A Little Storage Goes a Long Way: Accelerating Private Information Retrieval

Peter Williams
Stony Brook Network Security and Applied Cryptography Lab
Stony Brook, NY 11794-4400
petertw@cs.sunysb.edu

Radu Sion
Stony Brook Network Security and Applied Cryptography Lab
Stony Brook, NY 11794-4400
sion@cs.sunysb.edu

Miroslava Sotáková
Stony Brook Network Security and Applied Cryptography Lab
Stony Brook, NY 11794-4400
gwhitehawk@gmail.com

ABSTRACT

The authors have previously shown that existing single-server computational private information retrieval (PIR) protocols *for the purpose of preserving client access patterns leakage* are orders of magnitude slower than trivially transferring the entire data sets to the inquiring clients. We thus raised the issue of designing efficient PIR mechanisms in practical settings.

We introduce exactly such a technique, guaranteeing access pattern privacy against a computationally bounded adversary, in outsourced data storage, with communication and computation overheads orders of magnitude better than existing approaches. In the presence of a small amount ($O(\sqrt{n}\sqrt{\log n}\sqrt{\log\log n})$, where n is the size of the database) of temporary storage, clients can achieve access pattern privacy with computational complexities of less than $O(log^2 n)$ per query (as compared to e.g., $O(log^4 n)$ for existing approaches).

We achieve these novel results by applying new insights based on probabilistic analyses of data shuffling algorithms to Oblivious RAM, allowing us to significantly improve its asymptotic complexity. This results in a protocol crossing the boundary between theory and practice and becoming generally applicable for access pattern privacy. We show that on off the shelf hardware, large data sets can be queried obliviously orders of magnitude faster than in existing work.

1. INTRODUCTION

In an increasingly networked world, computing and storage services require security assurances against malicious attacks or faulty behavior. As networked storage architectures become prevalent – e.g., networked file systems and online relational databases in sensitive public and commercial infrastructures such as email and storage portals, libraries, health and financial networks – protecting the confidentiality and integrity of *stored* data is paramount to ensure safe computing. In networked storage, data is often geographically distributed, stored on potentially vulnerable remote servers or transferred across untrusted networks; this adds security vulnerabilities compared to direct-access storage.

Moreover, today, sensitive data is being stored on remote servers maintained by third party storage vendors. This is because the total cost of storage management is 5–10 times higher than the initial acquisition costs [2]. Moreover, most third party storage vendors do not provide strong assurances of data confidentiality and integrity. For example, personal emails and confidential files are being stored on third party servers such as FilesAnywhere.com, Gmail, Yahoo! Mail, MSN Hotmail [5].

Privacy guarantees of such services are at best declarative and often subject customers to unreasonable fine-print clauses— e.g., allowing the server operator (and thus malicious attackers gaining access to its systems) to use customer behavior for commercial profiling, or governmental surveillance purposes [7].

To protect data stored in such an untrusted server model, security systems should offer users assurances of data confidentiality and access pattern privacy. However, a large class of existing solutions delegate this by assuming the existence of co-operating, non-malicious servers. As a first line of defense, to ensure confidentiality, all data and associated meta-data can be encrypted at the client side using non-malleable encryption, before being stored on the server. The data remains encrypted throughout its lifetime on the server and is decrypted by the client upon retrieval.

Encryption provides important privacy guarantees at low cost. It however, is only a first step as significant amounts of information are still leaked through the access pattern of encrypted data. For example, consider an adversarial storage provider that is able to determine a particular region of the encrypted database corresponds to an alphabetically sorted keyword index. This is not unreasonable, especially if the adversary has any knowledge of the client-side software logic. The adversary can then correlate keywords to documents by observing which locations in the encrypted keyword index are updated when a new encrypted document is uploaded.

In existing work, one proposed approach for achieving access pattern privacy is embodied in Private Information Retrieval (PIR) [1]. PIR protocols aim to allow clients to re-

trieve information from public or private databases, without revealing to the database servers which record is retrieved. Recently however, we showed [8] that deployment of existing non-trivial single server PIR protocols on real hardware of the recent past would have been orders of magnitude more time-consuming than trivially transferring the entire database. Their deployment would in fact *increase* overall execution time, as well as the probability of *forward* leakage, when the present trapdoors become eventually vulnerable – e.g., today's queries will be revealed once factoring of today's values will become possible in the future. We stressed that this result goes beyond existing knowledge of mere "impracticality" under unfavorable assumptions. On real hardware, *no* existing non-trivial single server PIR protocol could have possibly had out-performed the trivial client-to-server transfer of records in the past, and is likely not to do so in the future either. This negative result is due to the fact that on any known past general-purpose Von Neumann hardware, it is simply more expensive to PIR-process one bit of information than to transfer it over a network.

A related line of research tackles client-privacy of accesses to client-originated data on a server. Specifically, the server hosts information for a client, yet does not find out which items are accessed. Note that in this setup the client has full control and ownership over the data and other parties are able to access the same data through this client only. One prominent instance of such mechanisms is Oblivious RAM (ORAM) [4]. For simplicity, in the following we will use the term ORAM to refer to any such outsourced data technique.

The recent advent of tamper-resistant, general-purpose trustworthy hardware (such as the IBM 4764 Secure Co-Processor [6]) opens the door to effectively deploying ORAM primitives for PIR purposes (i.e., for arbitrary public or private data, not necessarily originated by the current client) by deploying such hardware as a trusted server-side client proxy. The SCPU offers complete tamper detection, as well as remote code attestation to prove to clients that a particular program is in fact running unmodified on such a SCPU. However, trusted hardware devices are not a panacea. Their practical limitations pose a set of significant challenges in achieving sound regulatory-compliance assurances. Specifically, heat dissipation concerns under tamper-resistance requirements limit the maximum allowable spatial gate-density. As a result, general-purpose secure coprocessors (SCPUs) are often significantly constrained in both computation ability and memory capacity, being up to one order of magnitude slower than host CPUs.

In this paper we first introduce an efficient ORAM protocol with significantly reduced computation complexity ($O(log^2 n)$ vs. $O(log^4 n)$ for [4]) – suited for deployment on constrained hardware such as SCPUs. We propose its deployment on existing secure hardware (IBM 4764 [6]) and show that the achievable throughputs are practical and orders of magnitude higher than existing work. These results constitute a first step to making PIR assurance truly practical.

2. MODEL

In our discourse, we will consider the following concise yet representative interaction model. Sensitive data is placed by a client on a data server. Later, the client or a third party will access the outsourced data through an online query interface exposed by the server. Network layer confidentiality is assured by mechanisms such as SSL/IPSec. Without sacrificing generality, we will assume that the data is composed of equal-sized blocks (e.g., disk blocks, or database rows).

Clients need to read and write the stored data blocks while revealing a minimal amount of information (preferably none) to the server. We will describe the protocols here from the perspective of the client who will implement to primitives: read(id), and write(id, $newvalue$). Specifically, the (untrusted) server need not be aware of the protocol, but rather just provide traditional store/retrieve primitives (e.g., a file server interface).

Timing attacks have the potential to leak some information during the operation of this algorithm, if the algorithm is not implemented properly to avoid these attacks. We assume that the implementation is built in a manner that does not leak any secret information (such as the location of the fake blocks) through timing, noting that (1) any implementation can be turned into a timing-attack-free implementation simply by waiting longer on paths determined by secret information so that the length of all paths matches the length of the longest path, and (2) the transformation in (1) can be achieved without affecting the running time complexity of this algorithm.

Adversary. The adversarial setting considered in this paper assumes a server that is *curious but not malicious*. While it desires to illicitly gain information about the stored data, it nevertheless executes all queries in a correct manner. We are not concerned here with denial of service behavior. We also assume the adversary can be represented by a polynomial-time Turing machine; i.e., it is computationally bounded, thereby allowing us to take advantage of the following cryptographic primitives.

Cryptography. We require three cryptographic primitives with all the associated semantic security [3] properties: (i) a secure hash function which builds a distribution from its input that is indistinguishable from a uniform random distribution, (ii) an encryption function that generates unique ciphertexts over multiple encryptions of the same item, such that a computationally bounded adversary has no non-negligible advantage at determining whether a pair of encrypted items of the same length represent the same or unique items, and (iii) a pseudo random number generator whose output is indistinguishable from a uniform random distribution over the output space.

3. REFERENCES

[1] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan. Private information retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[2] Gartner, Inc. Server Storage and RAID Worldwide. Technical report, Gartner Group/Dataquest, 1999. `www.gartner.com`.

[3] O. Goldreich. *Foundations of Cryptography*. Cambridge University Press, 2001.

[4] Oded Goldreich and Rafail Ostrovsky. Software protection and simulation on oblivious ram. *Journal of the ACM*, 45:431–473, May 1996.

[5] Mathias Hild and Jordan Mitchell. Free Email: Google, MSN Hotmail and Yahoo! (A). *SSRN eLibrary*, 2004.

[6] IBM Corp. IBM 4764 Model 001 Specification Sheet. Online at `http://www-03.ibm.com/security/cryptocards/pdfs/4764-001_PCIX_Data_Sheet.pdf`, 2008.

[7] Curtis Scribner. Comment and casenote: Subpoena to Google Inc. in ACLU v. Gonzales: "big brother" is watching your internet searches through government subpoenas. *University of Cincinnati Law Review*, 75(1273), 2007.

[8] Radu Sion and Bogdan Carbunar. On the Practicality of Private Information Retrieval. In *Proceedings of the Network and Distributed Systems Security Symposium*, 2007. Stony Brook Network Security and Applied Cryptography Lab Tech Report 2006-06.