STM JOURNALS

Journal of Advances in Shell Programming

Volume 1, Issue 3
www.stmjournals.com

Secure Two Party Computation

Ankush Rai*
Department of Applied Science, CRIAD Laboratories, Smiriti Nagar, Bhilai-490020, India

Abstract

The author investigated the flaws with two-party computation over a cloud or parallel services for an encrypted data which can be easily accessed by any efficient shell programmer. Generally, the computation with the slaves and the master enables the store-to-store data locally, even that may be a constant amount of data and thereby we have found that the third parties can remotely access the n amount of data items, while ensuring that the identities of the items are kept hidden. Hence, in this short communication the author presented a patched protocol for hidden scalable computational environment to safeguard its computing data from outside interventions.

Keywords: *Secure computational communication, encryption protocol*

****Author for Correspondence*** E-mail: ankushressci@gmail.com

INTRODUCTION

There have been various attempts for encryption of data storage already made and for the most part this field has been saturated for years [1–5]. Also, various encryption algorithm have been in common use for content encryption but apparently such methods limit the encryption advantage to only to and fro of data [6–8]. But for environment like that of searchable encryption in computing environment like that of cloud computing or parallel computing or even distributed computing the major setback that we have discovered is the easy accessibility of the computing data in the RAM using a clever shell scripting to determine the data or service usages that shall prove harmful to the other applications and individuals dependent over it. For example, whether data may or may not be encrypted but the service made in the server by the client will ultimately be used as monitoring or tackling the details such as based on app usage the web apps running on server can classify individuals based on their habits position, etc. Now, if one can configure the shell of the client's source to manipulate or extract the data held for processing in the RAM of the server or even of client can be extracted to use for malicious purposes [9–13]. This is where the encryption in computation is required which had been completely left off in the previous studies [12–14]. As the handheld embedded systems are increasing in popularity, the major route to increase their computing power is restricted to sharing off hardware devices over a network but in due process we forget that no matter what algorithm it shall be used to encrypt but the processor or the process in the computation is left alone for the third parties to intervene easily.

Now, the follow-up of the problem lies in the fact that if we use each thread to encrypt, the memory will overflow and thus will hinder computation. Therefore, to avoid this case scenario and also to use a methodology to provide security to the shared computation, we work on a real-world scenario of shared computing specifically of that of stock-exchange action in the following steps:

Consider that for a sequence of queries namely, q_1, q_2, $q_3 \ldots q_n$, is meant to follow by an action of the stock-exchange, thereby the curious server can easily have the capacity to learn about the content of queries, irrespective of the fact of being it encrypted. More importantly it can predict the action of user's preferences with the sequence of queries appear over and over again in a familiar pattern by counting the frequency of threads processing to be threaded over RAM accessing the same data items. Thus, the server at its adversary can only prosecute the resources in an attempt for decrypting only those data items

which are targeted excessively by the targeted user, which adds an ability to the server to frame a relation between user and its queries to the server. Thus, for encryption of such data leaks, we can follow certain steps as follows:
Initially, the data structure consists of data requests kept empty. For each request (of any type or process) of a register index (virtual address), the following operations are required to perform:

1. Scan through the entire first level in a sequential order to find the item whose register-index. This step includes reading all the items in the first level. If the requested item is found, it is stored in the client's secure memory, and the process continues as usual.
2. For each level of operation, i = 2: N, do: examine its two possible register index in the hashing of the cache memory table of the current level. Frame a dummy loop out of hashing table to rearrange the memory index in the register units by refreshing dummy table from the dummy loop. This requires almost no computational expense to perform so and no sorting algorithm will intervene in that environment.
3. Initiate scanning again all the way through entire first level in a sequential order, and write back the updated item of register-index in the next available memory units.

CONCLUSIONS

This framework confuses the server to carry out queries and understand the pattern without adding or carrying out other computational load, and is easily implementable all other known constructions of RAM even the oblivious ones. The data flows followed by reshuffle of request in several levels does not has any overheads than the one which is requested and if any operation or attempts to made to finding the pattern it will putt the memory to a overload and lead the system to crash. Thereby, providing a fully secured privacy without any additional computation than the ongoing one. The author hopes that his work in this short communication will in future provide a secure privacy of online services especially those involved with computing.

REFERENCES

1. *Amazon Simple Storage Service* (Amazon S3). http://aws.amazon.com/s3/.
2. Ajtai M. Oblivious RAMs without cryptographic assumptions. *STOC*. 2010.
3. Ajtai M, Kolm´os J, Szemer´edi E. An O(n log n) sorting network. In *STOC*. 1983; 1-9p.
4. Arbitman Y, Naor M, Segev G. De-amortized Cuckoo hashing: Provable worst-case performance and experimental results. In *ICALP* (1); 2009; 107–18p.
5. Y Arbitman Y, Naor M, Segev G. Backyard Cuckoo hashing: Constant worst-case operations with a succinct representation. *Manuscript*. 2010.
6. Batcher K. Sorting networks and their applications. In *AFIPS Spring Joint Computing Conference*. 1968; 32: 307–14p.
7. Benes VE. Optimal rearrangeable multistage connecting networks. *Bell System Technical Journal*. 1964; 4.
8. Bloom BH. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*. 1970; 13(7): 422–6p.
9. Cormen T, Leiserson C, Rivest R. *Introduction to Algorithms.* McGraw Hill and The MIT Press. 1990.
10. Damgard I, Meldgaard S, Nielsen JB. Perfectly secure oblivious RAM without random oracles. *Cryptology ePrint Archive*. Report 2010/108, 2010. http://eprint.iacr.org/2010/108.
11. Fan L, Cao P, Almeida J, et al. Summary cache: A scalable wide-area web cache sharing protocol. *IEEE/ACM Transactions on Networking* (TON). 2000; 8(3): 293p.
12. Goldreich O. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC*, ACM, 1987; 182–94p.
13. Iliev A, Smith SW. Private information storage with logarithm-space secure hardware. In *International Information Security Workshops*. 2004; 199–214p.
14. Iliev A, Smith SW. Protecting client privacy with trusted computing at the server. *IEEE Security & Privacy*. 2005; 3(2): 20–8p.