1	2	3	4	5	6	...	37
Leaf ID	6	5	5	1	6	5	...	7

Find leaf for block with ID 5



Remap block

Stash

$S = \{10, 1, 2, 5, 8, 7, 13, 15\}$

$data = \text{DECRYPTBLOCK}(5)$

$5' \leftarrow \text{ENCRYPTDATA}(data^*)$

$S = \{10, 1, 2, 5', 8, 7, 13, 15\}$

Re-encrypt all blocks in the stash.

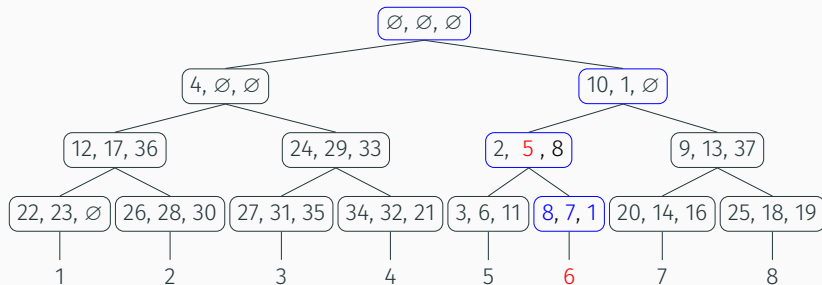
Remap

Assign random leaf

$\text{position}[5] := 3$



Write path back



Write bucket on level $L = 3$

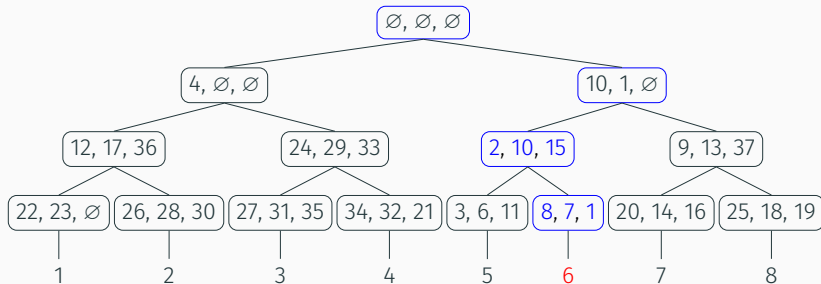
Stash

$$S' = \{8, 7, 1\}$$

$$S = \{10, 2, 5, 13, 15\}$$



Write path back



Write bucket on level $L - 1 = 2$

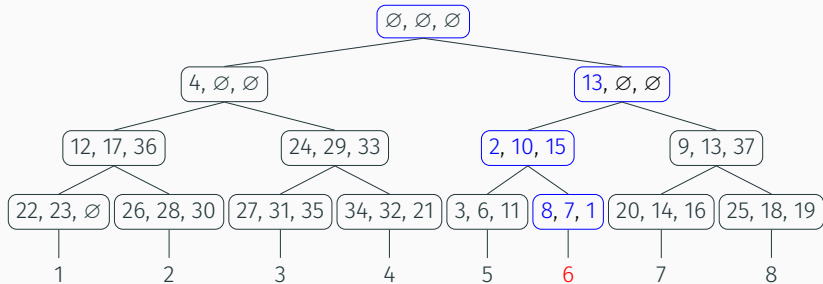
Stash

$$S' = \{2, 10, 15\}$$

$$S = \{5, 13\}$$



Write path back



Write bucket on level $L - 2 = 1$

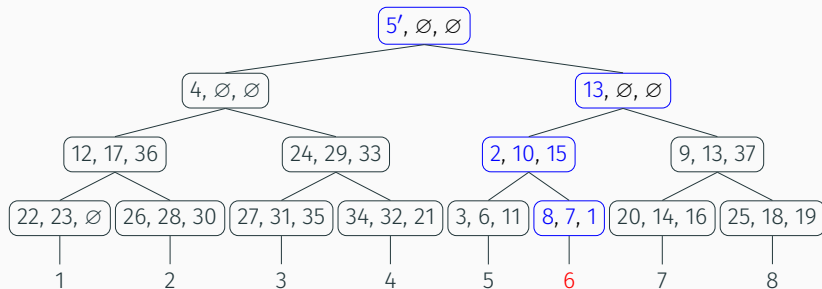
Stash

$$S' = \{13\}$$

$$S = \{5\}$$



Write path back



Write bucket on level $L - 3 = 0$ — root

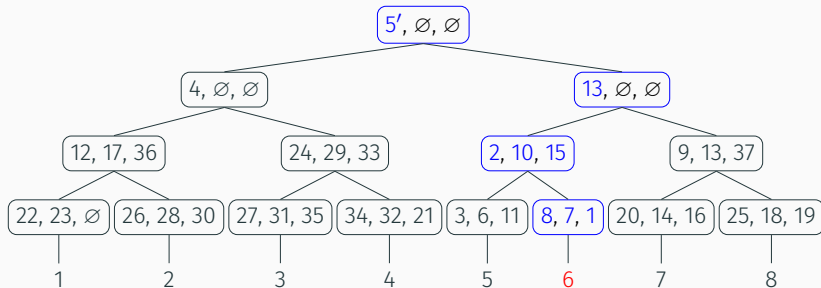
Stash

$$S' = \{5\}$$

$$S = \emptyset$$



Final state



Tree after the execution

Block ID	1	2	3	4	5	6	...	37
Leaf ID	6	5	5	1	3	5	...	7

Position map after the execution



RECURSION AND PARAMETRIZATION

Dmytro Bogatov

BU Class of 2023

In our non-recursive scheme described in the previous section, the client must store a relatively large position map.

The idea is simple: instead of storing the position map on the client side, we store the position map on the server side in a smaller ORAM, and recurse [[Shi+11](#); [SSS11](#)].

Dmytro Bogatov

BU Class of 2023



The client storage for the recursive PathORAM construction (with non-uniform block sizes) can be reduced from $\mathcal{O}(\log^2 N) \cdot \omega(1)$ to $\mathcal{O}(\log N) \cdot \omega(1)$ by having a *single stash* shared among all levels of the recursion. This is possible while still maintaining a negligible probability of stash overflow.

Dmytro Bogatov

BU Class of 2023



BOUNDS ON STASH USAGE

Dmytro Bogatov
BU Class of 2023

Probability of failure is negligible

Main theorem

Let \mathbf{a} be any sequence of block addresses with a working set of size at most N . For a bucket size $Z = 5$, tree height $L = \lceil \log N \rceil$ and stash size R , the probability of a PathORAM failure after a sequence of load/store operations corresponding to \mathbf{a} , is at most

$$\Pr [\text{st}(\text{ORAM}_L^5[\mathbf{s}]) > R \mid a(\mathbf{s}) = \mathbf{a}] \leq 14 \cdot (0.6002)^R$$

where the probability is over the randomness that determines \mathbf{x} and \mathbf{y} in $\mathbf{s} = (\mathbf{a}, \mathbf{x}, \mathbf{y})$.



BU Class of 2023

Proof outline

1. Introduce a second ORAM, called ∞ -ORAM
2. Characterize the distributions of real blocks over buckets in a ∞ -ORAM for which post-processing leads to a stash usage $> R$
3. Analyze the usage of subtrees T (introduced in the proof)



EVALUATION

Dmytro Bogatov

BU Class of 2023

Experiment setup

- PathORAM uses a binary tree with height $L = \lceil \log_2 N \rceil - 1$. It is empirically proven to be sufficient and efficient.
- *Stash occupancy* is the number of overflowing blocks. Represents client's *persistent* local storage.
- First load N blocks into ORAM and then access each block in a round-robin pattern. Worst-case access pattern in terms of stash occupancy.
- Single run for about 250 billion accesses after doing 1 billion accesses for warming-up.



Results

- Required stash size grows linearly with the security parameter. Failure probability decreases exponentially with the stash size. See **figure** in appendix.
- Extrapolate those results for realistic values of λ . See **table** in appendix.
- It is by definition infeasible to simulate for practically adopted security parameters (e.g., $\lambda = 128$)
- Required stash size for a low failure probability does not depend on N . This shows PathORAM has good scalability. See **figure** in appendix.



BU Class of 2023

Results

- For $Z \geq 5$, the usage of a subtree is close to the number of buckets in it. Also holds for $Z = 4$.
- For the levels close to the root, the bucket load is indeed 1 block.
- Leaves have slightly heavier loads as blocks accumulate at the leaves of the tree.
- $Z = 3$, however, exhibits a different distribution of bucket load and produces much larger stash sizes in practice.

BU Class of 2023



APPLICATIONS AND EXTENSIONS

Dmytro Bogatov
BU Class of 2023

Oblivious Binary Search Tree

PathORAM can be used to perform search on an oblivious binary search tree, using $\mathcal{O}(\log^2 N)$ bandwidth. [[Gen+13](#)]

Dmytro Bogatov

BU Class of 2023



In order to avoid complicated (and possibly expensive) oblivious state synchronization between the clients, Goodrich *et al.* introduce the concept of stateless ORAM [Goo+11] where the client state is small enough so that any client accessing the ORAM can download it before each data access and upload it afterwards.

Dmytro Bogatov

BU Class of 2023



PathORAM is particularly amenable to hardware design because of its simplicity and low on-chip storage requirements.

Maas [[Maa14](#)] built a hardware implementation of a PathORAM based secure processor using FPGAs and the Convey platform.

Fletcher, Dijk, and Devadas [[FDD12](#)] and Fletcher [[Fle13](#)] and Ren *et al.* [[Ren+13](#)] built a simulator for a secure processor based on PathORAM.



The protocol can be easily extended to provide integrity (with freshness) for every access to the untrusted server storage.

We can achieve integrity by simply treating the PathORAM tree as a *Merkle tree* where data is stored in all nodes of the tree.

$$H(b_1 || b_2 || \dots || b_z || h_1 || h_2)$$

Dmytro Bogatov

BU Class of 2023



CONCLUSION

Dmytro Bogatov
BU Class of 2023

Conclusion

Partly due to its simplicity, PathORAM is the most practical ORAM scheme known-to-date under a small amount of client storage. We formally prove asymptotic bounds on PathORAM, and show that its performance is competitive with or asymptotically better than the best known construction (for a small amount of client storage), assuming reasonably large block sizes. We also present simulation results that confirm our theoretic bounds. [Ste+13]



Data-X Talk

Path ORAM: An Extremely Simple Oblivious RAM Protocol

by *Emil Stefanov, Elaine Shi, Marten van Dijk, Christopher Fletcher, Ling Ren, Xiangyao Yu and Srinivas Devadas*

[[Ste+13](#)]

Dmytro Bogatov

dmytro@bu.edu

Built from [946fdbb7](#) on December 17, 2017

Boston University

REFERENCES



Oded Goldreich and Rafail Ostrovsky. “Software Protection and Simulation on Oblivious RAMs”. In: *J. ACM* 43.3 (May 1996), pp. 431–473. ISSN: 0004-5411. DOI: [10.1145/233551.233553](https://doi.org/10.1145/233551.233553). URL: <http://doi.acm.org/10.1145/233551.233553>.



Michael T. Goodrich *et al.* “Privacy-Preserving Group Data Access via Stateless Oblivious RAM Simulation”. In: *CoRR* abs/1105.4125 (2011). arXiv: [1105.4125](https://arxiv.org/abs/1105.4125). URL: <http://arxiv.org/abs/1105.4125>.

BU Class of 2023





Elaine Shi et al. “Oblivious RAM with $O(\log^3 N)$ Worst-Case Cost”. In: *Advances in Cryptology – ASIACRYPT 2011: 17th International Conference on the Theory and Application of Cryptology and Information Security, Seoul, South Korea, December 4-8, 2011. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 197–214. ISBN: 978-3-642-25385-0. DOI: [10.1007/978-3-642-25385-0_11](https://doi.org/10.1007/978-3-642-25385-0_11). URL: https://doi.org/10.1007/978-3-642-25385-0_11.



References III



Emil Stefanov, Elaine Shi, and Dawn Song. “Towards Practical Oblivious RAM”. In: *CoRR* abs/1106.3652 (2011), pp. 1–40. arXiv: [1106.3652](https://arxiv.org/abs/1106.3652). URL: <http://arxiv.org/abs/1106.3652>.



Christopher W. Fletcher, Marten van Dijk, and Srinivas Devadas. “A Secure Processor Architecture for Encrypted Computation on Untrusted Programs”. In: *Proceedings of the Seventh ACM Workshop on Scalable Trusted Computing*. STC '12. Raleigh, North Carolina, USA: ACM, 2012, pp. 3–8. ISBN: 978-1-4503-1662-0. DOI: [10.1145/2382536.2382540](https://doi.acm.org/10.1145/2382536.2382540). URL: <http://doi.acm.org/10.1145/2382536.2382540>.





Jonathan L. Dautrich Jr. and Chinya V. Ravishankar.
“Compromising Privacy in Precise Query Protocols”.
In: *Proceedings of the 16th International Conference
on Extending Database Technology*. EDBT '13. Genoa,
Italy: ACM, 2013, pp. 155–166. ISBN: 978-1-4503-1597-5.
DOI: [10.1145/2452376.2452397](https://doi.org/10.1145/2452376.2452397). URL: <http://doi.acm.org/10.1145/2452376.2452397>.



Christopher Wardlaw Fletcher. “Ascend: An
architecture for performing secure computation on
encrypted data”. PhD thesis. 2013.





Craig Gentry *et al.* “Optimizing ORAM and Using It Efficiently for Secure Computation”. In: *Privacy Enhancing Technologies: 13th International Symposium, PETS 2013, Bloomington, IN, USA, July 10-12, 2013. Proceedings*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 1–18. ISBN:

978-3-642-39077-7. DOI:

[10.1007/978-3-642-39077-7_1](https://doi.org/10.1007/978-3-642-39077-7_1). URL: https://doi.org/10.1007/978-3-642-39077-7_1.



References VI



Ling Ren et al. “Design space exploration and optimization of path oblivious ram in secure processors”. In: *ACM SIGARCH Computer Architecture News* 41.3 (2013), pp. 571–582.



Emil Stefanov et al. “Path ORAM: An Extremely Simple Oblivious RAM Protocol”. In: *Proceedings of the 2013 ACM SIGSAC Conference on Computer Communications Security*. CCS '13. Berlin, Germany: ACM, 2013, pp. 299–310. ISBN: 978-1-4503-2477-9. DOI: [10.1145/2508859.2516660](https://doi.org/10.1145/2508859.2516660). URL: <http://doi.acm.org/10.1145/2508859.2516660>.



BU Class of 2023

References VII



Martin Maas. “PHANTOM: Practical Oblivious Computation in a Secure Processor”. MA thesis. EECS Department, University of California, Berkeley, May 2014, pp. 1–87. URL: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2014/EECS-2014-89.html>.



Zhao Chang, Dong Xie, and Feifei Li. “Oblivious RAM: A Dissection and Experimental Evaluation”. In: *Proc. VLDB Endow.* 9.12 (Aug. 2016), pp. 1113–1124. ISSN: 2150-8097. DOI: [10.14778/2994509.2994528](https://doi.org/10.14778/2994509.2994528). URL: <http://dx.doi.org/10.14778/2994509.2994528>.



Max stash size grows linearly with λ

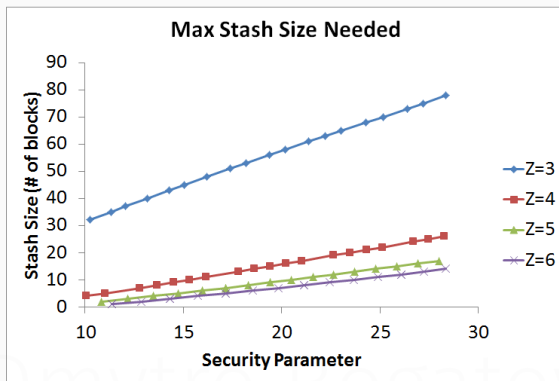


Figure 3 from [Ste+13].



Max stash size for large security parameters

Security parameter λ	Bucket size Z		
	4	5	6
	Max stash size		
80	89	63	53
128	147	105	89
256	303	218	186

Figure 5 from [Ste+13].



Max stash size does not depend on N

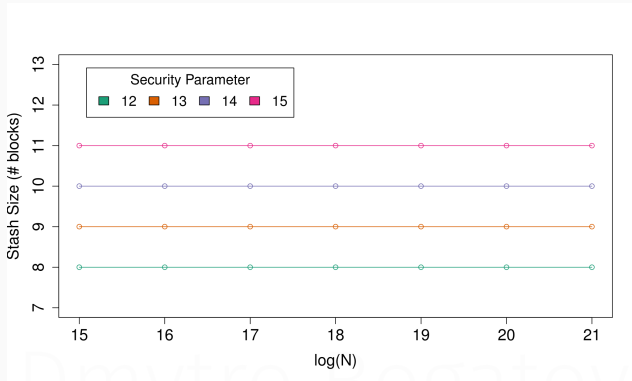


Figure 4 from [Ste+13].

