Practicing Oblivious Access on Cloud Storage: the Gap, Fallacy, and the New Way Forward

Vincent Bindschaedler\textsuperscript{1}, Muhammad Naveed\textsuperscript{1,3}, Xiaorui Pan\textsuperscript{2}, XiaoFeng Wang\textsuperscript{2}, and Yan Huang\textsuperscript{2}

\textsuperscript{1}University of Illinois at Urbana-Champaign
\textsuperscript{2}Indiana University Bloomington
\textsuperscript{3}Cornell University
Cloud Storage

Application

User Side

file1

file2

file3

file4

Cloud Storage Service
Cloud Storage

Application

User Side

Leaks access pattern

Cloud Storage Service

get(file1)
put(file1, data1)
get(file2)
Background: Oblivious RAM

• Obliviousness:
  • For any fixed size request sequence, the associated storages accesses observed (by the cloud) are statistically independent of the requests

• Techniques
  • Operates on fixed size data blocks
  • Encrypt blocks with ciphertext indistinguishability
  • Dummy accesses, re-encryption, shuffling, etc.
Oblivious Cloud Storage

Cloud Storage Service

Application

ORAM Client

Trusted User Side
Oblivious Cloud Storage

Application

ORAM Client

get(key1)

put(key1, val1)

get(key2)

Cloud Storage Service

Amazon Web Services

Google Cloud Storage

Trusted User Side
Oblivious Cloud Storage

Application

get(key1)
put(key1, val1)
get(key2)

ORAM Client

download(object57)
download(object32)
upload(object15, data4)
download(object3)
download(object28)
upload(object65, data19)
download(object11)
download(object44)
upload(object73, data26)

Cloud Storage Service

Amazon Web Services

Google Cloud Storage
How close is ORAM to practice?

• Are ORAM designs in line with the constraints of real-world cloud services?

• How close are ORAM techniques to offering practical support to cloud applications?

• Are we on the right track to narrow the gap?
Assumptions in ORAM literature

1. Bandwidth overhead is a good proxy metric
   • So, minimizing it optimizes application performance

2. Application is not taken into account
   • Implicit assumption that application has no impact on performance

Assumptions influence the way the problem is thought about and guide the research agenda.
Contribution
Contribution

Chose 4 representative ORAM designs
Contribution

1. Build ORAM Systems

Chose 4 representative ORAM designs
Contribution

1. ORAM Literature
   - Chose 4 representative ORAM designs

2. Build ORAM Systems

   ORAM

   Cloud Storage Evaluation Platform

   Performance Data
Contribution

1. ORAM Literature
   Chose 4 representative ORAM designs

2. Build ORAM Systems
   Cloud Storage Evaluation Platform
   ORAM
   Amazon S3
   Performance Data
   App

3. How ORAMs work on cloud storage
   What real apps need
   New understanding
Contribution

1. Chose 4 representative ORAM designs

2. Build ORAM Systems

3. How ORAMs work on cloud storage
   What real apps need
   New understanding

4. CURIOSUS (New ORAM Framework)
ORAM Systems We Built

1. Tree-based: PathORAM
2. Layered-based: LayeredORAM
3. Large messages-based: PracticalOS
4. Partition-based: ObliviStore

Application Selection

• We use Filebench: filesystem benchmarking tool

• Able to emulate several applications, e.g.:
  • Mail server
  • File server
  • Web proxy
  • Web server
Methodology
Methodology

Filebench

client

accesses

Amazon S3 bucket

extract logs

application traces
Findings
Bandwidth overhead as a proxy for response time
Bandwidth overhead as a proxy for response time
Bandwidth overhead as a proxy for monetary cost
Bandwidth overhead as a proxy for monetary cost
Bandwidth overhead as a proxy for monetary cost

![Graph showing Bandwidth overhead as a proxy for monetary cost with different data points for ObliviStore, PathORAM, LayeredORAM, and PracticalOS.]
Application traces

- What metric should be used?
  - Throughput?
  - Response time?

- We propose to use the slowdown:
  - Measures how much an ORAM scheme slows down an application
  - A slowdown of $x$ means the time to replay an application trace on top of ORAM is $x$ times that of without ORAM

- Slowdown := \( \frac{\text{time with ORAM}}{\text{time without ORAM}} \)
Application Traces

• According to slowdown measurements:
  • ObliviStore could easily handle two applications (i.e., varmail and webproxy), but could not handle the other two (i.e., webserver and fileserver)
  • PathORAM could not handle any of the four applications (it experienced slowdowns ranging from 3 to 92)

• In all cases, the monetary cost of running on top of ORAM was roughly 100 times (or more) than running without ORAM
PracticalOS & LayeredORAM

• Neither of the two schemes could support any of the applications

• PracticalOS has a low response time for requests
  • but a long and expensive reshuffling phase

• The cost of operating PracticalOS for varmail is roughly 15 USD / min
Main Findings

• Bandwidth overhead is not the bottleneck
• Network latency is the bottleneck

• Many real applications require the ORAM to process requests concurrently

• Downloads and uploads do not have the same cost
Asynchronicity & Concurrent Request Processing

• ObliviStore can process multiple requests concurrently and offer an asynchronous interface

• Others (e.g., PathORAM) are fundamentally synchronous
  • The current request must be fully completed before the processing of the next request can start

• ORAM schemes do not appear to consider **asynchronicity** as a crucial property
  • 3 out of 39 published papers have this property
Asynchronicity is a MUST!

- Asynchronicity has never been a main design goal.
- But, we found that:
  1. Asynchronicity is not only desirable but actually necessary
     - No synchronous ORAM scheme can fully support cloud applications
  2. Asynchronicity is difficult
     - E.g., the implementation of ObliviStore did not get it right
Bandwidth Asymmetricity

• S3: the monetary cost of an upload is 12.5 times that of a download
Bandwidth Asymmetricity

• S3: the monetary cost of an upload is 12.5 times that of a download
Bandwidth Asymmetricity

- S3: the monetary cost of an upload is 12.5 times that of a download
Bandwidth-only evaluation is INACCURATE!

- **Overhead evaluation**: total bandwidth only in existing literature
  - Bandwidth overhead := download overhead + upload overhead

- **But**, experimentally, their performance and monetary cost are different
  - Failure to incorporate this experimental insight in our thinking could lead us to make incorrect conclusions about how schemes perform in practice
  - Example: which is better?
    - Scheme 1: 20 download overhead, 20 upload overhead
    - Scheme 2: 40 download overhead, 10 upload overhead
CURIOUS
Novel ORAM Framework: CURIOUS

• Based on our findings, we propose CURIOUS

• Simple design:
  • Flexible due to modular design
  • Simple concurrency model

• Also, it preserves properties that applications expect from cloud
  • e.g., reliability
CURIOUS performs better than ObliviStore
CURIOSOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds
CURIOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds
CURIOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds
CURIOUS performs better than ObliviStore

- Monetary cost is only half to two-thirds
- Even though
  - CURIOUS uses 2X the bandwidth of ObliviStore
Conclusions

• Oblivious RAM has come a long way...
• ... and there is a long way to go still...
• But we found:
  • In theory there is no difference between theory and practice
  • But in practice, there is.

• Lesson:
  • align theory to practice
  • evaluate theory on practical systems
Open-Source Code (BSD license)

• Our entire system including CURI OUS, the 4 representative ORAM schemes (PathORAM, LayeredORAM, PracticalOS, ObliviStore), and our evaluation platform is open-source.

• Uses Amazon S3 as storage backend.

• Download URL: oblivious-storage.com

• Contact: bindsch2@illinois.edu