

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

2017-12-17

Data-X Talk

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	The algorithm
Overview of other ORAMs	Example
Problem definition	Recursion and parametrization
Path ORAM protocol	Bounds on stash usage
Overview	Evaluation
Server storage	Applications and extensions
Client storage	Conclusion

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

- Start with the problem statement. What an ORAM is and why we need it.
- Talk about what ORAMs are already there and what their disadvantages are.
- Then go in the same order as paper goes.



2017-12-17

Data-X Talk
└ Oblivious Memory

OBLIVIOUS MEMORY

OBLIVIOUS MEMORY

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

Problem statement

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

Data-X Talk
└ Oblivious Memory

2017-12-17

└ Problem statement

Problem statement

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

The idea is to build a secure cloud, and more specifically — secure database. The server is untrusted — we assume an adversary can read every byte on the disk and track all CPU operations, but cannot interfere. This threat model is called *honest-but-curious* adversary. The first step is encrypting the database, so that only client can decrypt. But that is just the first step — what are we missing?

An adversary can see the access pattern. It can see which records are accessed more often. Which records are accessed only after some other records were touched. How often read vs write operations occur.

Talk about Dautrich and Ravishankar [DR13] paper.



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.

Data-X Talk
└ Oblivious Memory

└ Problem statement

The idea is to build a secure cloud, and more specifically — secure database. The server is untrusted — we assume an adversary can read every byte on the disk and track all CPU operations, but cannot interfere. This threat model is called *honest-but-curious* adversary. The first step is encrypting the database, so that only client can decrypt. But that is just the first step — what are we missing?

An adversary can see the access pattern. It can see which records are accessed more often. Which records are accessed only after some other records were touched. How often read vs write operations occur.

Talk about Dautrich and Ravishankar [DR13] paper.

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.

2017-12-17



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].

Data-X Talk
└ Oblivious Memory

└ Oblivious RAM

2017-12-17

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].

A solution is to design an oblivious memory access system. This definition of oblivious machine is cited from the original paper on ORAMs by Goldreich and Ostrovsky [GO96] from May 1996 — around my birthday. Among the other things the paper states a number of theorems on computational bounds of generic ORAMs.

Although the paper analyzes generic ORAMs, 20 years ago people were more concerned about CPU working with RAM access patterns. The cloud did not really exist at that time.

So the one purpose of ORAM is to hide the access pattern. We will come to more formal security definition in a couple of slides.



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 $\text{RAM}(m)$

[GO96]



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 $\text{RAM}(m)$

[GO96]

These are some theorems stated and proved in the paper. I am not going to do proofs here. The idea is that these are the theoretical bounds for generic ORAMs. Designing our own ORAM the aim is to come as close as possible to lower bounds.

In fact, PathORAM does hit the lower bound — $\log m$ steps per each access.

OVERVIEW OF OTHER ORAMs

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

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

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

Chang, Xie, and Li [CXL16] published a great paper a year ago doing accurate comparison of known ORAM systems. They analyzed space and time complexity of the systems. The result is on the table.

Communication overhead measures the number of rounds of communication for one access. Server and client respectively show how much space is used by an ORAM on the server and on the client. Computational overhead is a composite of communication, encryption/decryption and client running overheads *per access*.

I am not going to elaborate on all ORAMs, but it is clear that one of them wins in almost every category. This is why we have chosen it for our secure cloud.



2017-12-17

Data-X Talk
└ Problem definition

PROBLEM DEFINITION

PROBLEM DEFINITION

Dmytro Bogatov
BU Class of 2023

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

Let us start with some notation. Assume that all data comes in atomic units called blocks and assume that we have N distinct blocks in ORAM.



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

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

Data-X Talk
└ Problem definition

└ Simplicity

2017-12-17

Simplicity

We aim to provide an extremely simple ORAM construction in contrast with previous work
Can then be implemented in hardware [[Maa14](#)].

One of the goals is simplicity. It not only allows us to analyze system easily. In case of PathORAM, it is so simple that it has been implemented in pure hardware.

One of the works that puts PathORAM on the silicon is written by Maas [[Maa14](#)].



Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

Security definitions

Formal security definition

Data request sequence of length M , where each op_i denotes a $read(a_i)$ or a $write(a_i, data_i)$ operation

$$\vec{y} := ((op_M, a_M, data_M), \dots, (op_1, a_1, data_1))$$

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

Data-X Talk
└ Problem definition

└ Security definitions

Formal security definition
Data request sequence of length M , where each op_i denotes a $read(a_i)$ or a $write(a_i, data_i)$ operation

$$\vec{y} := ((op_M, a_M, data_M), \dots, (op_1, a_1, data_1))$$

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

Let us define what we mean by access pattern. We mean a sequence of operations — writes and reads — on some blocks with identifiers a_i reading or writing some $data_i$. Say we have M operations in sequence.



Security definitions

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 $p \geq 1 - \text{negl}(|\vec{y}|)$.

Data-X Talk
└ Problem definition

└ Security definitions

2017-12-17

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 $p \geq 1 - \text{negl}(|\vec{y}|)$.

The most important one. We call a construction secure if an adversary cannot tell, which sequence of operations really went through the construction. Indistinguishability is a strong requirement — it says that we leak absolutely no information about our patterns. And indeed, we need to correctly respond to client requests. Since we allow randomized algorithms, we set another requirement on error probability. It has to be negligibly small.



2017-12-17

Data-X Talk
└ Path ORAM protocol

PATH ORAM PROTOCOL

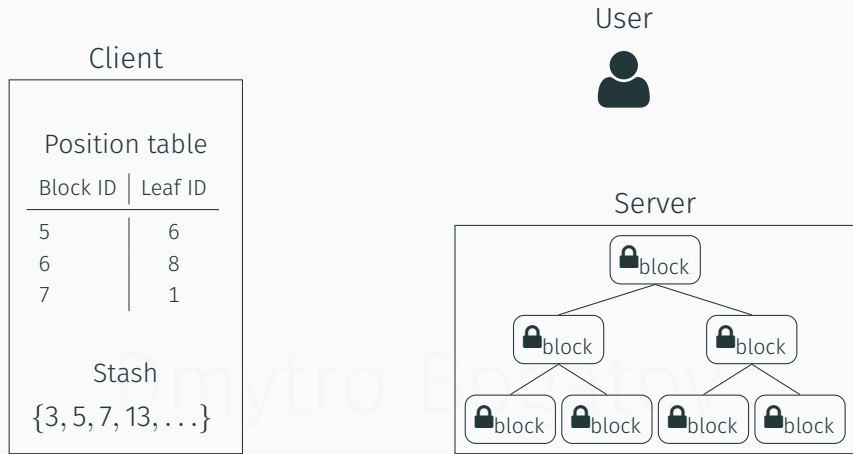
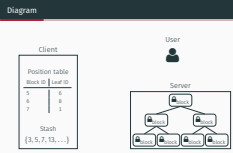
PATH ORAM PROTOCOL

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

2017-12-17

- Data-X Talk
 - Path ORAM protocol
 - Overview
 - Diagram



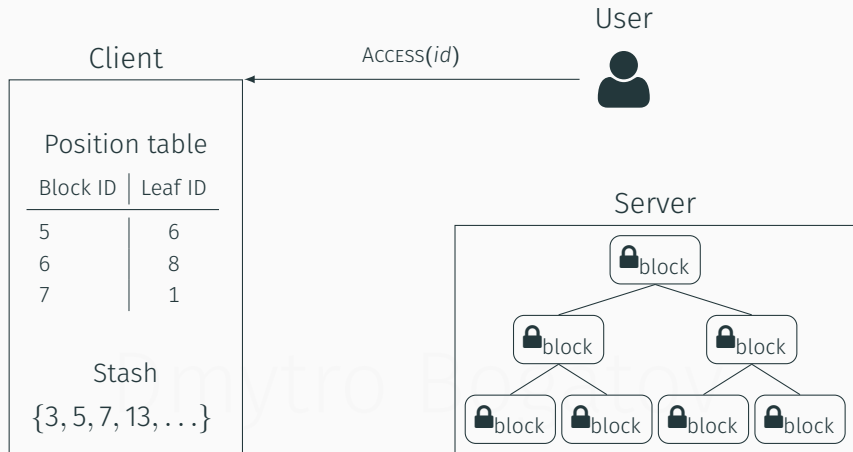
Here is the diagram of PathORAM protocol. There are two components in the ORAM model and a user interacting with the system.

Server basically holds a binary tree of buckets with encrypted blocks.

Client holds two data structures — position map and stash — we will talk about them in a moment.



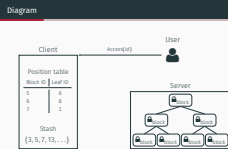
BU Class of 2023



Data-X Talk

- Path ORAM protocol
- Overview
- Diagram

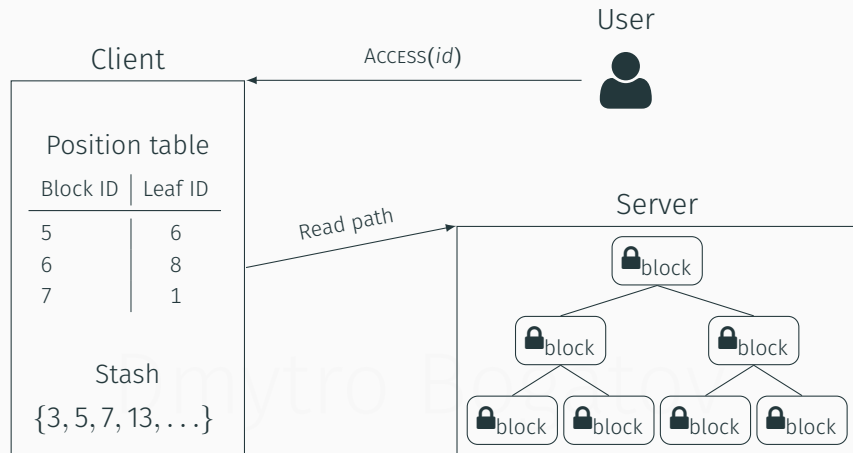
2017-12-17



First, user makes a request to the (trusted) client part. User requests — to read or to write — a block with certain identifier.

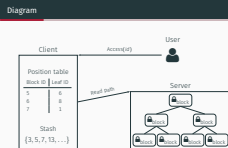


Diagram



Data-X Talk
└ Path ORAM protocol
└ Overview
└ Diagram

2017-12-17

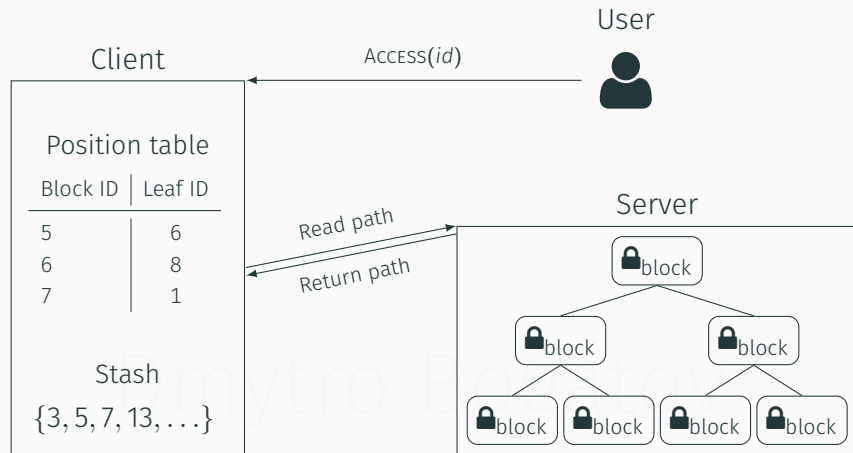


If client does not have the block in stash, it makes a request to the server and reads a path with encrypted blocks.

Server responds with a path of buckets of encrypted blocks.



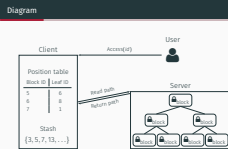
Diagram



Data-X Talk

- Path ORAM protocol
- Overview
- Diagram

2017-12-17

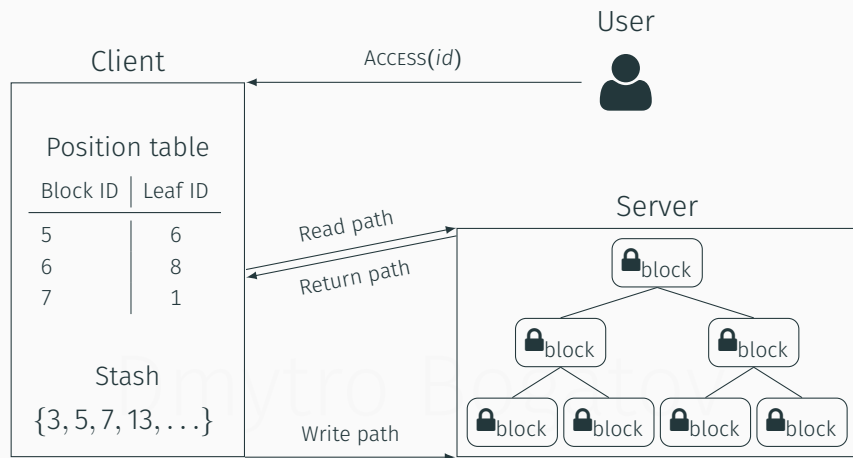


If client does not have the block in stash, it makes a request to the server and reads a path with encrypted blocks.

Server responds with a path of buckets of encrypted blocks.

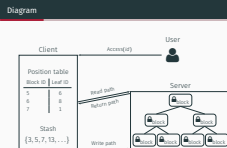


Diagram



2017-12-17

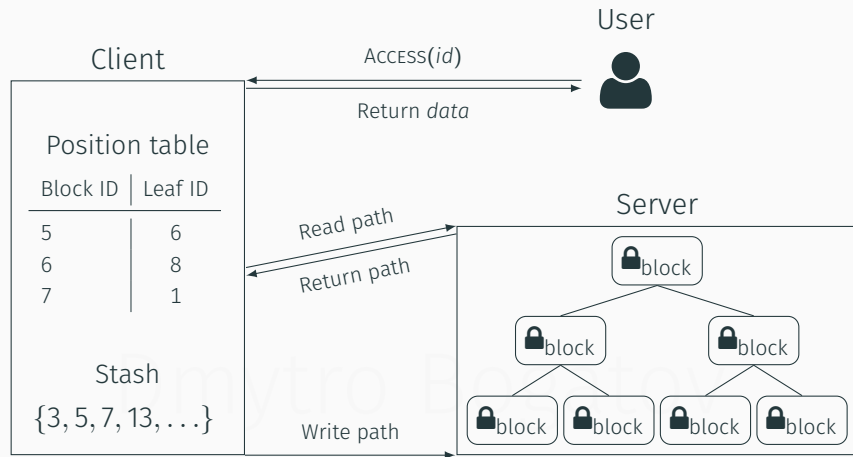
- Data-X Talk
 - Path ORAM protocol
 - Overview
 - Diagram



Client manipulates these blocks — re-encrypts them, shuffles them and write the path back. We will talk about how client writes path back in a randomized way.



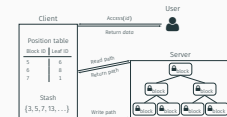
Diagram



Data-X Talk

- Path ORAM protocol
- Overview
- Diagram

2017-12-17



Finally, a client returns data to the user.



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**.

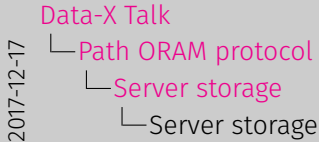
Invariant

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.

An underlying data structure for PathORAM is a binary tree. Client has a position table where each data block ID is mapped to the leaf node. Each time access occurs, whole path from the leaf to the root gets read and written. This ensures indistinguishability.

The goal of the invariant is to keep position map accurate at all times.





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.

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.

Let us define L — the height of our tree. Then reasonably it will be equal to $\log_2 N$ rounded up. Let us also define 0 level as root and L_{th} level as leaves.



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 .

2017-12-17

- Data-X Talk
 - Path ORAM protocol
 - Server storage
 - Server storage

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 .

Each node in a tree is a bucket that contains Z blocks. For the sake of indistinguishability, we pad bucket with dummy encrypted blocks. Choice of Z is a parameter. Experimental results show that small constant, eq. 4, will suffice.



Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

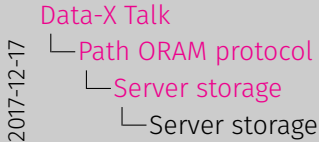
Path

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.

Let us define a path from leaf x to the root as $\mathcal{P}(x)$. Let us also define $\mathcal{P}(x, l)$ as the bucket along the path at level l .





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

Server storage size

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

Let us make an important observation. The total server storage used is the order of N since Z is a small constant.

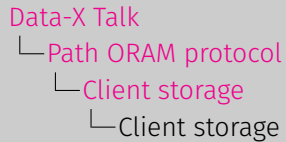


Dmytro Bogatov
BU Class of 2023

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.



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.

The core component of the client is stash. It is a local data structure that stores overflowing blocks. PathORAM protocol is randomized, and the error is the event that this stash overflows. The analysis, though, shows that for $\log N$ size of the stash this happens with negligible probability.



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.

2017-12-17

- Data-X Talk
 - Path ORAM protocol
 - Client storage
 - Client storage

Position map

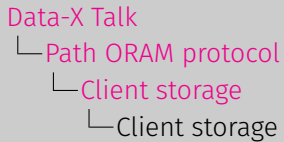
The client stores a position map, such that $x := \text{position}[a]$ means that block a is currently mapped to the x_{th} leaf node — this means that block a resides in some bucket in path $\mathcal{P}(x)$, or in the stash. The position map changes over time as blocks are accessed and remapped.

Another core structure of the client is the position table. It is a simple lookup table that maps the block with identifier a to some leaf x . The bucket does not necessarily live in the leaf, but it is guaranteed to live somewhere along the path or in the stash. The key to security is the re-randomization of this table on each access.



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.

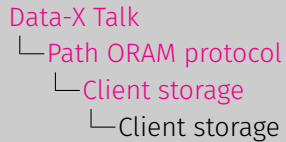


Bandwidth

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.

Each read and write, client reads the whole path and writes it back. A path is $Z \log N$ blocks, and since Z is a small constant, resulting usage is $\mathcal{O}(\log N)$ in the number of blocks.





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 $\mathcal{O}(\log N) \cdot \omega(1)$.

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 $\mathcal{O}(\log N) \cdot \omega(1)$.

Position map is the order of N and the stash is the order of $\log N$ for negligible error probability. So, for the basic PathORAM the client storage usage is order of N . However, it is possible to use recursive version of the ORAM, which I will talk about later, to lower the usage to the order of $\log N$.



ACCESS(op, a, data)*

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

The client stash S is initially empty. The server buckets are initialized to contain randomized encryptions of the dummy blocks. The client's position map is filled with independent random numbers between 0 and $2^L - 1$.

When the access occurs, the requested block is remapped.



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
```

Read the path $\mathcal{P}(x)$ containing block a .




```
ACCESS(op, a, data*)
6 data ← Read block a from S
7 if op = write then
8   S ← (S - {(a, data)}) ∪ {(a, data*)}
9 end if
```

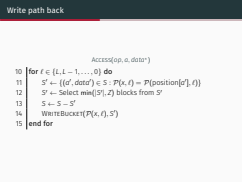
ACCESS($op, a, data^*$)

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

If the access is a **write**, update the data stored for block a .



- └ Path ORAM protocol
 - └ The algorithm
 - └ 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    $\text{WRITEBUCKET}(\mathcal{P}(x, \ell), S')$ 
15 end for
```

Write the path back and possibly include some additional blocks from the stash if they can be placed into the path. Buckets are greedily filled with blocks in the stash in the order of leaf to root, ensuring that blocks get pushed as deep down into the tree as possible. A block a' can be placed in the bucket at level ℓ only if the path $\mathcal{P}(\text{position}[a'])$ to the leaf of block a' intersects the path accessed $\mathcal{P}(x)$ at level ℓ . In other words, if $\mathcal{P}(x, \ell) = \mathcal{P}(\text{position}[a'], \ell)$.



16 | return data

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

At the end, return the block.



Dmytro Bogatov
BU Class of 2023

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.

Data-X Talk

2017-12-17

- └ Path ORAM protocol
 - └ The algorithm
 - └ Subroutines

Subroutines

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.

There are two important subroutines in the algorithm.

READBUCKET(*bucket*) reads all Z blocks — remember, there are always exactly Z blocks per bucket and decrypts them as it reads them.

WRITEBUCKET(*bucket*, *blocks*) writes blocks in the bucket on the server. It pads blocks to make it Z blocks total. Remember that all buckets should be filled to make them indistinguishable to adversary. Blocks are encrypted as they are written using randomized scheme — every cipher text is different for the same plaintext.



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.

Data-X Talk

2017-12-17

- └ Path ORAM protocol
 - └ The algorithm
 - └ Complexity

Complexity

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.

Each access the client reads and writes the whole path, which is order of $\log N$ in size. It re-encrypts each block, so the total complexity is $\mathcal{O}(\log N) \cdot \omega(1)$.

Server, on the other hand, does not perform encryption, so its complexity is $\mathcal{O}(\log N)$.



Dmytro Bogatov
BU Class of 2023

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
    
```

Data-X Talk

└ Path ORAM protocol

└ The algorithm

└ Full algorithm

2017-12-17

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
    
```

Please, take a few moments and look at the whole algorithm. It is a great time now to ask any questions.



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

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

- 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

Our definition of security requires that the access pattern is indistinguishable from random. Encrypted paths look random enough by the definition of secure encryption — and we use randomized encryption since deterministic one does not even satisfy chosen plaintext attack security. The access pattern is random enough since the probability of a particular sequence is $\frac{1}{2^L}$ — uniform in the number of blocks because each access the position table is re-randomized. For M access, the probability is even smaller by the Bayes rule.



2017-12-17

Data-X Talk
└ Example

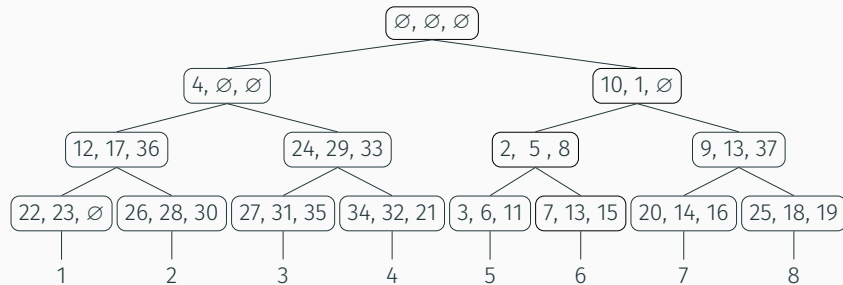
EXAMPLE

EXAMPLE

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

Initial state



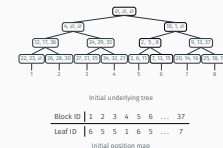
Initial underlying tree

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

Initial position map

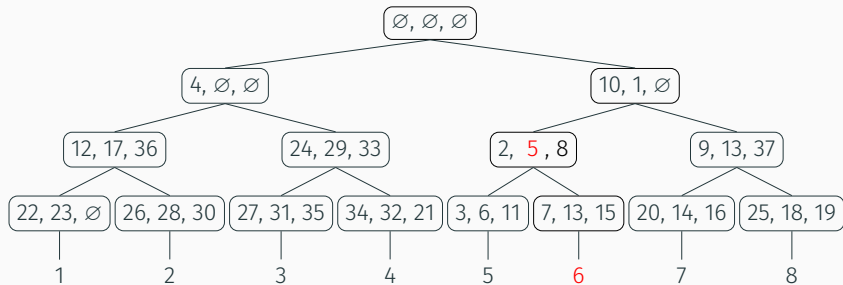
Data-X Talk
Example

Initial state



Here we see the initial state before we access an element. Buckets are filled greedily from the bottom to the top. Leaves are numbered left to right. Each number in the buckets corresponds to encrypted data block and each empty set symbol corresponds to dummy data block. Position table shows the mapping from data blocks to leaves.





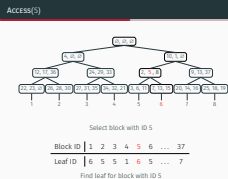
Select block with ID 5

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

Data-X Talk
Example

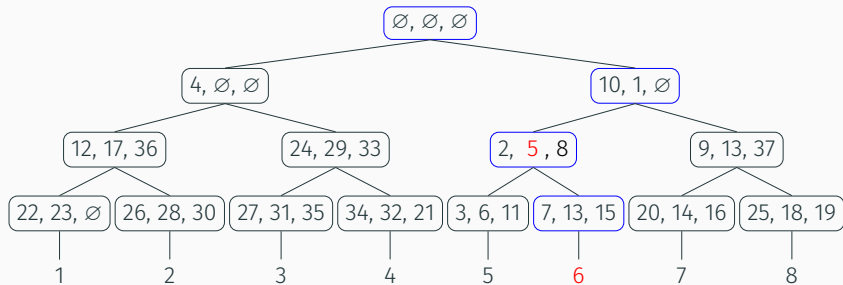
ACCESS(5)



Let us access the data block with ID 5. We first lookup position map to see, which leaf it corresponds to. We see we need a path to the leaf number 6.



Read path



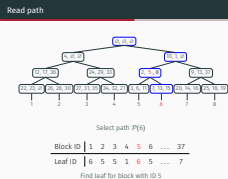
Select path $\mathcal{P}(6)$

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

Data-X Talk
Example

Read path



We read the whole path from the root to the leaf number 6.



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

$position[5] := 3$

Data-X Talk
└ Example

└ Remap block

2017-12-17

Remap block

Stash

```
S = {10, 1, 2, 5, 8, 7, 13, 15}
data = DECRYPTBLOCK(5)
5' ← ENCRYPTDATA(data*)
S = {10, 1, 2, 5', 8, 7, 13, 15}
```

Re-encrypt all blocks in the stash.

Remap

Assign random leaf

position[5] := 3

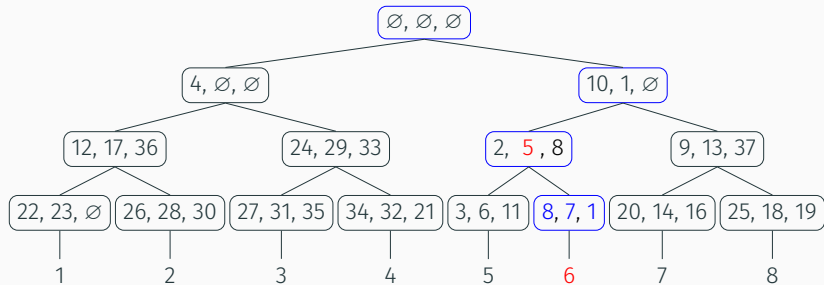
Now, all those data blocks appear in the stash. We know for a fact that our block of interest is there as well. We extract it to data variable. We change block's data and re-encrypt all data blocks. We then put it back to stash.

It is important that we also re-map the block. Let say we assign it a new leaf — number 3.

Now we need to write the path back.



Write path back



Write bucket on level $L = 3$

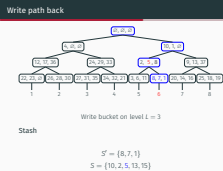
Stash

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

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

Data-X Talk
Example

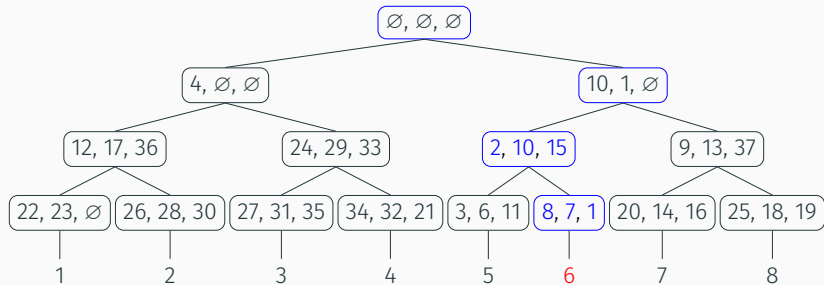
Write path back



We start filling buckets with data blocks greedily from leaves. We pick those $Z = 3$ blocks from the stash which can be placed on the level in path not breaking invariant. If fewer than Z blocks can be placed, we pad it with dummy blocks.



Write path back



Write bucket on level $L - 1 = 2$

Stash

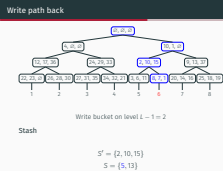
$$S' = \{2, 10, 15\}$$

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

Data-X Talk
Example

Write path back

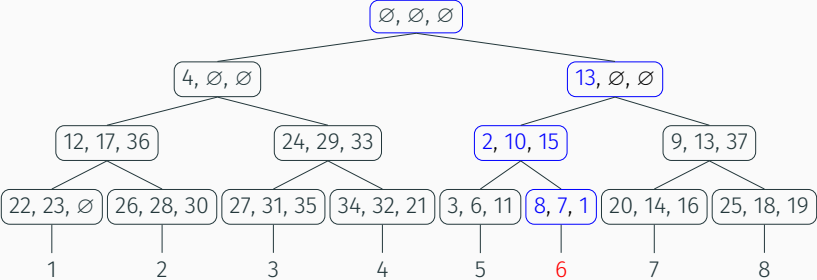
2017-12-17



We repeat the process for the next level.



Write path back



Write bucket on level $L - 2 = 1$

Stash

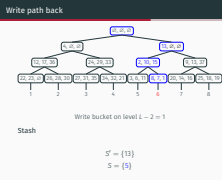
$$S' = \{13\}$$
$$S = \{5\}$$

2017-12-17

Data-X Talk

Example

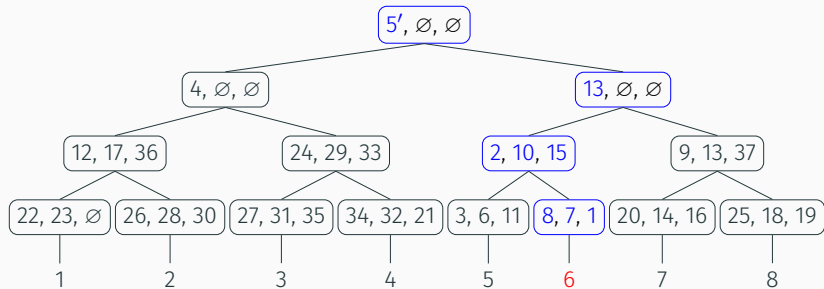
Write path back



Now, only one data block can be placed not breaking the invariant, so we pad.



Write path back



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

Stash

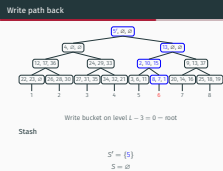
$$S' = \{5\}$$

$$S = \emptyset$$

Data-X Talk
Example

Write path back

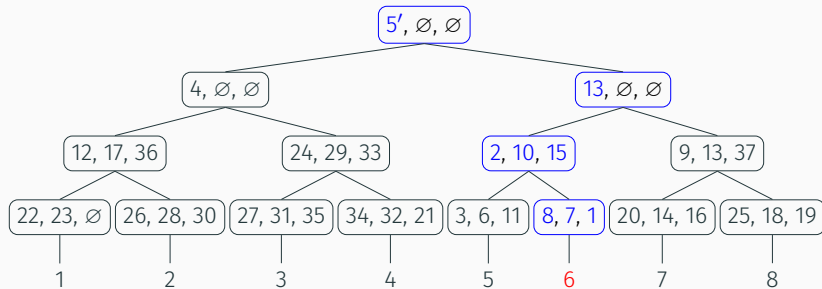
2017-12-17



Finally, we put last block in the root, because it is the only place along the path where $\mathcal{P}(6)$ and $\mathcal{P}(3)$ intersect.



Final state



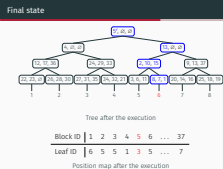
Tree after the execution

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

Position map after the execution

Data-X Talk
Example

Final state



That is how our tree looks like after the operation.

Please note that the adversary only sees that we have read whole path and written whole path. She is not able to see which data block was modified because we re-encrypted everything.

Keep in mind that it is possible that stash is not empty at the end of an operation. It is unlikely to happen. The stash is emptied during the next accesses.



2017-12-17

Data-X Talk

└ Recursion and parametrization

RECURSION AND PARAMETRIZATION

RECURSION AND PARAMETRIZATION

Dmytro Bogatov
BU Class of 2023

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](#)].

One of the ways to make the client even thinner is to use recursion. Zero-level ORAM contains data blocks, the position map of i_{th} ORAM is stored in the $(i + 1)_{\text{st}}$ ORAM, and the client stores the position map for the last ORAM. The access to a block in a zero-level ORAM triggers recursive calls all up to the last ORAM.

The idea of recursion was first described in the works of Shi *et al.* [[Shi+11](#)] and Stefanov, Shi, and Song [[SSS11](#)].



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.

It is proven in the paper that we do not need to have separate stash for every level of recursion to maintain negligible probability of failure. Having single stash reduces client storage by the order of $\log N$.



2017-12-17

Data-X Talk

└ Bounds on stash usage

BOUNDS ON STASH USAGE

BOUNDS ON STASH USAGE

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

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 $\mathbf{s} = (\mathbf{a}, \mathbf{x}, \mathbf{y})$.

└ Bounds on stash usage

└ Probability of failure is negligible

2017-12-17

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 $\mathbf{s} = (\mathbf{a}, \mathbf{x}, \mathbf{y})$.

The whole proof of negligible failure probability is about proving this theorem. It might look complex, but simply put, what it says is that for any ORAM with 5 blocks per bucket, stash usage exceeds some stash size R with probability at most exponentially small with respect to R .

Or, even simpler, the probability of exceeding stash capacity decreases exponentially with the stash size, given that the bucket size Z is large enough.

The authors prove this theorem in three steps.



Probability of failure is negligible

Data-X Talk

└ Bounds on stash usage

└ Probability of failure is negligible

2017-12-17

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)

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)

First, they introduce a second ORAM, called ∞ -ORAM, together with an algorithm that post-processes the stash and buckets of ∞ -ORAM in such a way that the blocks over buckets distribution of ∞ -ORAM and real ORAM are the same.

Second, they show that the stash usage after post-processing is $> R$ if and only if there exists a subtree T for which its “usage” in ∞ -ORAM is more than its “capacity”.

Finally, they show how a mixture of binomial and geometric probability distributions expresses the probability of the number of real blocks that do not get evicted from a subtree after a sequence of load/store operations.



Dmytro Bogatov
BU Class of 2023

2017-12-17

Data-X Talk
└ Evaluation

EVALUATION

EVALUATION

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

- 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.

Stash Occupancy Distribution

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.

Data-X Talk

Evaluation

2017-12-17

Stash Occupancy Distribution

In the experiments, authors used a binary tree of height $\lceil \log_2 N \rceil - 1$.

Let us define *stash occupancy* as the number of overflowing blocks. Thus, this would be a client's persistent local storage in addition to $Z \log_2 N$ transient storage for storing single path.

We access PathORAM in a round-robin fashion — it is proven to be the worst-case scenario for PathORAM. Round-robin fashion is like $\{1, 2, \dots, N, 1, 2, \dots, N, \dots, 1, 2, \dots\}$.

Finally, they perform around billion accesses for warming-up, then 250 billion accesses for actual experiment.



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.

Stash Occupancy Distribution

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.

Data-X Talk Evaluation

Stash Occupancy Distribution

Required stash size grows linearly with the security parameter, which is consistent with the theorem that failure probability decreases exponentially with the stash size. There are two ways to attack our ORAM — break encryption and trigger stash overflow. Authors give approximate values for stash size such that the probability of stash overflow is no greater than the probability of encryption break.

This linearity was used to predict required stash size for realistic security parameters. Note that it is by definition infeasible to simulate for practically adopted security parameters, since that would have proven insecurity.

Finally, it is discovered that required stash size does not depend on N , which proves that PathORAM is highly scalable.

2017-12-17



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.

Another set of observations about bucket load.

For $Z \geq 4$ the usage of a subtree is close to the number of buckets in it. This means, we do not waste space with dummy blocks.

For the levels close to the root, the bucket load is 1 block.

Leaves have slightly heavier loads as blocks accumulate at the leaves of the tree.

$Z = 3$ is different. It produces much larger stash sizes in practice, so should not be used in production.



APPLICATIONS AND EXTENSIONS

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

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

Gentry *et al.* [Gen+13] suggested that ORAMs can be used to perform search on an oblivious binary search tree.

Underlying data structure for PathORAM is an oblivious binary tree. One ACCESS for the ORAM is equivalent to binary tree search. This way, without re-randomization and write back subroutine, PathORAM ACCESS is the same as binary search. Thus, the bandwidth is $\mathcal{O}(\log^2 N)$.



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.

If we are using recursive PathORAM, we can upload and download client state — which is $\mathcal{O}(\log N) \cdot \omega(1)$ — before each access. This way we can build a *stateless* ORAM — potentially, multi-user ORAM.



Secure Processors

Data-X Talk

└ Applications and extensions

└ Secure Processors

2017-12-17

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.

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.

Due to its simplicity, PathORAM is particularly good for silicon implementations. For example, Maas [Maa14] has build such implementation using FPGAs. Fletcher, Dijk, and Devadas [FDD12] and Ren et al. [Ren+13] built a simulator for a 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.

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

It is possible to treat PathORAM internal tree structure as a Merkle tree. Each node is tagged with the hash of the following form, which is a concatenation of hashes of all blocks in the bucket, and the children of the node.



CONCLUSION

Dmytro Bogatov
BU Class of 2023

Dmytro Bogatov
BU Class of 2023

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]



BU Class of 2023

The conclusion given in the paper is very concise, so I have just copied it down from the paper. To iterate, PathORAM is simple, practical and requires small client storage. The paper proves asymptotic bounds and negligible probability of failure. Lastly, the practical evaluation of theoretical results is given.

Dmytro Bogatov

BU Class of 2023

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

2017-12-17

Data-X Talk
└ Conclusion

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>.



- 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>.



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.

2017-12-17

Data-X Talk
└ Conclusion

└ References

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.





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.org/10.1145/2382536.2382540). URL: <http://doi.acm.org/10.1145/2382536.2382540>.

2017-12-17

Data-X Talk
└ Conclusion

└ References

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.org/10.1145/2382536.2382540). URL: <http://doi.acm.org/10.1145/2382536.2382540>.





Jonathan L. Dautrich Jr. and China 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.

2017-12-17

Data-X Talk
└ Conclusion

└ References

- Jonathan L. Dautrich Jr. and China 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.





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>.

2017-12-17

Data-X Talk
└ Conclusion

└ References

- 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>.





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>.

2017-12-17

Data-X Talk
└ Conclusion

└ References

- 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

Data-X Talk

2017-12-17

└ Max stash size grows linearly with λ

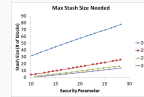


Figure 3 from [Ste+13].

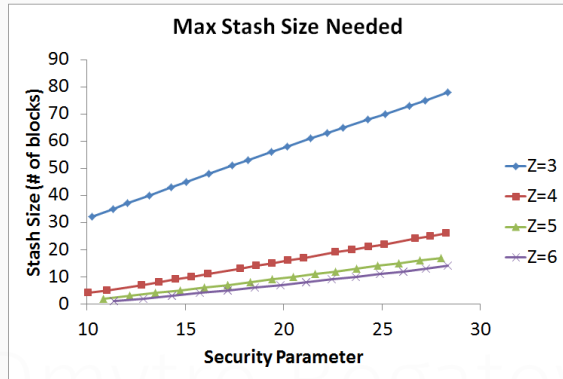


Figure 3 from [Ste+13].



BU Class of 2023

Dmytro Bogatov
BU Class of 2023

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 for large security parameters

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

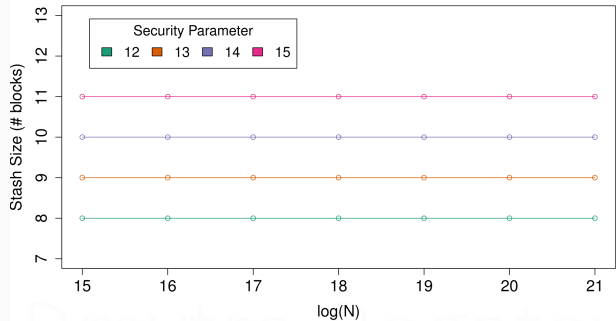


Figure 4 from [Ste+13].



Data-X Talk

2017-12-17

└ Max stash size does not depend on N

Max stash size does not depend on N

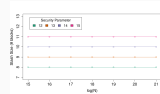


Figure 4 from [Ste+13].