Information Sciences 447 (2018) 1–11

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

DivORAM: Towards a practical oblivious RAM with variable block size

Zheli Liu[a], Yanyu Huang[a], Jin Li[b,*], Xiaochun Cheng[c], Chao Shen[d]

[a] College of Computer and Control Engineering, Nankai University, Tianjin, China
[b] School of Computer Science, Guangzhou University, Guangzhou, China
[c] Department of Computer Science, Middlesex University, London, UK
[d] School of Electronic and Information Engineering, Xi'an JiaoTong University, Xi'an, China

ARTICLE INFO

Article history:
Received 30 November 2017
Revised 21 February 2018
Accepted 26 February 2018
Available online 6 March 2018

Keywords:
Oblivious RAM
Data privacy
Cloud computing
Access pattern

ABSTRACT

Oblivious RAM (ORAM) is important for applications that require hiding access patterns. Many ORAM schemes have been proposed but most of them support only storing blocks of the same size. For the variable length data blocks, they usually fill them upto the same length before uploading, which leads to an increase in storage space and network bandwidth usage. To develop the first practical ORAM with variable block size, we proposed the "DivORAM" by remodeling the tree-based ORAM structure. It employs an additively homomorphic encryption scheme (Damgård–Jurik cryptosystem) executing at the server side to save the client computing overhead and the network bandwidth cost. As a result, it saves network bandwidth 30% comparing with Ring ORAM and 40% comparing with HIRB ORAM. Experiment results show that the response time of DivORAM is 10 × improved over Ring ORAM for practical parameters.

1. Introduction

With the increasing popularity of cloud computing, more and more personal and commercial applications are deployed in clouds to provide services. While it reduces the cost of software delivery and maintenance, users still hesitate to use these cloud applications due to privacy concerns. Weak authentication, dishonest employee, and other attacks all lead to data leakage and further reduce the confidence of customers [3,12–15,17–19,23]. Recently, lots of schemes [1,2,9,11,24,28,29] have been proposed for protecting data privacy by uploading the encrypted data to the cloud server.

However, encryption along is not enough for protecting the data privacy. The attacks caused by the leakage of data access patterns have gradually gained more and more attention. In the latest information service architecture, more and more applications store a great quantity of private data in outsourced devices or servers. Therefore, for an honest and curious server, it can analyze the access pattern to get some important privacy information that could potentially lead to privacy leaks without revealing the plaintext. This may lead to serious consequences. For example, a hospital stores privacy data such as patient information in an outsourced server. When the doctor accesses the data, it is reasonable for the server to estimate that the patient information accessed by the oncologist is mostly a cancer patient. Therefore, the server can recognize which category the stored data belongs to. In the meantime, when the same patient information is accessed more

* Corresponding author.
E-mail addresses: liuzheli@nankai.edu.cn (Z. Liu), onlyerir@163.com (Y. Huang), jinli71@gmail.com (J. Li), X.Cheng@mdx.ac.uk (X. Cheng), cshen@sei.xjtu.edu.cn (C. Shen).

https://doi.org/10.1016/j.ins.2018.02.071
0020-0255/

Table 1
Comparison with typical ORAMs with tree-based and variable block-size ORAM scheme. Server storage in all schemes can be set to $O(BN)$, B is the size of one data blocks, N is the maximum number of blocks stored in ORAM schemes.

Scheme	Block size	Bandwidth	Computation	Client Storage
Ring ORAM	64 bytes	$O(\log N)$	-	$n(T)\cdot A/2$
HIRB ORAM	$(20\|\tau\|+R)B$	$O(\log N)$	-	$\widetilde{O}(\log N)$
DivORAM	$O(\log^5 N)$	$O(1)$	$O(\log^3 N)$	$O(\log N)$

frequently, the server may deduce that the patient's medical information is given a high priority, which is likely to be a critically ill patient, thereby disclosing the patient's private information.

Oblivious RAM (ORAM). ORAM [8] is a general cryptographic primitive which allows sensitive data accessed obliviously. The purpose of ORAM is hiding access patterns, through re-encrypting each data and confusing the storage location of data in every access. After the concept of ORAM was proposed, many ORAM mechanisms [10] emerged. These mechanisms can be roughly classified into two types depending on the storage structure, layer-based ORAM [20] where data is stored in several levels and tree-based ORAM [7,25] where the storage construction is a binary tree. Meanwhile, more and more ORAM protocol are combined secure multi-party computation [6,26] or oblivious databases [22] for better performance.

1.1. The vORAM and challenges

Few works focuses on the variable block-size ORAM (vORAM) like [22]. The simplest way to solve the problem of variable block size is to fill all the blocks to the same length. Apparently, this method will make the server store a lot of invalid information. In the worst case, large storage space in the server is wasted to store meaningless information when small size data accounts for the majority of all data. In practice, it makes the entire ORAM impractical while causing redundancy in storage space.

In order to solve the above problem, we try to split different size of data into fragments of the same size. Like tree-based ORAM, we choose a binary tree as the storage construction as well. Each node in the tree has several slots to store a fragment as a splinter.

Challenges. When blocks are divided into several splinters, we should use the tree's storage structure as reasonably as possible to store splinters that belong to the same block on the same path. How to store splinter without wasting storage space on the server has become a problem we need to solve. Meanwhile, how to read and write splinters that belong to the same block efficiently and at a fraction of the cost is one challenge we had when designing a solution. In previous tree-based ORAM scheme like [21], the eviction will cause heavy communication overhead between the client and the server. In eviction process, the server must send a lot of block to the client to re-encrypt and permutate the position of these blocks. In that way, the server knows nothing about whether or not to access the same block. We also hope to find a solution to reduce the computational and communication overheads of shuffle operations in the client.

1.2. Our contribution

In this paper, we proposed DivORAM to deal with the problem caused by variable block size. It is worth noting that we have redesigned the tree-based ORAM to rationalize the operation process. We adopt the Private Information Retrieve (PIR) technique and set up a trusted proxy so that the execution time and network bandwidth can be feasible for the practical application. We compared bandwidth overhead with Ring ORAM [21] and HIRB ORAM [22] in Table 1. The main contribution is as follows:

Optimize server storage. We redesign the structure of the tree with different bucket sizes instead of previous solutions with same bucket size. In addition, we "cut" a block to several splinters so that no longer need to pad the block, to save server-side storage. In our new tree structure, each node stores several splinters and no longer stores a block. Every splinter which belongs to the same block will spread among the same path. In order for the upper level nodes to have enough space to temporarily store splinters which have not been shuffled, we define the size of the parent node to be twice the size of the child node.

Constant complexity communication. In DivORAM protocol, we achieve constant complexity bandwidth consumption in a complete access process. We adopt PIR to minimize the bandwidth during the read operation when we set the block size to $O(\log^5 N)$. The specific analysis is described in Section 5.1. Meanwhile, we transfer the evict operation to a trusted proxy so that the write operation was simplified. In this paper, we pay attention to the communication overhead between the client and the server. Therefore, write operation uses constant complexity bandwidth. Since eviction does not involve clients, we do not take into account the bandwidth consumption of eviction.

Lightweight client load. The client undertakes little work during the whole visit. Compared to previous ORAM scheme, the client in DivORAM is no longer involved in shuffle work, and only need a small amount of calculation.