ARC'18

مؤتمر مؤسسة قطر
السنوي للبحوث

QATAR FOUNDATION
ANNUAL RESEARCH CONFERENCE

البحث والتطوير: التركيز على
الأولويات، وإحداث الأثر

R&D: FOCUSING ON PRIORITIES,
DELIVERING IMPACT

20-19 مـــارس
19-20 MARCH

مؤسسة قطر
Qatar Foundation
لإطلاق قدرات الإنسان.
Unlocking human potential.

Computing & Information Technology - Paper Presentation

http://doi.org/10.5339/qfarc.2018.ICTPP277

Implementing and Analyzing a Recursive Technique for Building Path Oblivious RAM

Maan Haj Rachid*, Ryan Riley, Qutaibah Malluhi

Department of Computer Science and Engineering, Qatar University
* maanhajrachid@hotmail.com

It has been demonstrated that encrypting confidential data before storing it is not sufficient because data access patterns can leak significant information about the data itself (Goldreich &amp; Ostrovsky, 1996). Oblivious RAM (ORAM) schemes exist in order to protect the access pattern of data in a data-store. Under an ORAM algorithm, a client accesses a data store in such a way that does not reveal which item it is interested in. This is typically accomplished by accessing multiple items each access and periodically reshuffling some, or all, of the data on the data-store. One critical limitation of ORAM techniques is the need to have large storage capacity on the client, which is typically a weak device. In this work, we utilize an ORAM technique that adapts itself to working with clients having very limited storage. A trivial implementation for an oblivious RAM scans the entire memory for each actual memory access. This scheme is called linear ORAM. Goldreich and Ostrovsky (Goldreich &amp; Ostrovsky, 1996) presented two ORAM constructions with a hierarchical layered structure: the first, Square-root ORAM, provides square root access complexity and constant space requirement; the second, Hierarchical ORAM, requires logarithmic space and polylogarithmic access complexity. Square-root ORAM was revisited by (Zahur, et al.) to improve its performance in a multi-party secure computation setting. The work of Shi et al. (Shi, Chan, Stefanov, &amp; Li, 2011) adopted a sequence of binary trees as the underlying structure. (Stefanov, et al., 2013) utilized this concept to build a simple ORAM called path ORAM. In path ORAM, every block (item) of data in the input array A is mapped to a (uniformly) random leaf in a tree (typically a binary tree) on the server. This is done using a position map stored in the client memory. Each node in

دار جامعة حمد بن خليفة للنشر
HAMAD BIN KHALIFA UNIVERSITY PRESS

Cite this article as: Rachid M et al. (2018). Implementing and Analyzing a Recursive Technique for Building Path Oblivious RAM. Qatar Foundation Annual Research Conference Proceedings 2018: ICTPP277
http://doi.org/10.5339/qfarc.2018.ICTPP277.

the tree has exactly Z blocks which are initially dummy blocks. Each data item is stored in a node on the path extending from the leaf to which the data item was mapped, to the root. When a specific item is requested, a position map is used to point out the leaf to which the block is mapped. Then the whole path is read starting from the block's mapped leaf up to the root into a stash. We call this procedure a read path method. The stash is a space which also exists on the client. The block is mapped to a new leaf (uniformly random). The client gets the required block from the stash. We then try to evict the contents of the stash to the same path which we read from starting from the leaf to the root. We call this procedure a write path method. To transfer a block into a node in the tree: - The node that is tested should have enough space. - The node should be on the path to which the tested block in the stash is mapped. If both conditions are met, a block is evicted to the tested node. Clearly, the tree is encrypted and whenever the client reads a path into the stash, all read blocks are decrypted and the requested block is sent to the client. The client encrypts the blocks before writing them back to the path. The security proof of this ORAM type is explained in (Stefanov, et al., 2013). We assume that a position map can be fit in the client's memory. Since it requires O(N) space, that could be a problem. (Stefanov, et al., 2013) mentioned a general idea for a solution that uses another smaller ORAM O1 on the server to store the position map and stores the position map for O1 in the client's memory. We employ a recursive generalized version of this approach. If the position map is still larger than the client's capacity, a smaller ORAM O2 is built to store the position map for O1. We call these additional trees auxiliary trees. We implemented the recursive technique for Path ORAM and studied the effect of the threshold size of position map (and consequently, the number of auxiliary trees) on the performance of path ORAM. We tested our implementation on 1 million items using several threshold sizes of position map. The number of accesses is 10,000 in all tests. Our results show expected negative correlation between the time consumption and the threshold size of the position map. However, the results suggest that unless the increase in the threshold size of position map decreases the number of trees, no significant improvement in performance will be noticed. It is also clear that the initialization process which is the process of building the items› tree and the auxiliary trees and filling the initial values comprises more than 98% of the consumed time. Accordingly, this type of ORAM suits the case of large number of accesses since the server can fulfil client›s request very fast after finishing the initialization process.

References   Goldreich, O., &amp; Ostrovsky, R. (1996). Software protection and simulation on oblivious rams. Journal of the ACM (JACM), vol. 43, no. 3, pp. 431–473. Shi, E., Chan, T.-H., Stefanov, E., &amp; Li, M. (2011). Oblivious ram with o ((logn) 3) worst-case cost. International Conference on The Theory and Application of Cryptology and Information Security. Springer, pp. 197–214. Stefanov, E., Dijk, M. V., Shi, E., Fletcher, C., Ren, L., Yu, X., et al. (2013). Path oram: an extremely simple oblivious ram protocol. Proceedings of the 2013 ACM SIGSAC conference on Computer &amp; communications security. ACM, pp. 299–310. . Zahur, S., Wang, X., Raykova, M., Gascon, A., Doerner, J., Evans, D., et al. (n.d.). Revisiting square-root oram: Efficient random access in multi-party computation. 2016: Security and Privacy (SP), IEEE Symposium on. IEEE, pp. 218–234.