POSTER: DataLair: A Storage Block Device with Plausible Deniability

Anrin Chakraborti
Stony Brook University
anchakrabort@cs.stonybrook.edu

Chen Chen
Stony Brook University
chen18@cs.stonybrook.edu

Radu Sion
Stony Brook University
sion@cs.stonybrook.edu

Abstract

Sensitive information is present on our phones, disks, watches and computers. Its protection is essential. Plausible deniability of stored data allows individuals to deny that their device contains a piece of sensitive information. This constitutes a key tool in the fight against oppressive governments and censorship.

Unfortunately, existing solutions, such as the now defunct TrueCrypt [2], can defend only against an adversary that can access a user's device at most once ("single-snapshot adversary"). Recent solutions have traded significant performance overheads for the ability to handle more powerful adversaries able to access the device at multiple points in time ("multi-snapshot adversary").

In this paper we show that this sacrifice is not necessary. We introduce and build DataLair, a practical plausible deniability mechanism. When compared with existing approaches, DataLair is two orders of magnitude faster (and as efficient as the underlying raw storage) for public data accesses, and 3-5 times faster for hidden data accesses.

An important component in DataLair is a new, efficient write-only ORAM construction, which provides an improved access complexity when compared to the state-of-the-art.

1. INTRODUCTION

With increasing amounts of sensitive data being stored, encryption has become a necessity. Although full disk encryption provides security against unauthorized adversaries attempting to access sensitive data at rest, it does not allow denial of the possession of sensitive data. This is a serious challenge in the presence of oppressive regimes and other powerful adversaries that may want to coerce the user into revealing encryption keys. This unfortunately is an all-too-common occurrence and there are numerous examples [7] where sensitive data in possession of human rights activists have been subject to government scrutiny in oppressive regimes, thus endangering the witnesses.

Plausible deniability (PD) provides a strong defense against such coercion. A system with PD allows its users to deny the existence of stored sensitive information or the occurrence of a sensitive transaction[6].

An example of a plausibly deniable storage solution is the successful, yet unfortunately now-defunct TrueCrypt [2]. TrueCrypt divides a disk into multiple "password-protected" volumes and allows some of these volumes to be "hidden" and store sensitive data. Password-derived encryption keys are used to encrypt each such volume. Upon coercion, a user can plausibly deny the existence of a hidden volume by simply providing a valid password for one of the non-hidden ones, thus showing a plausible use for the disk without revealing the hidden volume data. TrueCrypt stores hidden volumes in the free space of non-hidden (public) volumes. To mask their existence, TrueCrypt fills all free space with random data and encrypts the hidden data with a randomized semantically secure encryption scheme with output indistinguishable from random.

However, as pointed out by Czeskis [4], TrueCrypt is not secure against an adversary that can access the disk at multiple points in time (e.g., multiple security checks or border crossings). In such scenarios, an adversary can save a snapshot and compare subsequent snapshots with it. Any changes in the free space occurring between snapshots will suggest the existence of hidden data.

A major reason why TrueCrypt fails to provide PD against an adversary with multiple snapshots is because it does not attempt to hide *access patterns*. The adversary can point out exact locations on disk that have changed in between snapshots and notice the apparently free portion of the disk (potentially containing hidden data) appears altered.

To defeat a multi-snapshot adversary we need to eliminate any evidence of hidden data and corresponding accesses by ensuring any modifications are attributable and indistinguishable from the traces of public data operations.

This means that modifications to apparently free space should be part of normal behavior of plausible public operations and the traces of hidden data accesses should be indistinguishable from the traces of public data accesses.

One effective way to achieve that is to "cloak" hidden accesses within public data accesses by always performing a public operation for every hidden operation. Further, oblivious access mechanisms (ORAM) can be used for randomizing accesses and making them indistinguishable [3]. Unfortunately, ORAMs come with very high overheads and reduce overall throughput by orders of magnitude.

Fortunately, a new insight emerges that enables a significant throughput increase: accesses to public data do not

CCS'16 October 24-28, 2016, Vienna, Austria

ACM ISBN 978-1-4503-4139-4/16/10.
DOI: http://dx.doi.org/10.1145/2976749.2989061

need to be hidden since since the existence of public data is admitted. In fact, revealing access patterns to public data reinforces deniability since it shows non-hidden disk use to a curious adversary.

Consequently, DataLair uses this insight to design a significantly more efficient way to achieve strong PD: protecting only operations on the hidden data, while ensuring that they are indistinguishable from operations on public data (thus allowing the user to claim all I/O as being due to public accesses). Further, DataLair also optimizes the oblivious access mechanism deployed for hidden data.

In summary, public data is accessed (almost) directly without the need to use an oblivious access mechanism while hidden data accesses are mitigated through a new throughput-optimized write-only ORAM which significantly reduces access complexity when compared to existing work [3, 5]. As a result, DataLair is two orders of magnitude faster (and as efficient as the underlying raw storage) for public data accesses, and 3-5 times faster for hidden data accesses, when compared to existing work.

2. PROPOSED SOLUTION

In the following we detail a brief intuition behind DL-ORAM and show how it can be used to achieve efficient multi-snapshot plausible deniability in DataLair.

2.1 DL-ORAM

The most bandwidth-efficient write-only ORAM is the construction by Blass *et al.* [3] (further referred to as HIVE-ORAM). HIVE-ORAM [3] maps data from a logical volume uniformly randomly to the physical blocks on the underlying device. The position map for the ORAM is recursively stored in $O(logN)$ smaller ORAMs, a standard technique introduced in [8]. The recursive technique reduces communication complexity by storing the position map in logical blocks of smaller size. This allows the ORAM to access the position map with $O(logN)$ logical block accesses. Under the assumption of non-uniform logical block sizes, HIVE-ORAM [3] accesses a constant number of logical blocks per ORAM operation at the cost of some overflow that is stored in an in-memory stash. To keep the stash bounded to a constant size, half of the ORAM needs to be empty.

However, on local storage devices (which is the primary target for plausible deniability), practical complexity needs to be measured in terms of the number of physical blocks (sectors) accessed per I/O, as accessing a logical block entails access to the corresponding physical block. In this case, HIVE-ORAM [3] has a block read complexity (number of blocks read) of $O(log_\beta N)$ and a block write complexity (number of blocks written) of $O(log_\beta^2 N)$ where $\beta = B/addr$, B is the physical block size in bytes and $addr$ is the size of one physical block address in bytes

DL-ORAM also maps data from logical blocks to random physical blocks, but fundamentally differs from HIVE-ORAM [1] as a result of two new techniques for reducing write access complexity – (i) instead of a recursive construction, the position map is stored obliviously in a B+ tree within the same ORAM as the data, (ii) an $O(1)$ free block finding scheme that can select uniformly random blocks out of all currently free blocks on the device. Storing the map in a B+tree (in the same ORAM as the data) ensures an equivalent read complexity of $O(log_\beta N)$ as storing the map recursively (in $O(log_\beta N)$ ORAMs), while also allowing better space utilization. For (ii), DL-ORAM stores the IDs of all currently free blocks in an auxiliary data structure that is accessed obliviously to select uniformly random free block IDs. Due to the $O(1)$ free block finding scheme, updating $O(log_\beta N)$ nodes of the map B+tree has an overall complexity of $O(log_\beta N)$ instead of $O(log_\beta^2 N)$ as in case of HIVE-ORAM [3].

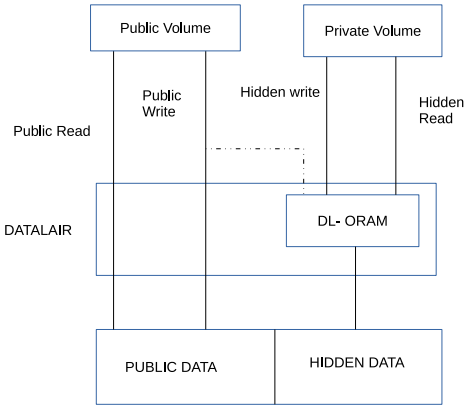

Figure 1: Basic Design for DataLair. Public data is written to the public partition. Writes to the private partition are through DL-ORAM which ensures write-access privacy. Reads for the private volume are not protected since they are not observable to the adversary.

2.2 DataLair Design

The basic design for DataLair creates two partitions on the device as per user configuration. The user can then use one partition for storing public data (data that can be revealed to an adversary) and the other partition for storing private data (data that is hidden from the adversary). Each partition is encrypted with a key – the key for the public partition is disclosed to an adversary upon coercion while the key for the private partition is always hidden. Note that the reads for private data does not need to be protected in DataLair since they are not observable to the adversary.

Figure 1 shows the basic design for DataLair. Data from a logical public volume is written straightforwardly to the public partition without any internal re-mappings. The private partition deploys DL-ORAM for write-access privacy. Each write to the public partition also perform a write to the private partition. If there is an outstanding private write, then it is performed through DL-ORAM with the public write. Otherwise, DataLair simulates an access to DL-ORAM. Even in the case when the private partition is not being used for storing data, DataLair simulates an access to the private partition for each public write. Thus, a user can deny using the private partition and attribute any changes to the private partition as a result of the simulated access. It can be shown that the adversary cannot detect whether the private partition is being used with more than non-negligible advantage over pure guessing. We omit details in this poster.

Merging Volumes. The final DataLair design maps all

user created logical volumes to the same physical partition. Thus, in this case both public and hidden data co-exist on the same physical partition, which achieves better utilization of space. However, the solution is significantly more challenging than the basic design and hence omitted from the poster.

2.3 Evaluation

DataLair has been implemented as a Linux kernel device mapper and compared to Hive [1]. DataLair public data reads perform almost as fast as a raw device formatted with ext4, as compared to a slowdown of almost *two* orders of magnitude for Hive [1]. Note that since public data is written to the device without any re-mappings, public data reads are not subject to the performance penalty imposed by DL-ORAM. More specifically, reads for public data does not incur a performance overhead. Public writes are slower since they need to perform an additional DL-ORAM simulation in the case where no private data is to be written. Even in that case, they perform almost 3-5 times better than Hive [1] due to the improved access complexity of DL-ORAM.

3. ACKNOWLEDGEMENT

This work was supported by the National Science Foundation through awards 1161541, 1318572, 1526102, and 152670.

4. REFERENCES

[1] *Hive*. "http://www.onarlioglu.com/hive".

[2] *TrueCrypt*. "http://truecrypt.sourceforge.net/".

[3] Blass, E.-O., Mayberry, T., Noubir, G., and Onarlioglu, K. Toward robust hidden volumes using write-only oblivious ram. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security* (New York, NY, USA, 2014), CCS '14, ACM, pp. 203–214.

[4] Czeskis, A., Hilaire, D. J. S., Koscher, K., Gribble, S. D., Kohno, T., and Schneier, B. Defeating encrypted and deniable file systems: Truecrypt v5.1a and the case of the tattling os and applications. In *Proceedings of the 3rd Conference on Hot Topics in Security* (Berkeley, CA, USA, 2008), HOTSEC'08, USENIX Association, pp. 7:1–7:7.

[5] Li, L., and Datta, A. Write-only oblivious ram based privacy-preserved access of outsourced data. Cryptology ePrint Archive, Report 2013/694, 2013. http://eprint.iacr.org/.

[6] McDonald, A. D., and Kuhn, M. G. Stegfs: A steganographic file system for linux. In *Proceedings of the Third International Workshop on Information Hiding* (London, UK, UK, 2000), IH '99, Springer-Verlag, pp. 462–477.

[7] Peters, T., Gondree, M., and Peterson, Z. N. J. DEFY: A deniable, encrypted file system for log-structured storage. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2014* (2015).

[8] Shi, E., h. Hubert Chan, T., Stefanov, E., and Li, M. Oblivious ram with o((log n) 3) worst-case cost.