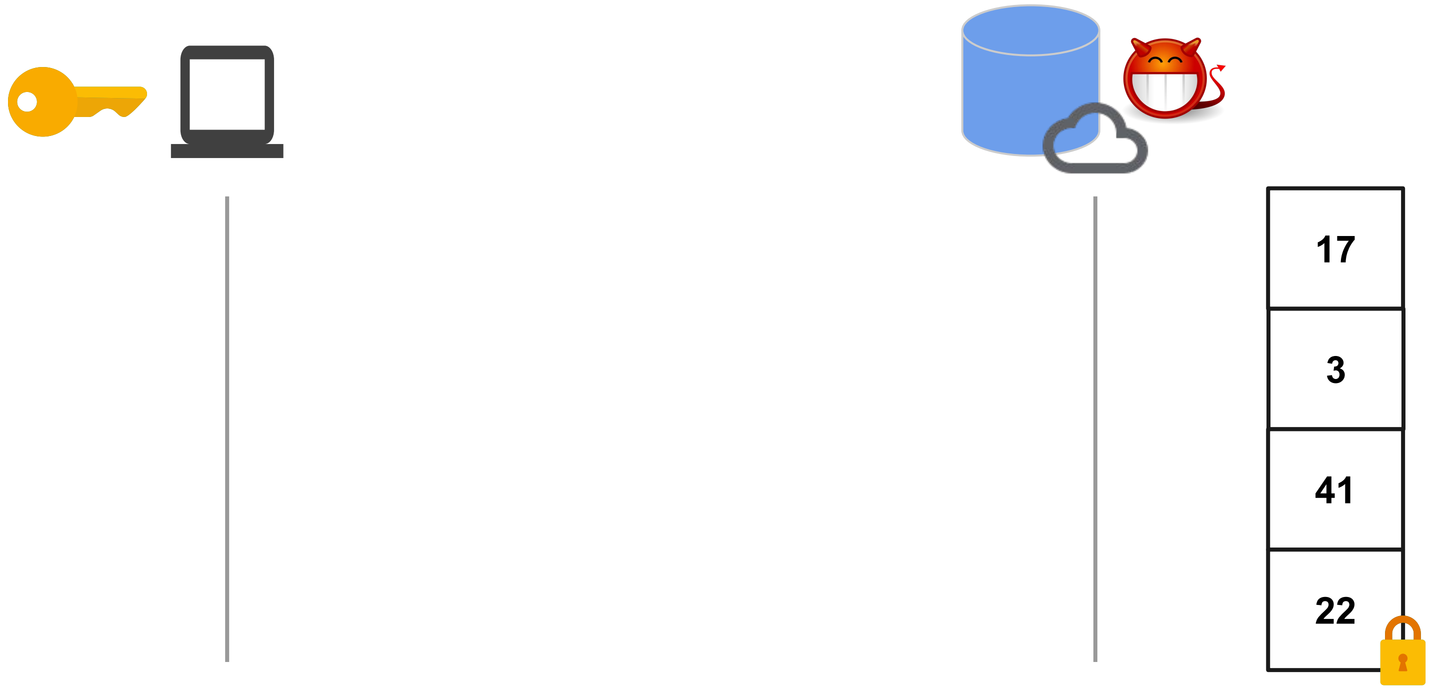
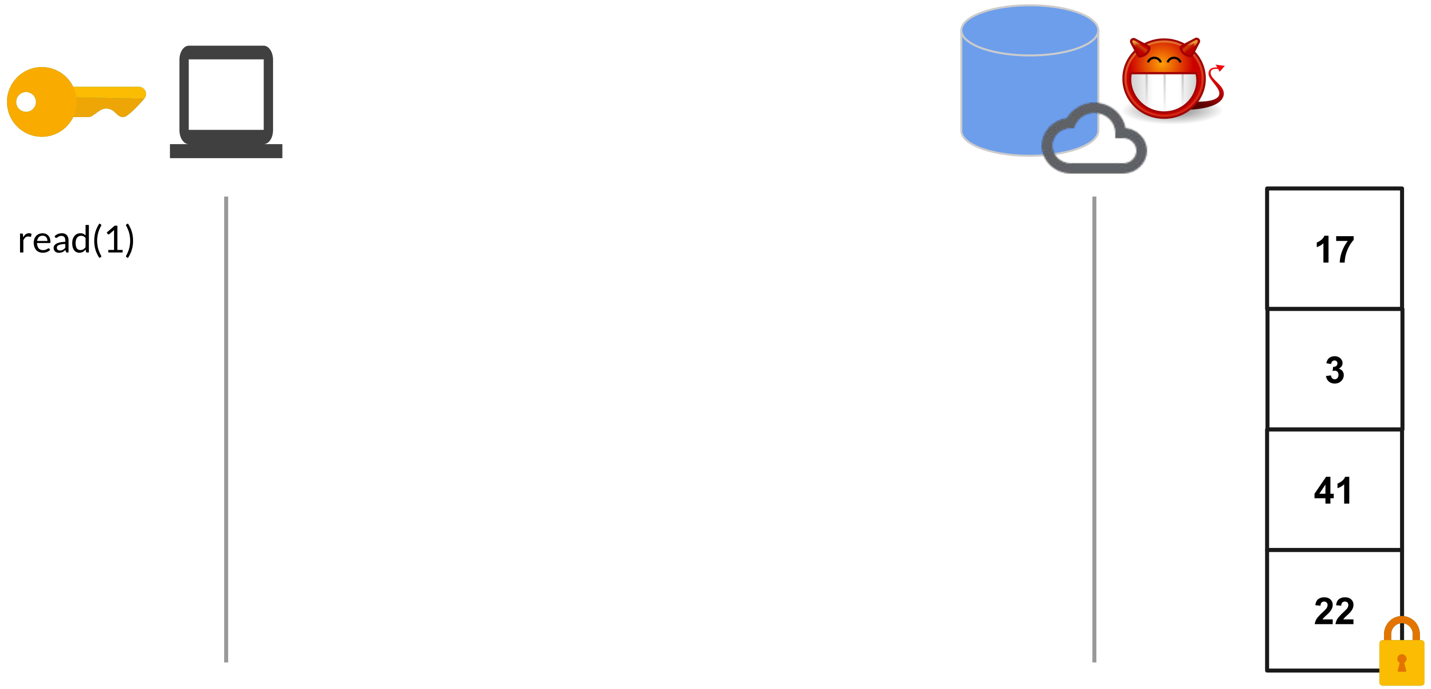
Limits of Breach-Resistant and Snapshot-Oblivious RAMs

Giuseppe Persiano and **Kevin Yeo**

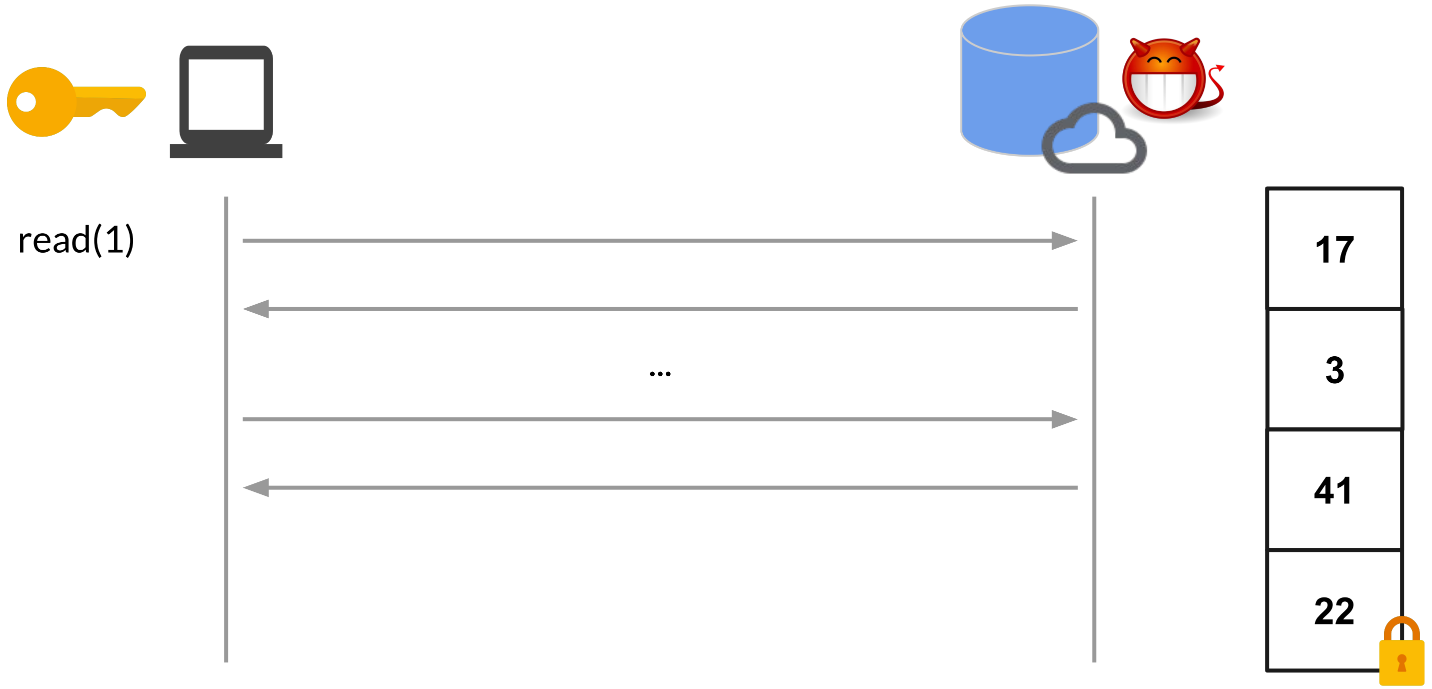
Oblivious RAM [GO'96]



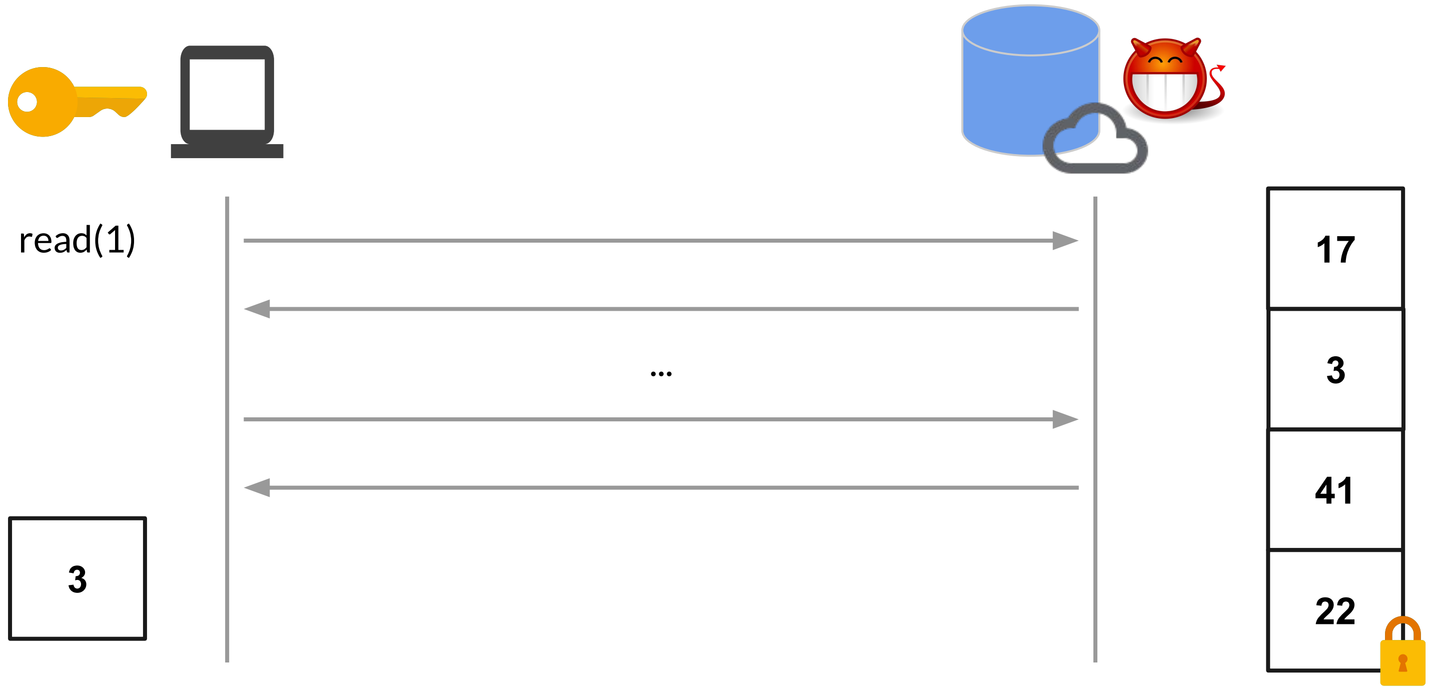
Oblivious RAM



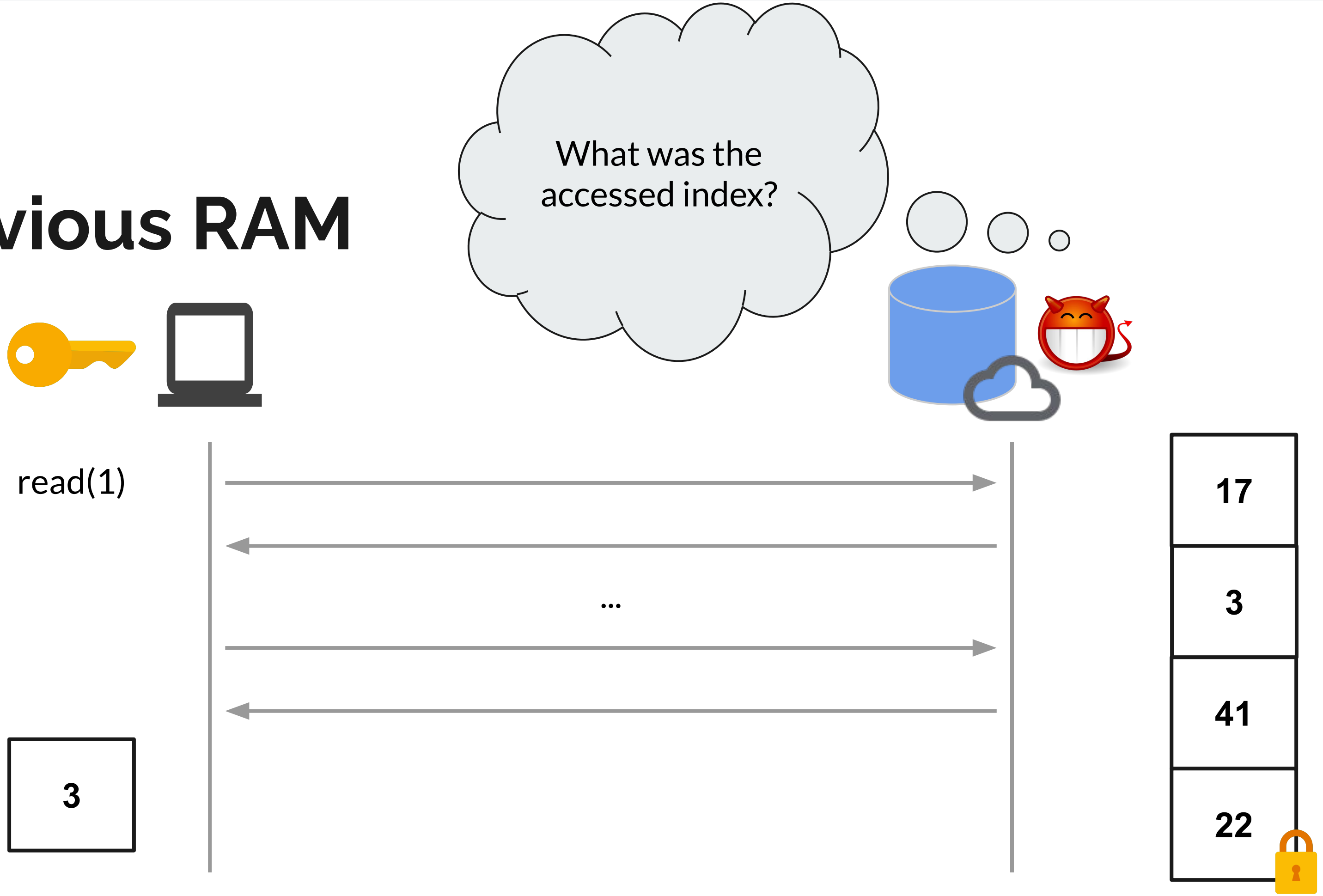
Oblivious RAM



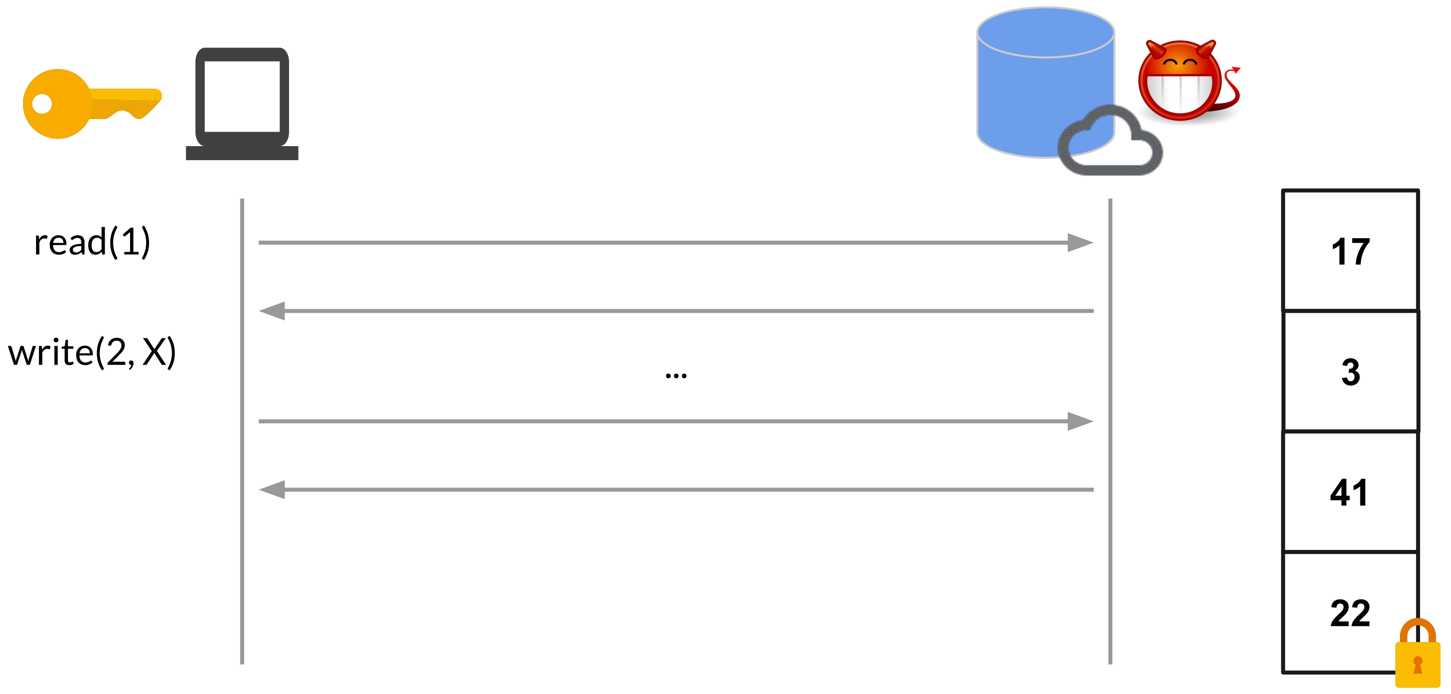
Oblivious RAM



Oblivious RAM



Oblivious RAM

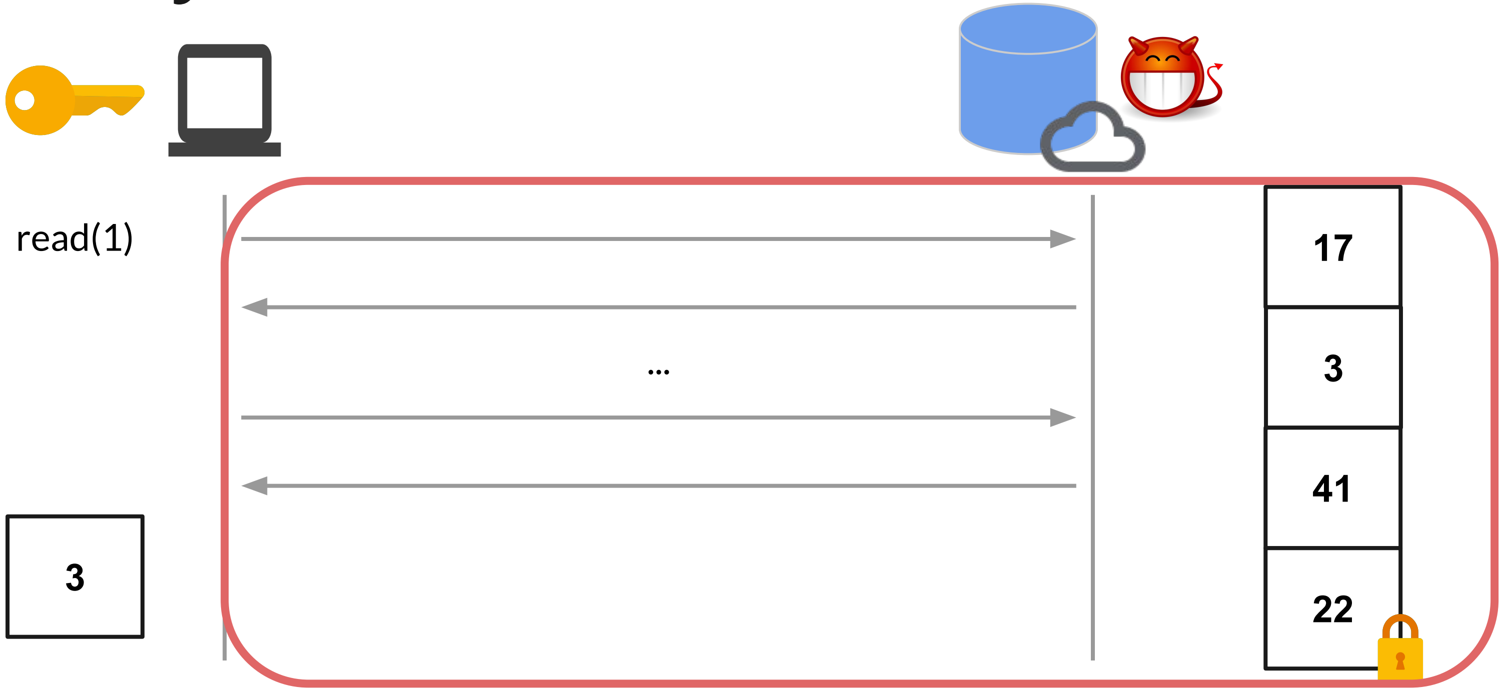


Oblivious RAM Security

Definition. For any sequence of equal-length operations O_1 and O_2, the adversary's view ObvDS(O_1) and ObvDS(O_2) from a construction ODS must be indistinguishable to a computational adversary A:

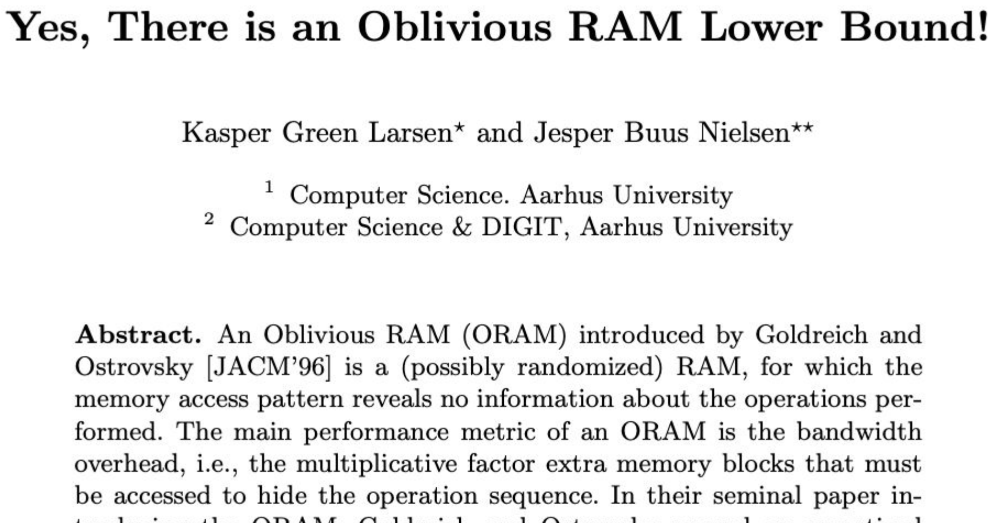
$$\Pr[A(\text{ObvDS}(O_1)) = 1] \cong \Pr[A(\text{ObvDS}(O_2)) = 1]$$

Adversary’s View



ORAM Lower Bound

Theorem [LN‘18]. In the cell probe model, oblivious RAMs require **Ω**(log N) overhead.


Yes, There is an Oblivious RAM Lower Bound!

Kasper Green Larsen* and Jesper Buus Nielsen**

[1] Computer Science. Aarhus University
[2] Computer Science & DIGIT, Aarhus University

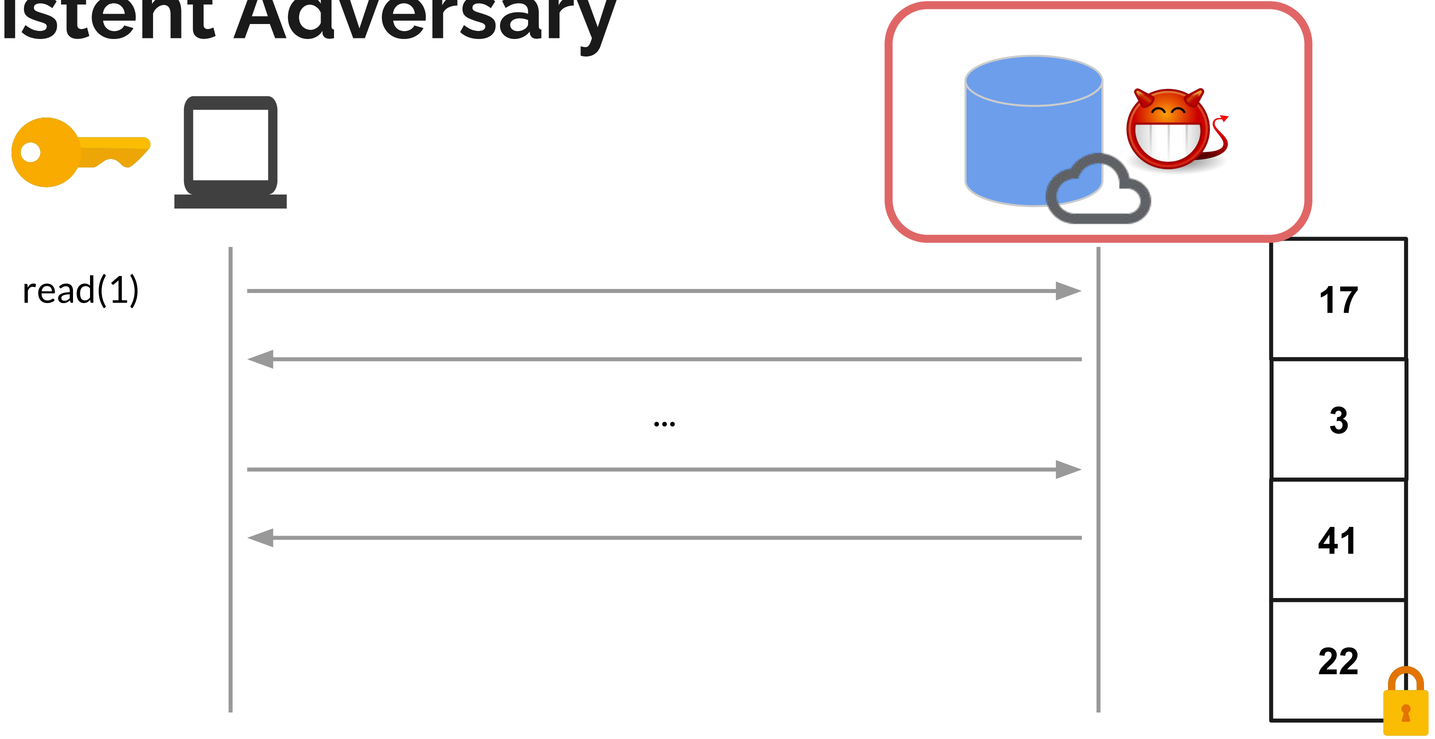
Abstract. An Oblivious RAM (ORAM) introduced by Goldreich and Ostrovsky [JACM’96] is a (possibly randomized) RAM, for which the memory access pattern reveals no information about the operations performed. The main performance metric of an ORAM is the bandwidth overhead, i.e., the multiplicative factor extra memory blocks that must be accessed to hide the operation sequence. In their seminal paper in-

More ORAM Lower Bounds

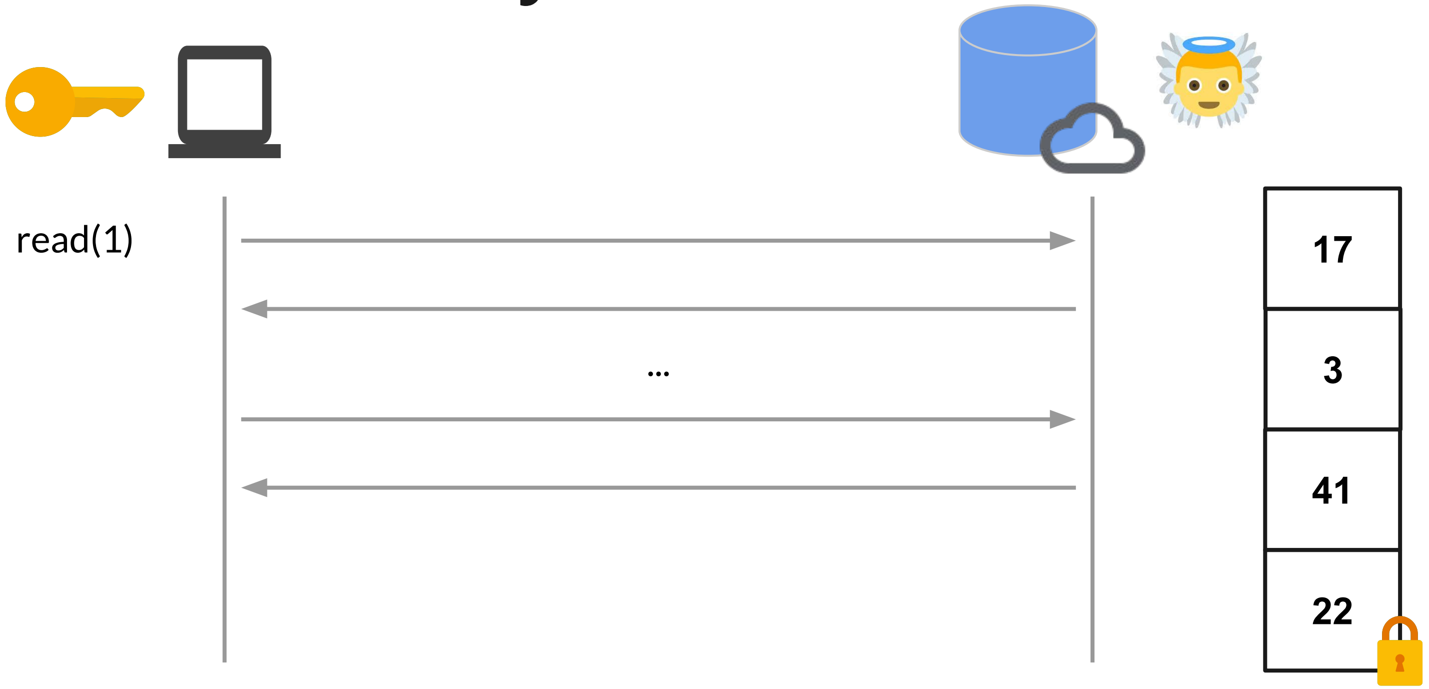
Logarithmic $\Omega(\log N)$ lower bounds have been proven for the many other privacy notions and settings:

- Differential Privacy [PY'19]
- Hidden Operational Boundaries [HKKS'19]
- Multiple Non-Colluding Servers [LSY'20]
- Encrypted Search Leakage [PPY'20]
- Small Data Blocks [KL'21], [PY'23]

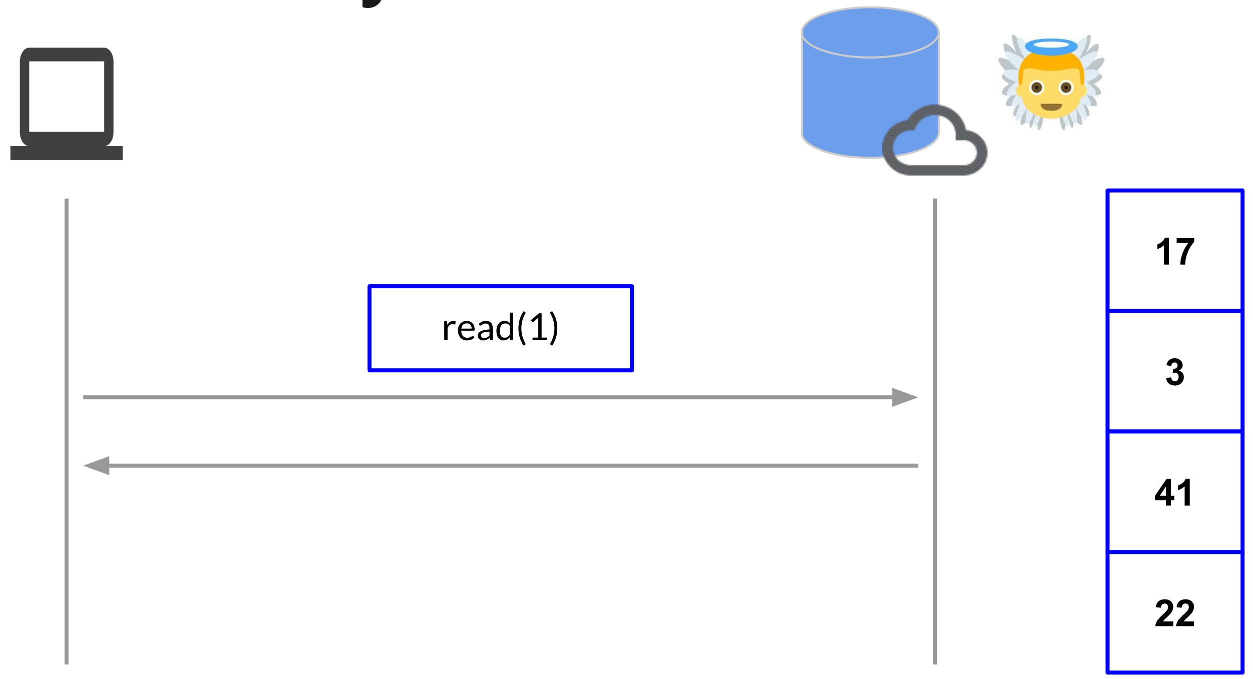
Persistent Adversary



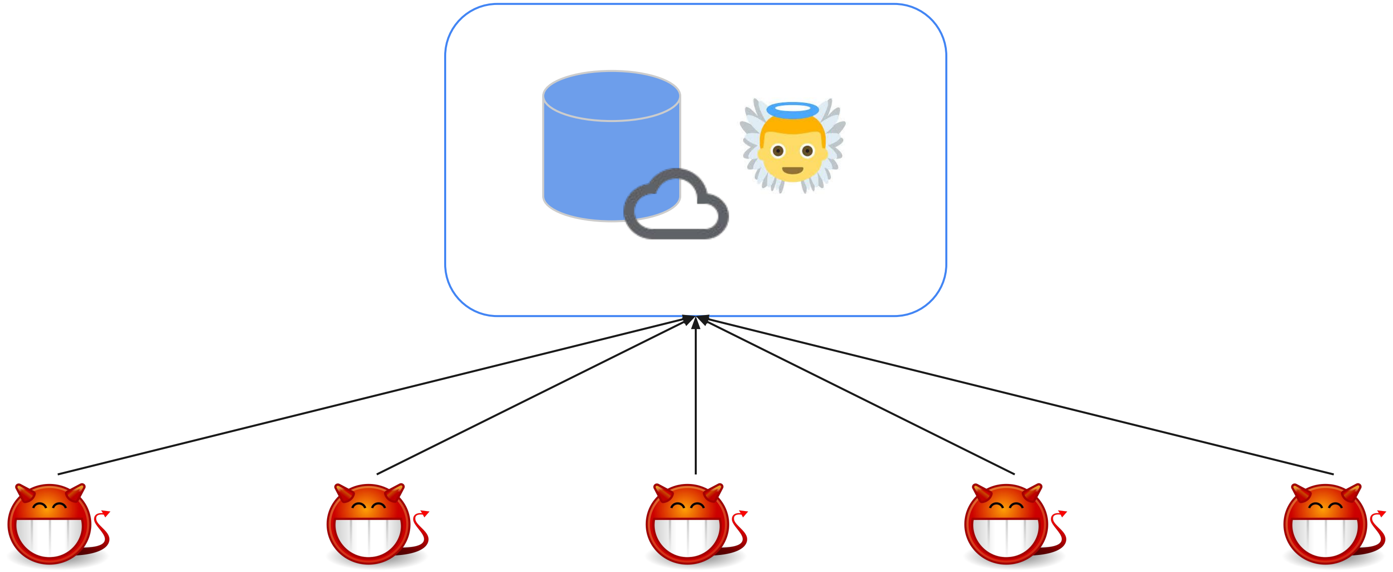
Trusted Third Party



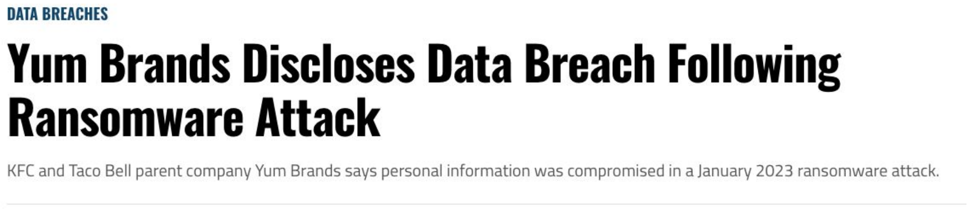
Trusted Third Party



Trusted Third Party



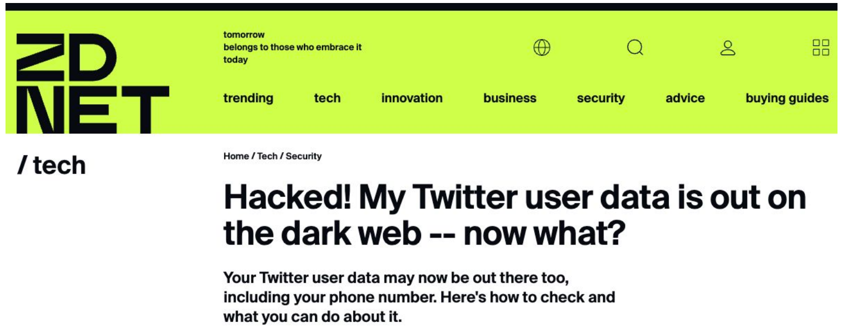
Breaches



DATA BREACHES

Yum Brands Discloses Data Breach Following Ransomware Attack

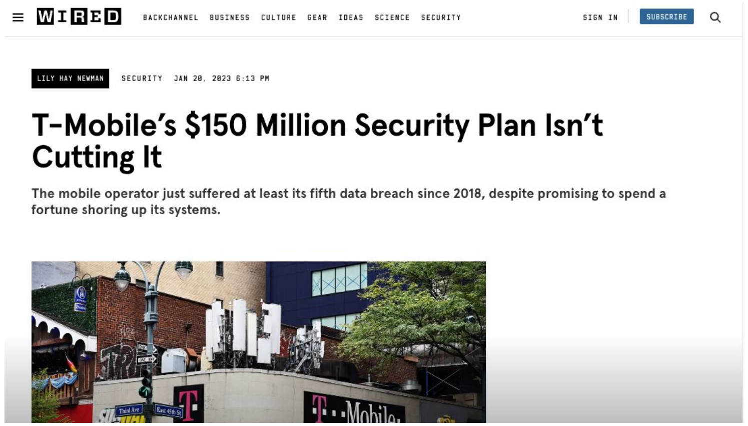
KFC and Taco Bell parent company Yum Brands says personal information was compromised in a January 2023 ransomware attack.



Home / Tech / Security

Hacked! My Twitter user data is out on the dark web -- now what?

Your Twitter user data may now be out there too, including your phone number. Here's how to check and what you can do about it.

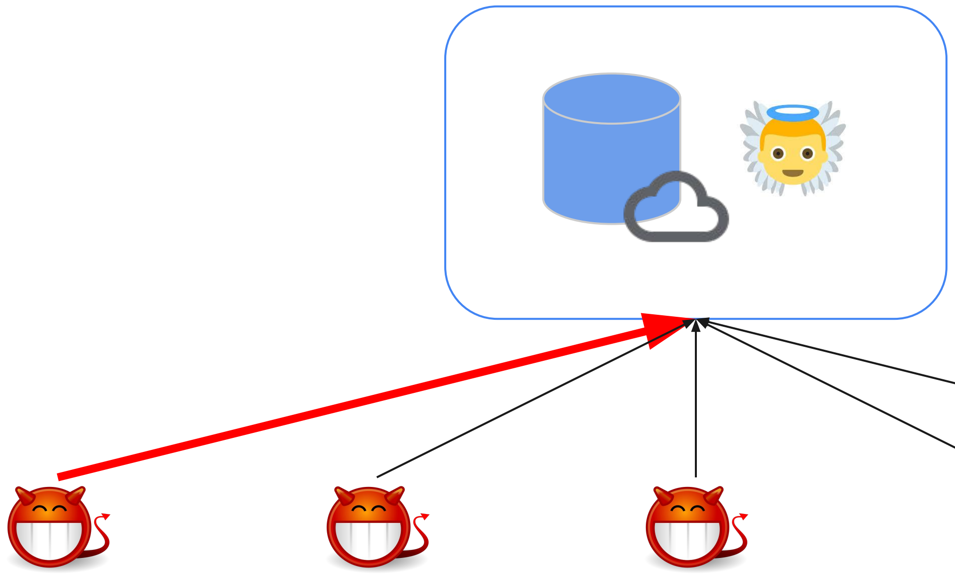


LILY HAY NEWMAN SECURITY JAN 28, 2023 6:13 PM

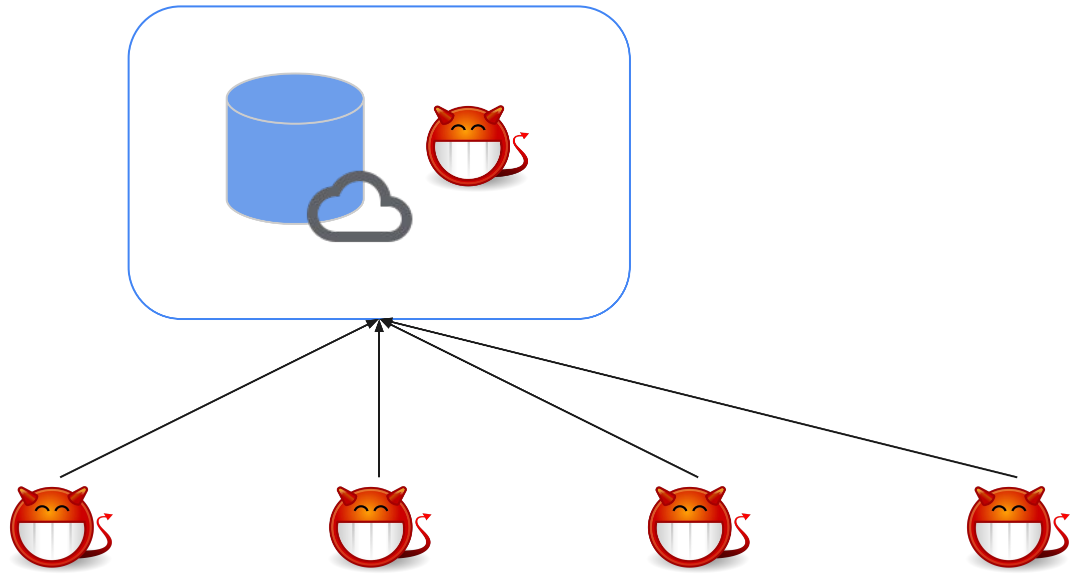
T-Mobile's $150 Million Security Plan Isn't Cutting It

The mobile operator just suffered at least its fifth data breach since 2018, despite promising to spend a fortune shoring up its systems.

Trusted Third Party



Breached Third Party



Breach Remediation

Good News:

Most breaches are remediated within a few days according to DBIR: Data Breach Investigations Report.

Bad News:

Attackers can download and observe activity during those few days.

Adversary’s View

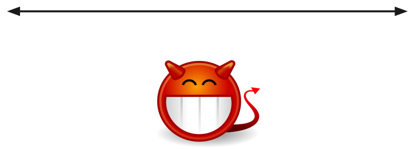
read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

Persistent Adversary's View

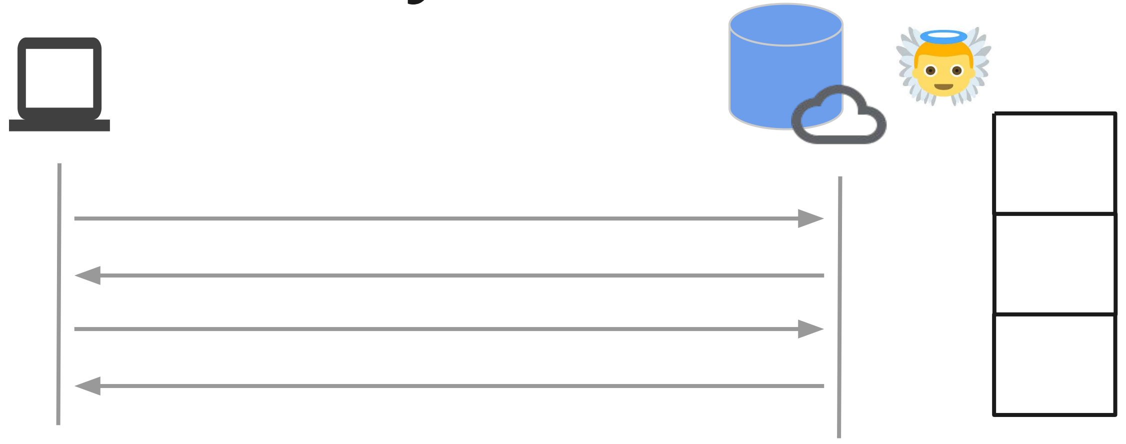
read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

Snapshot Adversary's View

read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

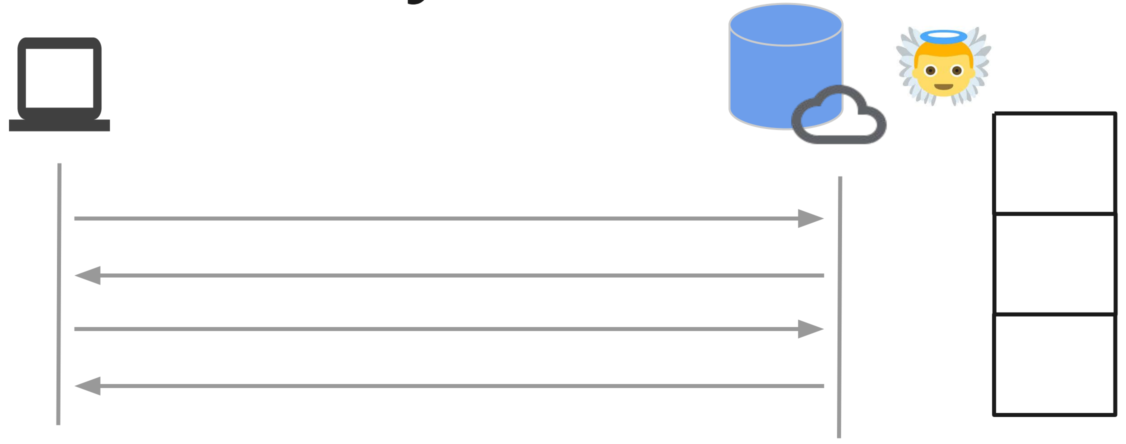


Snapshot Adversary's View

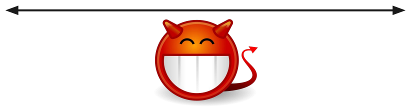


read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

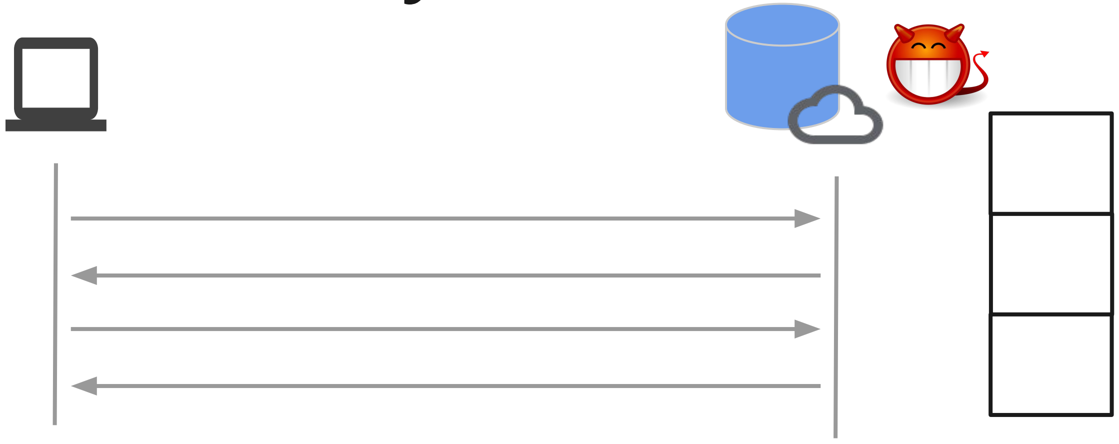
Snapshot Adversary's View



read(10), **write(17)**, read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

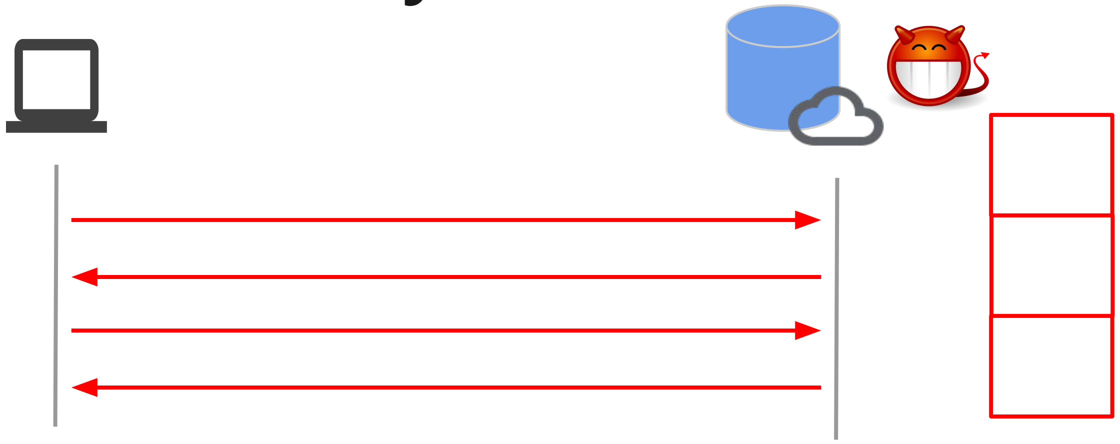


Snapshot Adversary's View

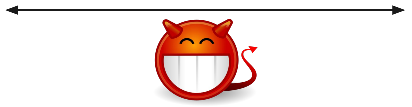


read(10), write(17), **read(2)**, write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

Snapshot Adversary's View



read(10), write(17), **read(2)**, write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)



(S, L)-Snapshot Adversary

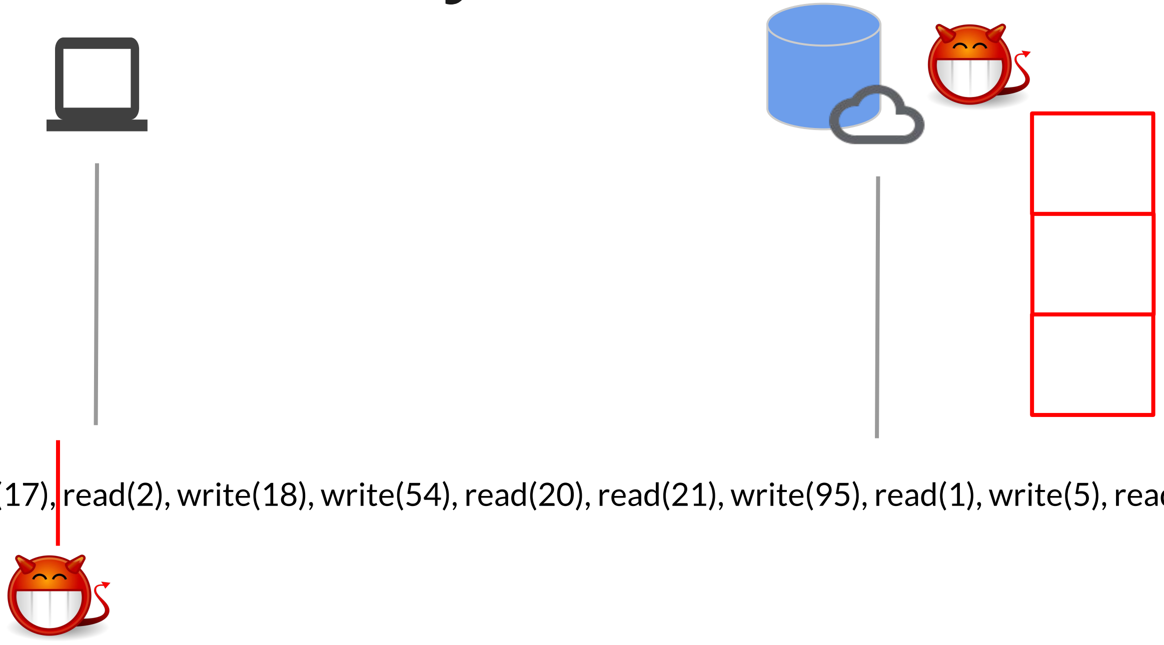
Definition. An (S, L)-snapshot adversary may perform at most S breaches such that at most L operations are performed during the S breaches. The adversary sees all server memory contents during breaches as well as the transcripts of the L operations during the S breaches.

(S, L)-Snapshot Adversary

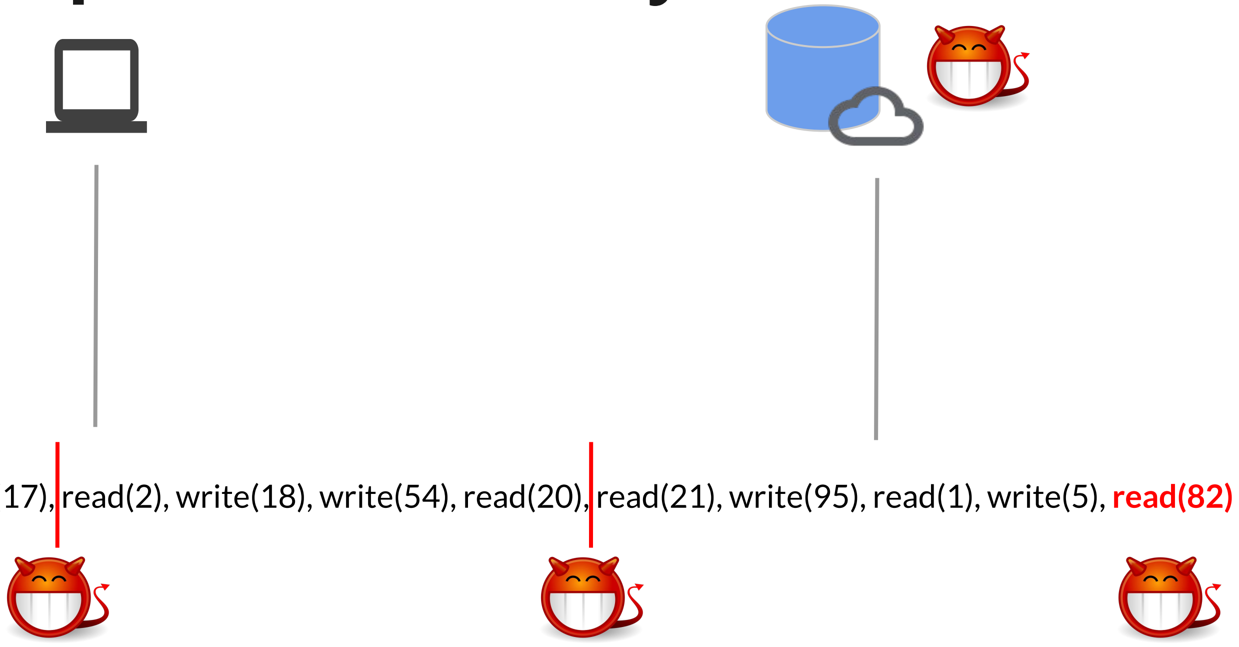
Definition. An (S, L)-snapshot adversary may perform at most S breaches such that at most L operations are performed during the S breaches. The adversary sees all server memory contents during breaches as well as the transcripts of the L operations during the S breaches.

Note. It is possible that L < S to simulate data breaches.

Snapshot Adversary's View


read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

(3, 1)-Snapshot Adversary's View

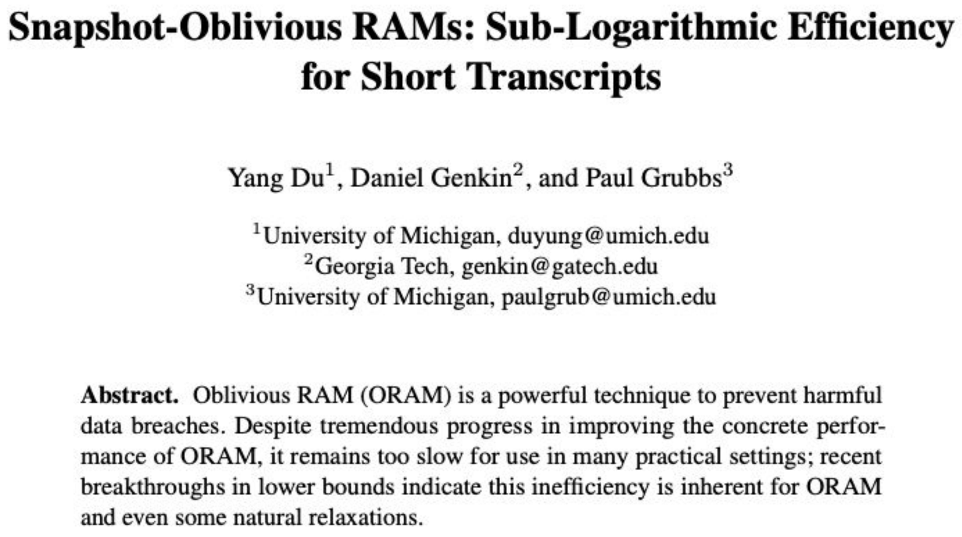

read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

(S, L)-Snapshot-Oblivious RAM

Definition. For any sequence of equal-length operations O_1 and O_2, the adversary's view ObvDS(O_1) and ObvDS(O_2) from a construction ODS must be indistinguishable to any **(S, L)-snapshot adversary A**:

$$\Pr[A(\mathrm{ObvDS}(O_1)) = 1] \cong \Pr[A(\mathrm{ObvDS}(O_2)) = 1]$$

(1,L)-Snapshot-Oblivious RAMs



Snapshot-Oblivious RAMs: Sub-Logarithmic Efficiency for Short Transcripts

Yang Du[1], Daniel Genkin[2], and Paul Grubbs[3]

[1]University of Michigan, duyung@umich.edu
[2]Georgia Tech, genkin@gatech.edu
[3]University of Michigan, paulgrub@umich.edu

Abstract. Oblivious RAM (ORAM) is a powerful technique to prevent harmful data breaches. Despite tremendous progress in improving the concrete performance of ORAM, it remains too slow for use in many practical settings; recent breakthroughs in lower bounds indicate this inefficiency is inherent for ORAM and even some natural relaxations.

(1,L)-Snapshot Adversaries

Theorem [Du, Genkin, Grubbs '23]. Assuming one-way functions, there exists an (1,L)-snapshot-oblivious RAM with O(log L) overhead.

Corollary. For $L = N^{o(1)}$, the above (1, L)-snapshot-oblivious RAM has o(log N) overhead circumventing oblivious RAM lower bounds with persistent adversaries.

Limits of (1,L)-Snapshot-Oblivious

Many providers get breached multiple times:

- 6+ large organizations were breached at least 3 times in the past decade

(1,L)-snapshot-oblivious RAMs cannot protect against multiple breaches

read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

Open Question

What is the complexity of ORAM against adversaries with multiple breaches?

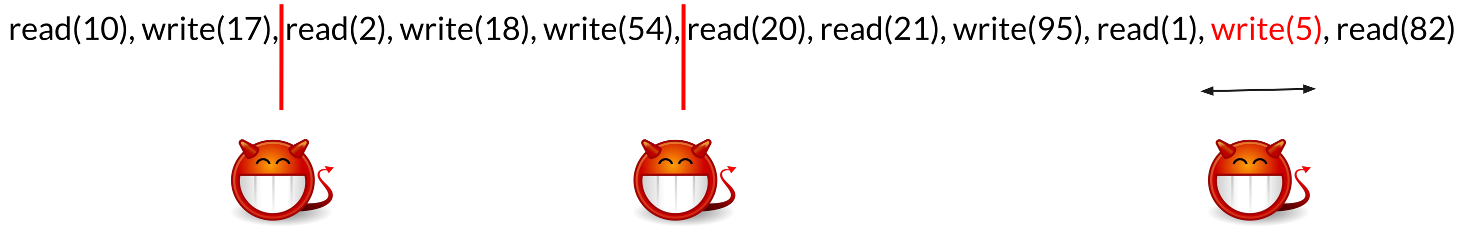
Our Contributions

Theorem. In the cell probe model, any (3,1)-snapshot oblivious RAM must have $\Omega(\log N)$ overhead.

(3,1)-Snapshot Adversary

Performs three breaches:

- Sees only memory contents twice
- See transcript of one query exactly once



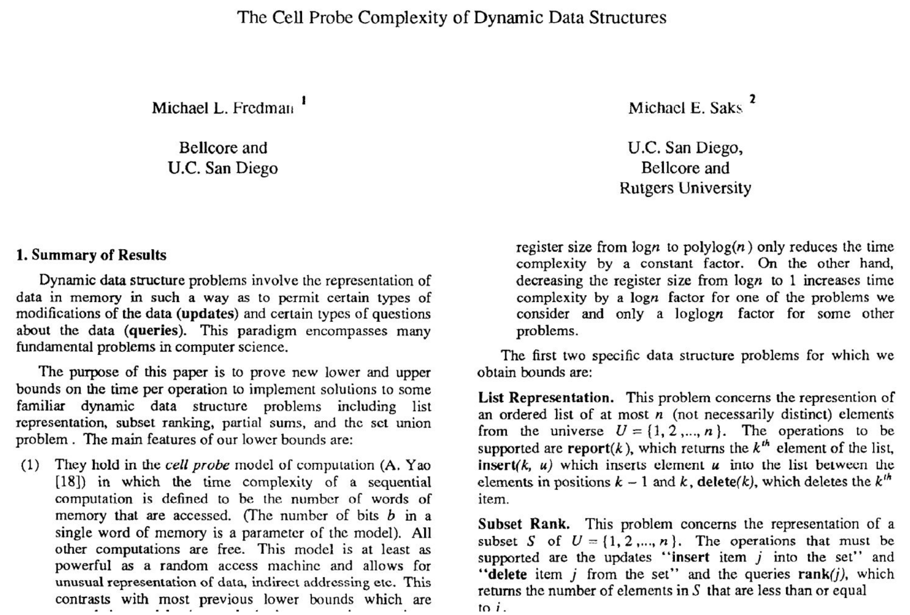
read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

Our Contributions

Theorem. In the cell probe model, any (3,1)-snapshot oblivious RAM must have $\Omega(\log N)$ overhead.

Takeaway. Building oblivious RAMs against general snapshot adversaries with three breaches is as hard as protecting against persistent adversaries.

Chronogram



The Cell Probe Complexity of Dynamic Data Structures

Michael L. Fredman[1]

Bellcore and
U.C. San Diego

Michael E. Saks[2]

U.C. San Diego,
Bellcore and
Rutgers University

1. Summary of Results

Dynamic data structure problems involve the representation of data in memory in such a way as to permit certain types of modifications of the data **(updates)** and certain types of questions about the data **(queries)**. This paradigm encompasses many fundamental problems in computer science.

The purpose of this paper is to prove new lower and upper bounds on the time per operation to implement solutions to some familiar dynamic data structure problems including list representation, subset ranking, partial sums, and the set union problem . The main features of our lower bounds are:

(1) They hold in the *cell probe* model of computation (A. Yao [18]) in which the time complexity of a sequential computation is defined to be the number of words of memory that are accessed. (The number of bits b in a single word of memory is a parameter of the model). All other computations are free. This model is at least as powerful as a random access machine and allows for unusual representation of data, indirect addressing etc. This contrasts with most previous lower bounds which are

register size from $\log n$ to polylog(n) only reduces the time complexity by a constant factor. On the other hand, decreasing the register size from $\log n$ to 1 increases time complexity by a $\log n$ factor for one of the problems we consider and only a $\log\log n$ factor for some other problems.

The first two specific data structure problems for which we obtain bounds are:

List Representation. This problem concerns the represention of an ordered list of at most n (not necessarily distinct) elements from the universe $U = \{1, 2, ..., n\}$. The operations to be supported are **report**(k), which returns the k^{th} element of the list, **insert**(k, u) which inserts element u into the list between the elements in positions $k-1$ and k, **delete**(k), which deletes the k^{th} item.

Subset Rank. This problem concerns the representation of a subset S of $U = \{1, 2, ..., n\}$. The operations that must be supported are the updates "**insert** item j into the set" and "**delete** item j from the set" and the queries **rank**(j), which returns the number of elements in S that are less than or equal to j.

Chronogram

- Logarithmic lower bound for DPRAMs [PY ‘20]
- Super-logarithmic lower bounds for (statistically) oblivious near-neighbor search [LMWY ‘20]
- Logarithmic lower bounds for small block sizes and general data structures [PY’23]
- All for persistent adversaries

Revisiting PY'20

write(1), write(2), write(3), …, write(n), **read(x)**

Revisiting PY'20

Epoch k with r^k writes	...	Epoch 1 with r^1 writes	Epoch 0 with r^0 writes
write(1), write(2), …, write(r^k)	...	write(...), … write(...)	write(n)

Revisiting PY’20

Revisiting PY’20

Label each cell with epoch to last overwrite the contents

5	12	7	0	7	6
4	12	10	8	11	9
2	6	5	11	7	5
5	9	11	12	8	10
…	…	…	…	…	…
5	10	7	11	9	12

Revisiting PY'20

- Fix epoch i
    - For concreteness, let's say i = 5

Revisiting PY'20

- Fix epoch i
  - For concreteness, let's say i = 5
- What does adversary need to observe?

Revisiting PY'20

5	12	7	0	7	6
4	12	10	8	11	9
2	6	5	11	7	5
5	9	11	12	8	10
...	...	...	...	...	...
5	10	7	11	9	12

Revisiting PY'20

5	12	7	0	7	6
4	12	10	8	11	9
2	6	5	11	7	5
5	9	11	12	8	10
...	...	...	...	...	...
5	10	7	11	9	12

Revisiting PY’20

5	12	7	0	7	6
4	12	10	8	11	9
2	6	5	11	7	5
5	9	11	12	8	10
...	...	...	...	...	...
5	10	7	11	9	12

Revisiting PY'20

- Fix epoch i
  - For concreteness, let's say i = 5
- What does adversary need to observe?
  - **Cells last overwritten before epoch 5**
  - **Cells last overwritten in epoch 5**
  - **Cells last overwritten after epoch 5**

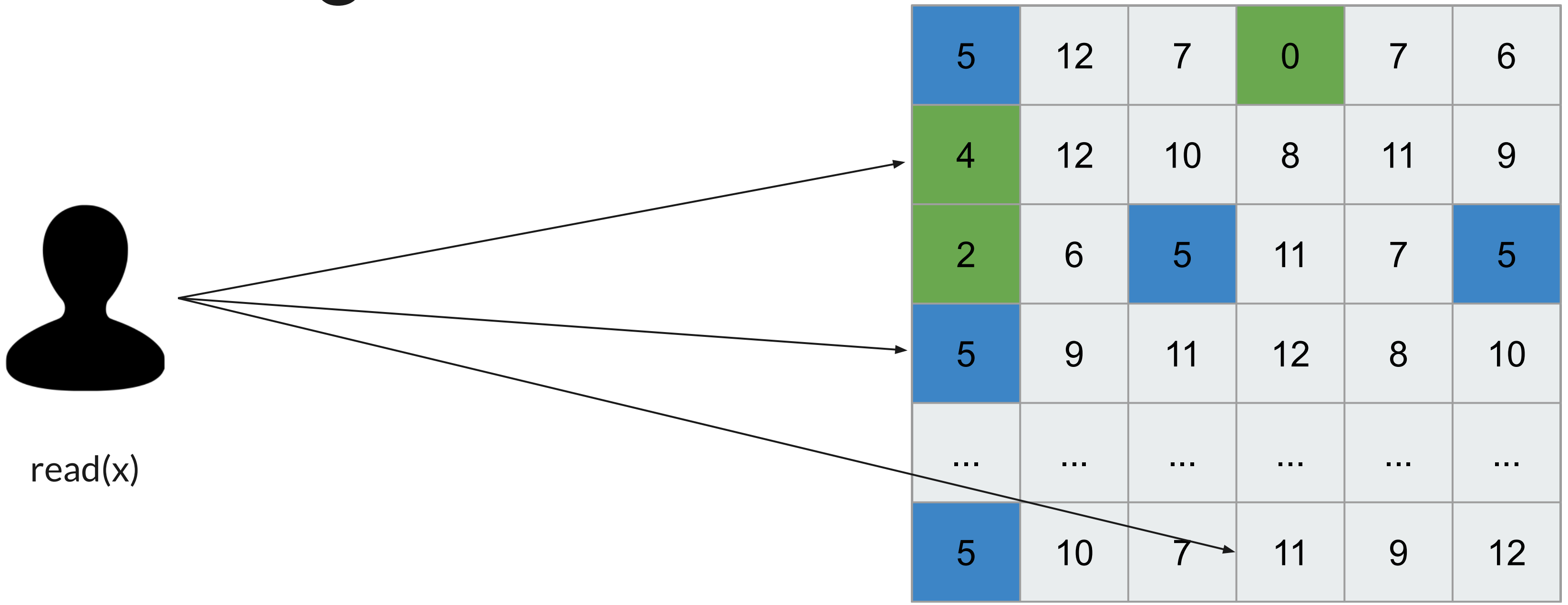
Revisiting PY'20

write(1), write(2), write(3), …, write(n), **read(x)**

Revisiting PY’20

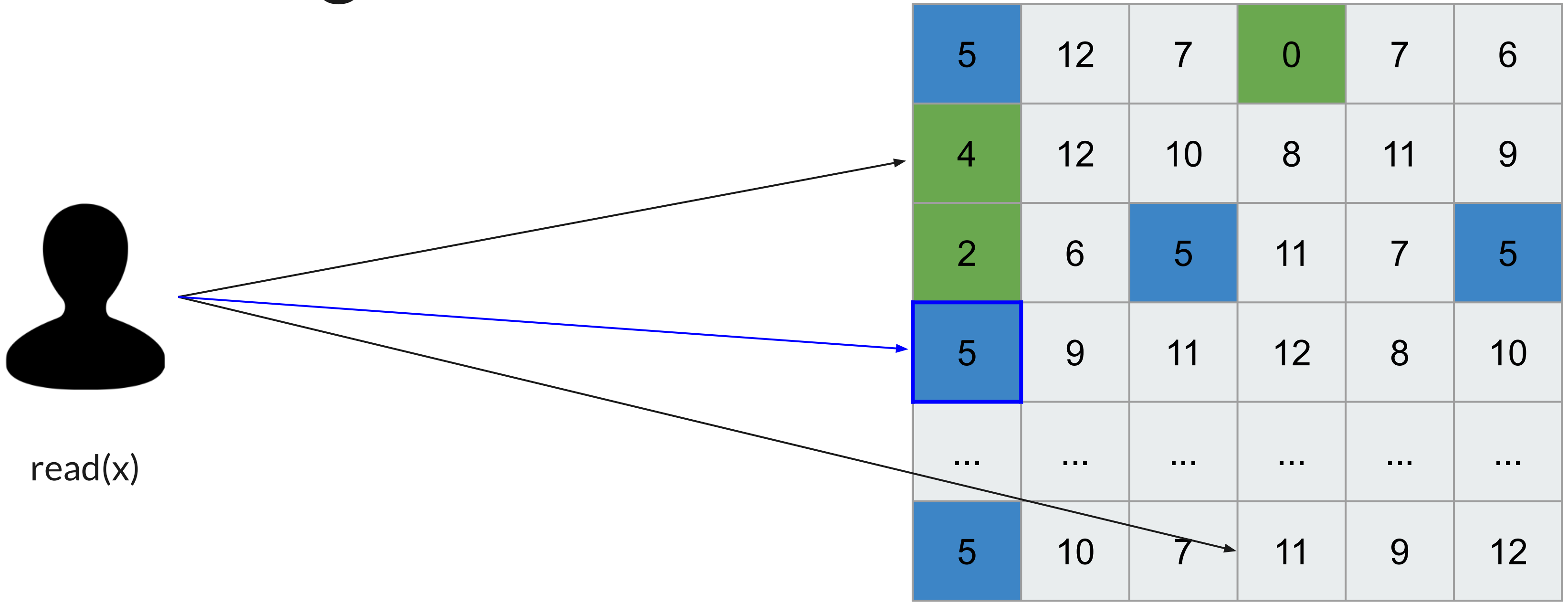
- Fix epoch i
  - For concreteness, let’s say i = 5
- What does adversary need to observe?
  - Cells last overwritten before epoch 5
  - Cells last overwritten in epoch 5
  - Cells last overwritten after epoch 5
- **Test whether the final read probes a cell last overwritten in epoch 5**

Revisiting PY'20



5	12	7	0	7	6
4	12	10	8	11	9
2	6	5	11	7	5
5	9	11	12	8	10
...	...	...	...	...	...
5	10	7	11	9	12

Revisiting PY’20



5	12	7	0	7	6
4	12	10	8	11	9
2	6	5	11	7	5
5	9	11	12	8	10
…	…	…	…	…	…
5	10	7	11	9	12

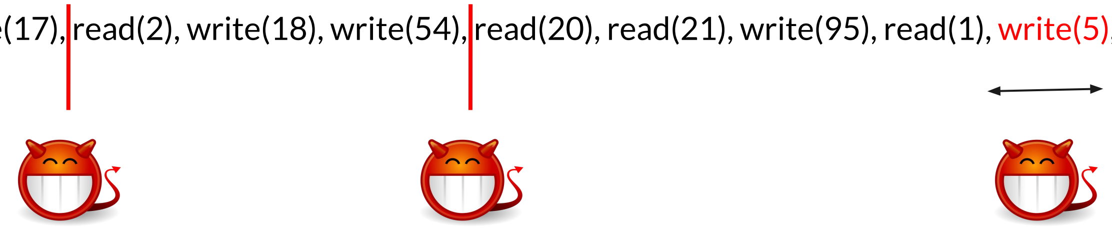
(3,1)-Snapshot Adversary

Label each cell with epoch to last overwrite the contents

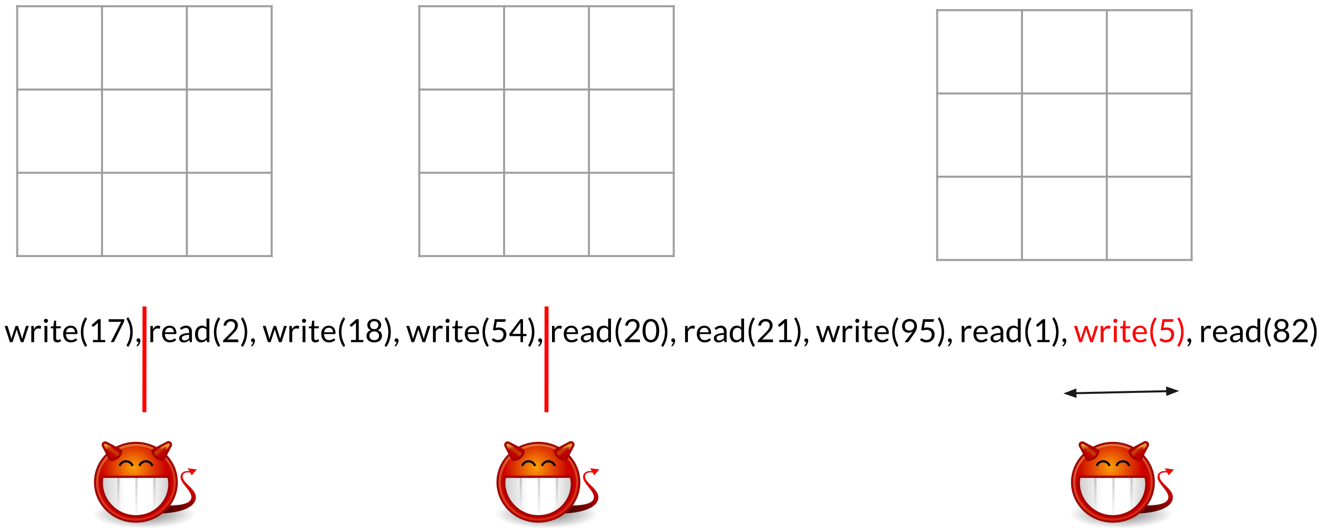
??	??	??	??	??	??
??	??	??	??	??	??
??	??	??	??	??	??
??	??	??	??	??	??
...	...	...	...	...	...
??	??	??	??	??	??

(3,1)-Snapshot Adversary

read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

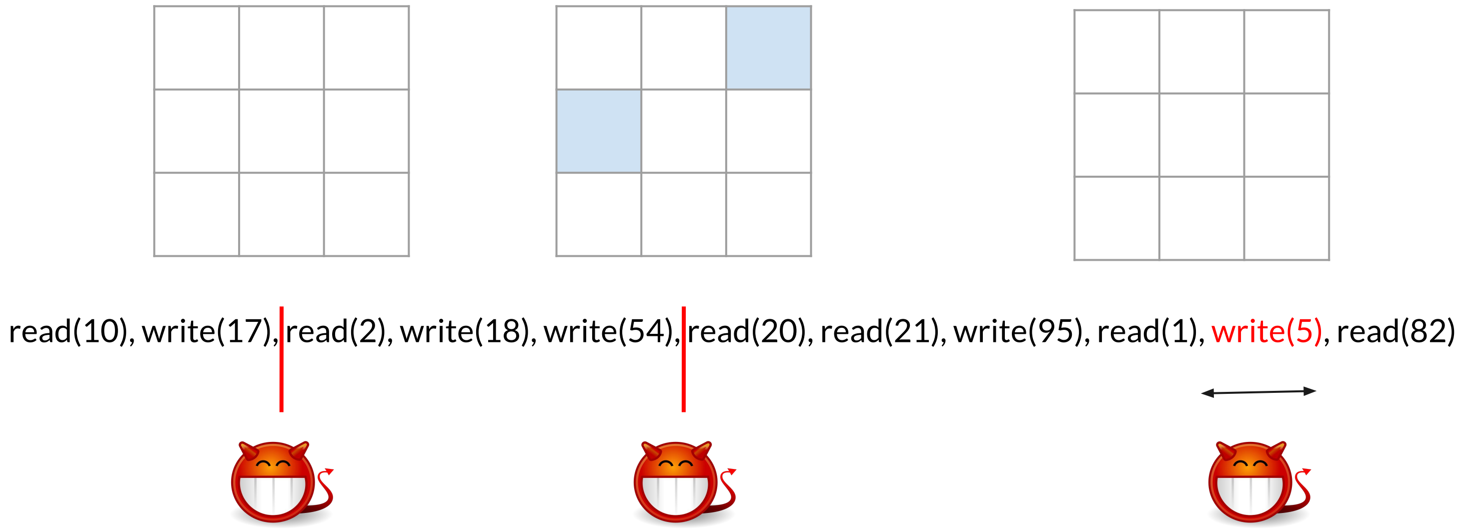


(3,1)-Snapshot Adversary



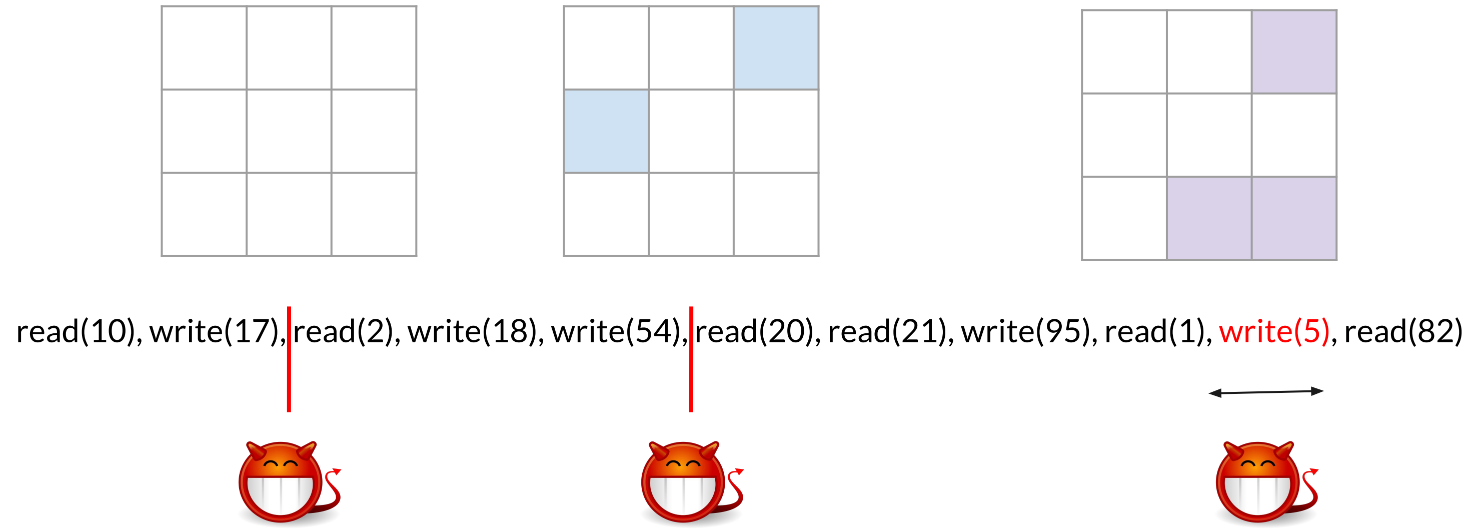
read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

(3,1)-Snapshot Adversary



read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

(3,1)-Snapshot Adversary


read(10), write(17), read(2), write(18), write(54), read(20), read(21), write(95), read(1), write(5), read(82)

(3, 1)-Snapshot Adversary

...	Epoch 5 with r^5 writes	...	Final Read
...	write(...), write(...), ..., write(...)	...	**read(x)**

First Breach

> 5	> 5	> 5	> 5	> 5	> 5
> 5	> 5	> 5	> 5	> 5	> 5
> 5	> 5	> 5	> 5	> 5	> 5
> 5	> 5	> 5	> 5	> 5	> 5
...	...	...	...	...	...
> 5	> 5	> 5	> 5	> 5	> 5

Second Breach

> 5	> 5	> 5	**5**	> 5	> 5
> 5	> 5	> 5	> 5	> 5	> 5
5	> 5	**5**	> 5	> 5	> 5
> 5	> 5	> 5	> 5	**5**	> 5
...	...	...	...	...	...
> 5	**5**	> 5	> 5	> 5	> 5

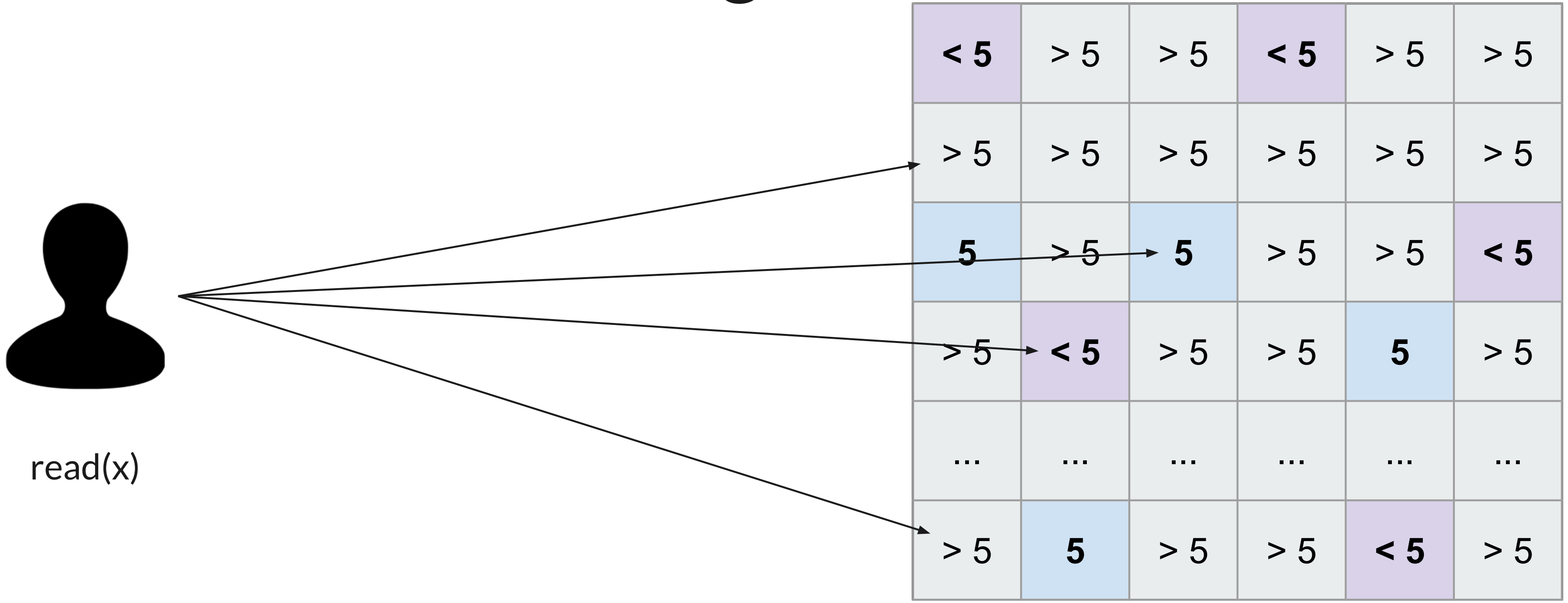
Third Breach (Before Read)

< 5	> 5	> 5	**< 5**	> 5	> 5
> 5	> 5	> 5	> 5	> 5	> 5
5	> 5	**5**	> 5	> 5	**< 5**
> 5	**< 5**	> 5	> 5	**5**	> 5
...	...	...	...	...	...
> 5	**5**	> 5	> 5	**< 5**	> 5

Our Snapshot Lower Bound

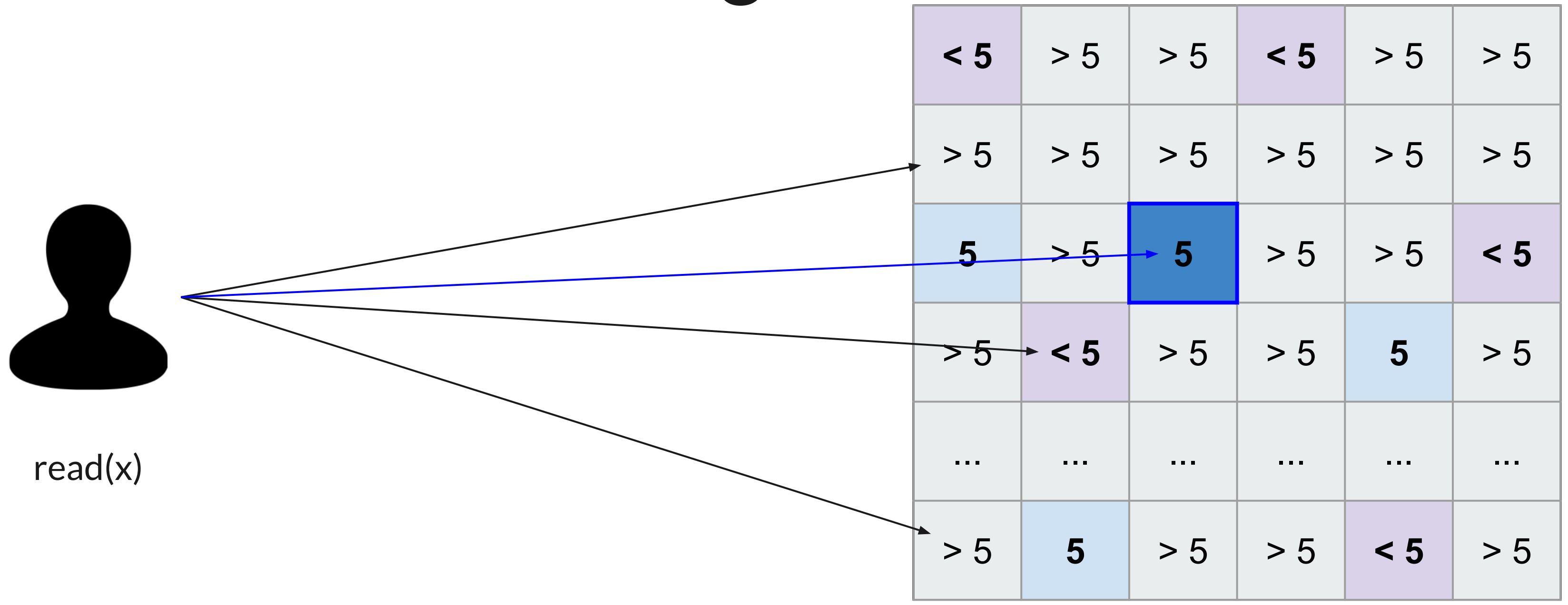
- Fix epoch i
    - For concreteness, let’s say i = 5
- What does adversary need to observe?
    - **Cells last overwritten before epoch 5**
    - **Cells last overwritten in epoch 5**
    - **Cells last overwritten after epoch 5**
- Test whether the final read probes a cell last overwritten in epoch 5

Third Breach (During Read)



< 5	> 5	> 5	**< 5**	> 5	> 5
> 5	> 5	> 5	> 5	> 5	> 5
5	> 5	**5**	> 5	> 5	**< 5**
> 5	**< 5**	> 5	> 5	**5**	> 5
...	...	...	...	...	...
> 5	**5**	> 5	> 5	**< 5**	> 5

Third Breach (During Read)



Our Snapshot Lower Bound

- Fix epoch i
  - For concreteness, let’s say i = 5
- What does adversary need to observe?
  - **Cells last overwritten before epoch 5**
  - **Cells last overwritten in epoch 5**
  - **Cells last overwritten after epoch 5**
- **Test whether the final read probes a cell last overwritten in epoch 5**

Our Snapshot Lower Bound

- Fix epoch i
    - For concreteness, let’s say i = 5
- What does adversary need to observe?
    - **Cells last overwritten before epoch 5**
    - **Cells last overwritten in epoch 5**
    - **Cells last overwritten after epoch 5**
- **Test whether the final read probes a cell last overwritten in epoch 5**

The adversarial strategy for a persistent adversary can be simulated even with a weaker (3,1)-snapshot adversary.

Our Lower Bound Extensions

Theorem. In the cell probe model, any (3,1)-snapshot RAM with any of the following privacy properties must have $\Omega(\log N)$ overhead.

- Differential Privacy
- Read-Only Obliviousness
- Encrypted Search Leakage

Beyond Lower Bounds: Our New Constructions

Theorem. Assuming one-way functions, there exists O(1) overhead constructions for the following primitives:

- (3, 1)-Snapshot No-Write ORAMs
- (∞, 1)-Snapshot Oblivious Stacks/Queues/Deques

Open Problems

1. What is the complexity for (S, 0)-snapshot oblivious RAMs for S>= 2?
2. What is the complexity for (2, L)-snapshot oblivious RAMs for L >= 1?

Questions?

Email: kwlyeo@google.com or giuper@gmail.com

ePrint: ia.cr/2023/811