Oblivious Storage: The Practicality of Private Access to Remote Data

Peter Williams, Radu Sion, Alin Tomescu
{petertw,sion}@cs.stonybrook.edu, alin.tomescu@gmail.com

We present *privatefs*, a fully-functional oblivious network file system in which files can be accessed on a remote server with computational access privacy and data confidentiality.

Oblivious RAM [1] is a construction providing *access pattern privacy*, hiding not just data contents but the location of reads and writes to this data. It lends itself naturally to the creation of a block device suitable for a file system. Due to existing results' impractical performance overhead this has not been previously possible. PD-ORAM [4] is a recent ORAM implementation that offers efficient access to parallel clients. A Linux-based deployment of PD-ORAM is used here to design and build *privatefs*.

An initial implementation of *privatefs* was built on top of the Linux Network Block Device (NBD) driver, which is the simplest and most natural approach, since PD-ORAM already provides a block interface. However, NBD supports only serial, synchronous requests. To take advantage of the parallel nature of PD-ORAM, *privatefs* is instead built on FUSE (Filesystem in Userspace [2]).

A second attempt used *ext2fuse*, a FUSE-based ext2 implementation [3], by rerouting block access through PD-ORAM. However, thread safety difficulties prevented us from modifying it to support parallel writes or reads. Additionally, because of its nature as a block device file system, *ext2fuse* requires mechanisms for allocating blocks for files, such as block groups, free block bitmaps and indirect file block pointers inside inodes. These mechanisms are not all thread-safe and pose a challenge to synchronize. Moreover, locking the code using synchronization primitives would not result in a sufficient degree of parallelization.

Instead, we implemented our own *privatefs* using the FUSE libraries in C++. It fully leverages the parallelism of PD-ORAM. Moreover, it takes advantage of the non-contiguous block labeling of PD-ORAM in a way that block-device file systems cannot.

Following the Linux file system model, in *privatefs* files are represented by inodes. Directories are inodes containing a list of directory entries; each directory entry is the name of a file or subdirectory along with its inode number. Inodes are numbered using 256-bit values and are mapped directly to ORAM blocks, such that inode x is stored in ORAM block x. Inodes hold metadata such as type, size and permissions. Both *privatefs* and (this instance of) PD-ORAM use 256-bit block identifiers and 4096-byte blocks.

Because the ORAM provides random access to 256-bit addressable blocks, a block can be allocated simply by generating a random 256-bit number. We take advantage of this in two ways. First, to read or write the i^{th} block of file with inode number x, the pair (x, i) is hashed with the collision-resistant SHA256 hash, yielding the 256-bit ORAM block ID for that file block. Second, when a new file is created, a 256-bit inode number is randomly generated, as opposed to maintaining and synchronizing access to an inode counter.

Our design eliminates the complexity of contiguous block device file systems and minimizes the need for locking when writing or reading files. As opposed to *ext2fuse*, *privatefs* does not incur the overhead of maintaining free block or inode bitmaps, grouping blocks into block groups, or traversing indirect block pointers to read files. The potential drawback is that sequential blocks of a given file will not be stored contiguously in the file system. However, this is harmless in ORAM, since there is no notion of sequential block numbers (which would compromise access privacy).

privatefs employs exclusive locks when reading and writing directories. In addition, an LRU cache is implemented to quickly retrieve an inode's data given its inode number and also for file path to inode number translation, which helps avoid long directory traversals (and associated locking).

privatefs communicates with the ORAM server by means of a proxy (written in Java), which receives block requests from the file system and satisfies them using parallel connections to the ORAM server. This design choice affords us a higher degree of modularity, enabling us to connect *privatefs* to other ORAM schemes in the future.

Overall, *privatefs* features a throughput modest when compared to unsecured file systems. However, this is the inherent cost of achieving privacy. *privatefs* is the first file system to provide access pattern privacy, and is fully functional and immediately usable.

References

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

[2] Csaba Henk and Miklos Szeredi. FUSE: Filesystem in Userspace. Online at http://sourceforge.net/projects/fuse, 2012.

[3] Tom Scholl. ext2fuse: ext2 filesystem in userspace. Online at http://sourceforge.net/projects/ext2fuse/, 2009.

[4] Peter Williams, Radu Sion, and Alin Tomescu. PD-ORAM: Parallelizing and de-amortizing oblivious access. *Under Submission*, 2012.