Dog ORAM: A Distributed and Shared Oblivious RAM Model with Server Side Computation.

Alexandre Pujol, *Christina Thorpe*

University College Dublin

UCC 2015 8th IEEE/ACM International Conference on Utility and Cloud Computing

December 7, 2015

Outline of the Presentation

Introduction

Securely outsource data on cloud server:

- Encrypting data is good but not sufficient.
- What about metadata?

From an attacker spying on the server:

- Intercept the metadata and retrieve sensitive knowledge.
- Also works if all the data is encrypted.
- Attacks are cumulative.
- Metadata can reveal more knowledge than data.

Examples

Medical Records: Oncologist access you records $\Rightarrow$ Cancer.

ORAM Definition

Oblivious RAM / Oblivious Storage

Oblivious Random Access Machine

The server cannot determine:

- What data is being accessed,
- How old the data is,
- If the same data has been accessed multiple times,
- The access pattern (sequential, random, etc),
- Whether the access is a read or a write.

Overview

State of the Art

- The obvious Solution:
  - Download, re-encrypt & re-upload all the data
  - Complexity problem
- Newer Solution:
  - Path ORAM [1]: Tree based
  - Onion ORAM [2]: Server side computation
  - Group ORAM [3]: Shared

[1] E. Stefanov & al, "Path ORAM: An Extremely Simple Oblivious RAM Protocol" ACM SIGSAC 2013

[2] S. Devadas & al, "Onion ORAM: A Constant Bandwidth And Constant Client Storage ORAM (Without FHE or SWHE)" 2015

[3] M. Maffei & al, "Privacy And Access Control For Outsourced Personal Records" in 36th IEEE Symposium on Security and Privacy, 2015

Overview

Our goal

- Our goals:
  - Secure, practical cloud storage model
  - Keep "classic" features (shared, encrypted, distributed)
- Our Solution:
  - Dog ORAM: Distributed Onion Group ORAM

Scheme	Bandwidth Cost	Client Storage	Server Computation
Path ORAM [1]	$O(B \log N)$	$O(B\lambda)$	N/A
Group ORAM [2]	$O((B + c) \log N)$	$O(B \log N)$	N/A
Onion ORAM [3]	$O(B)$	$O(B)$	$O(B \log N)$
Dog ORAM	$O(B)$	$O(B)$	$O(B \log N)$

Table: ORAM scheme comparison

Overview
Setup

- Overview:
  - Owner ($\mathcal{O}$) $\Rightarrow$ Trusted,
  - Clients (C_i) $\Rightarrow$ Malicious,
  - Authentication Server (S_{auth}) $\Rightarrow$ *Trusted*,
  - Servers (S_i) $\Rightarrow$ Honest but Curious,
- Security properties:
  - Secrecy,
  - Integrity,
  - Anonymity,
  - Obliviousness,

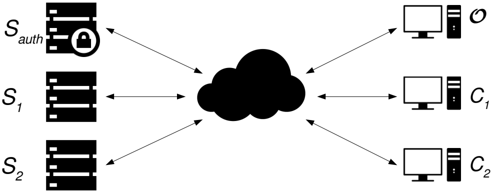
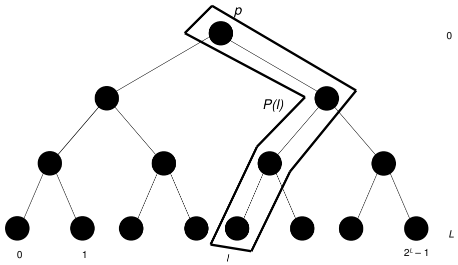


Figure: Dog ORAM: General Setup

Dog ORAM
Server

- Server Structure:
  - Z blocks of data by node,
  - leaf l, virtual address a,
  - $P(l)$: Path from the root to the leaf l.
- Metatada: $PositionMap[a] = l$
- Action (read/write):
  - Oblivious selection of a on $P(l)$,
  - Re-encrypt and write a on the root node,
  - Eviction when the root node is full.



Dog ORAM

Double Homomorphic Selection

- Homomorphic encryption scheme:
  - $\mathcal{E}$ probabilistic additive homomorphic encryption scheme,
  - $\mathcal{E}(x) \oplus \mathcal{E}(y) = \mathcal{E}(x + y)$
  - $\mathcal{E}(x) \odot y = \mathcal{E}(x \cdot y)$

- Selection Vector: b
- Access Right Vector: a_r
- Server does not know:
  - The block address.
  - Client's ID,
  - Client's AR.

$$
\begin{array}{ccc}
b & & block_i \in P(l) \\
\mathcal{E}(\mathcal{E}(0)) & \odot & \mathcal{E}(block_1) \\
 & \oplus & \\
\mathcal{E}(\mathcal{E}(0)) & \odot & \mathcal{E}(block_2) \\
 & \oplus & \\
\mathcal{E}(\mathcal{E}(1)) & \odot & \mathcal{E}(block_j) \\
 & \oplus & \\
\mathcal{E}(\mathcal{E}(0)) & \odot & \mathcal{E}(block_n) \\
\hline\hline
 & \mathcal{E}(\mathcal{E}(block_j)) &
\end{array}
$$

Dog ORAM

Double Homomorphic Selection

- Homomorphic encryption scheme:
  - $\mathcal{E}$ probabilistic additive homomorphic encryption scheme,
  - $\mathcal{E}(x) \oplus \mathcal{E}(y) = \mathcal{E}(x+y)$
  - $\mathcal{E}(x) \odot y = \mathcal{E}(x \cdot y)$

- Selection Vector: b
- Access Right Vector: a_r
- Server does not know:
  - The block address.
  - Client's ID,
  - Client's AR.

$$
\begin{array}{ccccc}
b & & ar & & block_i \in P(l) \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(0))) & \odot & \mathcal{E}(\mathcal{E}(1)) & \odot & \mathcal{E}(block_1) \\
 & & \oplus & & \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(0))) & \odot & \mathcal{E}(\mathcal{E}(0)) & \odot & \mathcal{E}(block_2) \\
 & & \oplus & & \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(1))) & \odot & \mathcal{E}(\mathcal{E}(1)) & \odot & \mathcal{E}(block_j) \\
 & & \oplus & & \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(0))) & \odot & \mathcal{E}(\mathcal{E}(0)) & \odot & \mathcal{E}(block_n) \\
\hline
 & & & & \mathcal{E}(\mathcal{E}(\mathcal{E}(block_j)))
\end{array}
$$

Dog ORAM

Access Right (AR)

Access Right

How to manage the AR?

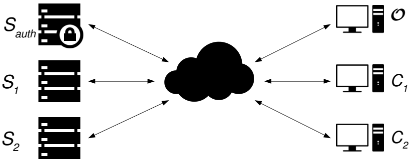
AR requirements:

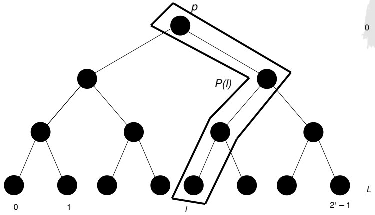
- AR vector must be used only on the blocks of $P(l)$.
- Client must not be able to change/reuse AR.
- Server must not know that a AR is from a client k.
- Only the data owner can change the AR.

Dog ORAM
Access Right (AR)

- Owner signs all the AR.
- Authentication Server:
  1. Stores AR matrix,
  2. Does not manage the AR,
  3. Gives and signs an AR vector corresponding to the block of $P(I)$.
- Prevent AR replay:
  - Why? owner changes AR,
  - Add unique random number to each AR.
- Add unique random number to each AR?
  - The server updates the AR,
  - Using a chameleon signature scheme:
    Hash scheme with a trapdoor.

Conclusion





$$
\begin{array}{ccccc}
b & & ar & & block_i \in P(l) \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(0))) & \odot & \mathcal{E}(\mathcal{E}(1)) & \odot & \mathcal{E}(block_1) \\
 & & \oplus & & \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(0))) & \odot & \mathcal{E}(\mathcal{E}(0)) & \odot & \mathcal{E}(block_2) \\
 & & \oplus & & \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(1))) & \odot & \mathcal{E}(\mathcal{E}(1)) & \odot & \mathcal{E}(block_j) \\
 & & \oplus & & \\
\mathcal{E}(\mathcal{E}(\mathcal{E}(0))) & \odot & \mathcal{E}(\mathcal{E}(0)) & \odot & \mathcal{E}(block_n) \\
\hline\hline
 & & & & \mathcal{E}(\mathcal{E}(\mathcal{E}(block_j)))
\end{array}
$$

Future work:

- Improve Dog ORAM Model:
  - Modes management,
  - Distributed part,
  - Eviction Algorithm,
  - Malicious server,
  - Improve the complexity.
- Implementation & Tests.

THE IRISH SOFTWARE RESEARCH CENTRE

Thank you for your attention

Questions?

Science Foundation Ireland For what's next

Ireland's European Structural and Investment Funds Programmes 2014-2020
Co-funded by the Irish Government and the European Union

European Union
European Regional Development Fund