Spread Spectrum Storage with Mnemosyne

Steven Hand*
Sprint Advanced Technology Labs
1 Adrian Court
Burlingame, CA 94010, USA
steven.hand@cl.cam.ac.uk

Timothy Roscoe†
Intel Research Berkeley
2150 Shattuck Avenue, Suite 1300
Berkeley, CA 94704, USA
timothy.roscoe@intel-research.net

Abstract

We motivate and describe Mnemosyne[1], a highly private and reliable distributed storage system built over a number of untrusted, unreliable block stores. A client selects the pseudo-random set of block locations used to store a particular file by successive hashing of an initial key value. Resiliency to collisions is obtained by the use of erasure codes. The resulting system operates with decentralized key generation, is resistant to targeted attacks, and has soft capacity dependent on current load.

1 Introduction

Redundancy in distributed systems is used to provide increased performance and reliability by techniques such as striping and mirroring [1], fast fail-over [2], and byzantine fault-tolerance [3]. These schemes are oriented toward collections of machines and devices which are fairly small (a few hundred machines at most) relative to modern wide-area distributed systems – particularly 'peer-to-peer' systems – which may have particpant nodes numbering in the hundreds of thousands.

In the following we introduce *spread spectrum computing* as a new term for an old idea. In spread-spectrum computing a subset of a large number of distributed resources are selected according to some keyed pseudo-random process, using redundancy to remove the need to explicitly arbitrate usage between independent users. Although the selection is decentralized, if the candidate set is large enough and the pseudo-random procedure fairly uniform, we can expect relatively good load balancing. If the keys are good enough, the set of resources used by any particular client should be unpredictable and hence difficult to attack.

Mnemosyne [4] is a spread spectrum storage system which takes advantage of the widespread availability and low cost of network bandwidth and disk space. The system comprises servers that provide unreliable block storage, and clients which write and read blocks to and from the servers. Although our original motivation for the system was to provide practical steganographic storage, the design has a number of other benefits.

Firstly, it places all responsibility for higher-level functionality (such as a filing system) in the client. Hence each individual user can choose their own trade-offs in terms of cost, reliability availability and privacy. Secondly, tolerance to collisions effectively provides "soft capacity" – the total amount of data which may be stored in the system is shared automatically between the number of users. Thirdly, the extremely simple block storage model lends itself to a commercial deployment in which charging is done for write transactions rather than for storage space [5]. In addition to operational simplicity, this approach mitigates against certain denial of service attacks.

In the remainder of this short paper we first briefly survey related work, then provide an overview of how Mnemosyne operates. Finally we discuss how the principles of Mnemosyne relates to the wider notion of spread spectrum computing.

2 Related Work

Several research efforts are building Internet-scale storage systems [6–9]. In most cases, replication

*On leave from the University of Cambridge Computer Laboratory, JJ Thompson Avenue, Cambridge CB3 OFD, UK.

†This work was done while Timothy Roscoe was employed at Sprint Advanced Technology Labs.

[1] Pronounced *ne moz'nē*.

and/or an erasure code is used over a distributed hash table scheme such as [10–13]. This results in self-organising distributed file storage that provides high availability in the face of node failure or network partition. Other systems have used distribution and self-organisation to provide anonymity of access and prevent censorship [14–18].

Mnemosyne was originally conceived to be more in line with this latter class of system, although focusing less on the wide-scale publication of data. Instead, the system was designed to support the storage and retrieval of small size, high value information, and where the expectation is that each piece of information is shared by at most a small number of users. Indeed, two of our anticipated usage scenarios were private data storage for a single user, and confidential interpersonal messaging.

Mnemosyne also shares some common ground with private information retrieval systems [19, 20], although our focus is on hiding the storage of information rather than its communication.

3 Mnemosyne: Operational Overview

Mnemosyne was designed as a steganographic storage system; that is, a system in which users who do not have the required key not only are unable to read the contents of files stored under that key (as with a conventional encrypting file system), but furthermore are unable to determine the existence of files stored under that key.

This leads to an interesting property: since users cannot know anything about the location of file blocks stored by other users, it is always possible for them to unwittingly overwrite them; the existence of a file allocation table or list of in-use blocks defeats the steganographic properties of the system. Instead, Mnemosyne uses redundancy in the form of erasure codes to prevent file data being lost due to the write activity of other users.

The process by which a user of Mnemosyne stores a vector of bytes in the system can be broken down into four phases: dispersal, encryption, location, and distribution, which we describe in turn.

Dispersal:
In the dispersal phase, the data is encoded to make it robust in the face of block losses. We use Rabin's Information Dispersal Algorithm [21] in the field $GF(2^{16})$ to transform n blocks of data into $m > n$ blocks, any n of which suffice to recover the original data. In our prototype, for example, file data is by default treated as chunks of $n = 32$ blocks and each chunk transformed into $m = 3n = 96$ blocks. There are clearly trade-offs involved in the choice of m and n: we investigate these in detail in [5].

Encryption:
The dispersed blocks from the previous step are now encrypted under the user's key K. The purpose of this is twofold: firstly for security and privacy, but secondly for authenticity. This is necessary since blocks may be overwritten without notice by other users at any time. Hence we need a mechanism by which a user may determine whether a block subsequently retrieved from the network is really the one originally written. We use the AES algorithm in OCB mode [22] to provide security and a 16-byte MAC in one step

Location:
Mnemosyne achieves its security properties by storing encrypted data blocks in pseudo-random (and, to an adversary, unpredictable) locations in a large virtual network store, which is then mapped onto distributed physical storage devices. The locations of the encrypted blocks making up a data set are determined by a sequence of 256-bit values obtained by successively hashing (using SHA256) an initial value h_0 which depends ultimately on the filename and the user's key K.

Distribution:
The sequence of 256-bit location identifiers from the previous step is finally mapped onto physical storage using a peer-to-peer network of storage nodes, each of which holds a fixed-size physical block store.

For each block to be stored, both the node identifier and the block offset within the block store are derived from the corresponding location identifier. The top 160 bits of this identifier are used to as a node identifier in a Tapestry [13] network and a randomly selected Tapestry node is asked to route the block to the "surrogate" node for the 160-bit node identifier. The next 20 bits of the location id are then used as a block number in a 1GB block store.

The block store at each peer-to-peer node supports only the following two operations:

- **putBlock**(*blockid*, *data*)
- **getBlock**(*blockid*) $\rightarrow$ *data*

Note that the block storage nodes themselves need perform no authentication, encryption, access checking, or block allocation to ensure correct functioning of the system, though they might for billing purposes. Indeed, a block store may ignore the above operations entirely: as long as sufficiently many block stores implement the operations faithfully, users' data can be recovered.

3.1 Implementation

A working implementation of Mnemosyne for Linux exists. The client is written in C and C++, using freely available reference implementations of SHA-256 and AES-OCB. It provides a command-line interface with operations for key management, creating and listing directories, and copying files between Mnemosyne and the Unix file system. A simple block protocol over UDP is used for communication with block servers. The block server is implemented in Java and runs on Tapestry [13] nodes. Performance is plausible: we can copy files into Mnemosyne at 80 kilobytes per second, and read them at 160 kilobytes per second. A principal limiting factor in both cases is our (unoptimised) implementation of matrices over $GF(2^{16})$.

In [4] we describe one implementation of a per-user filing system over the data storage and retrieval procedures described above. The filing system uses directories and inodes to simplify the management of keys and initial hash values, and also handles versioning of files, a necessity since data is never actually deleted from Mnemosyne, but rather decays over time. As noted earlier, the choice of filing system and filing system parameters is completely under the control of the user.

We hope to make the source code for our implementation available in the near future.

4 Discussion

Mnemosyne is an illustrative example of spread spectrum computing in the area of storage, and it is instructive to compare it with traditional approaches to remote storage.

A typical file server implements a multi-user file system, explicitly allocating storage blocks to files and metadata. The server ensures the consistency of the file system metadata (including which blocks belong to which files), keeps storage quotas, and enforces security by preventing unauthorised users gaining access to files. Storage-Area Networks (SANs) and their recent extensions over IP networks present a simpler abstraction of storage: authorised users are now allocated virtual block devices which they manage themselves, an approach which moves filing system metadata to the client side of the interface. The SAN provides storage allocation between users and ensures that unauthorised users cannot write to a given block device.

Mnemosyne stands in stark contrast to these two schemes: block servers enforce no boundaries between users at all. Indeed, they do not need to know the identity of a user writing a block to ensure correct operation of the system. The block servers perform no arbitration whatsoever between users, clients or requests, other than to serialize read and write operations to the disks. All mechanisms for effective sharing of the global storage medium are dispersed among the clients.

This is what we mean by *spread spectrum storage*, by analogy with the use of bandwidth in many wireless communcation systems, including 802.11. Rendundancy, error correction, and encryption distributed among clients are used in place of explicit resource arbitration and allocation centralised in servers.

We feel spread spectrum techniques like Mnemosyne can be appropriate for very large, global-scale applications where central abitration is undesirable for a variety of reasons: scalability, reliability, and the need for a federation of service providers with no controlling authority. Our current work includes investigating the application of these principles to other wide-area resource allocation problems.

REFERENCES

[1] Thomas Anderson, Michael Dahlin, Jeanna Neefe, David Patterson, Drew Roselli, and Randolph Wang. Serverless network file systems. In *Proceedings of the 15th Symposium on Operating System Principles. ACM*, pages 109–126, Copper Mountain Resort, Colorado, December 1995.

[2] Fernando Pedone and Svend Frolund. Pronto: A Fast Failover Mechanism for Off-the-Shelf Commercial Databases. Technical Report HPL-2000-96, HP Laboratories, July 2000.

[3] Miguel Castro and Barbara Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of the 3rd Symposium on Operating Systems Design and Implementation, Usenix Association*, New Orleans, LA, USA, February 1999. USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS.

[4] Steven Hand and Timothy Roscoe. Mnemosyne: Peer-to-Peer Steganographic Storage. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems, Boston, MA*, March 2002.

[5] Timothy Roscoe and Steven Hand. Transaction-based Charging in Mnemosyne: a Peer-to-Peer Steganographic Storage System. In *Proceedings of the International Workshop on Peer-to-Peer Computing at Networking 2002, Pisa, Italy.*, May 2002.

[6] F. Dabek, M. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01), Banff, Canada.*, October 2001.

[7] Peter Druschel and Antony Rowstron. PAST: A large-scale, persistent peer-to-peer storage utility. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems (HotOS-VIII). Schloss Elmau, Germany*, May 2001.

[8] John Kubiatowicz, David Bindel, Yan Chen, Steven Czerwinski, Patrick Eaton, Dennis Geels, Ramakrishna Gummadi, Sean Rhea, Hakim Weatherspoon, Westley Weimer, Chris Wells, and Ben Zhao. OceanStore: An Architecture for Global-Scale Persistent Storage. In *Proceedings of the Ninth international Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS 2000)*, November 2000.

[9] Tim D. Moreton, Ian A. Pratt, and Timothy L. Harris. Storage, Mutability and Naming in *Pasta*. In *Proceedings of the International Workshop on Peer-to-Peer Computing at Networking 2002, Pisa, Italy.*, May 2002.

[10] S Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A Scalable Content-Addressable Network. In *Proceedings of ACM SIGCOMM 2001, San Diego, California, USA.*, August 2001.

[11] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications. In *Proceedings of ACM SIGCOMM 2001, San Diego, California, USA.*, August 2001.

[12] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proceedings of the 18th IFIP/ACM Internation Conference on Distributed Systems Platforms (Middleware 2001), Heidelberg, Germany*, November 2001.

[13] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB//CSD-01-1141, U. C. Berkeley, April 2000.

[14] D. Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Communications of the ACM*, 24(2):84–88, February 1981.

[15] Ross Anderson. The Eternity Service. In *Proceedings of the 1st International Conference on the Theory and Applications of Cryptology (PRAGOCRYPT'96)*. CTU Publishing House, Prague, 1996.

[16] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 46–66, July 2000.

[17] Roger Dingledine, Michael J. Freedman, and David Molnar. The Free Haven Project: Distributed Anonymous Storage Service. In *Workshop on Design Issues in Anonymity and Unobservability*, pages 67–95, July 2000.

[18] Marc Waldman, Aviel D. Rubin, and Lorrie Faith Cranor. Publius: A robust, tamper-evident, censorship-resistant, web publishing system. In *Proceeding of the 9th USENIX Security Symposium*, pages 59–72, August 2000.

[19] Benny Chor, Oded Goldreich, Eyal Kushilevitz, and Madhu Sudan. Private Information Retrieval. In *IEEE Symposium on Foundations of Computer Science*, pages 41–50, 1995.

[20] Michael K. Reiter and Aviel D. Rubin. Crowds: anonymity for Web transactions. *ACM Transactions on Information and System Security*, 1(1):66–92, 1998.

[21] M. Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Communications of the ACM*, 36(2):335–348, April 1989.

[22] Phillip Rogaway, Mihir Bellare, John Black, and Ted Krovetz. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. In *Eighth ACM Conference on Computer and Communications Security (CCS-8)*. ACM Press, August 2001.