

A New Approach for Fault Tolerant and Secure Distributed Storage

Arun Subbiah
School of Electrical and Computer Engineering
Georgia Institute of Technology
Atlanta, GA USA 30332
arun@ece.gatech.edu

1. Introduction

Fault-tolerant and secure data storage is an essential requirement in any dependable system. We consider the case where a fault-tolerant and secure data storage service is provided by a set of storage servers. Fault tolerance / security for such a storage service is characterized by three attributes - availability, integrity, and confidentiality. Availability and integrity are achieved by using multiple storage servers with the restriction that only upto a certain threshold of servers can become faulty. Confidentiality of the stored data is typically achieved using encryption.

Few works actually consider how the cryptographic keys, which are used to encrypt the data, must be stored and managed in a fault-tolerant way. Managing cryptographic keys becomes especially hard when we consider the storage service in collaborative environments, where several users have access rights to the same data objects and may access them concurrently. Changes in the access control lists will require re-encrypting the stored data with new cryptographic keys, which may be cumbersome. For data with long lifetimes, a good security practice is to periodically change the cryptographic keys, which would again require re-encryption of data with the new keys. The re-encryption process must not leak the plaintext, thus requiring a trusted agent to perform it. Also, from a design point of view, it may not be good to have the confidentiality of large volumes of data rely on the secrecy of a single cryptographic key. This key may get leaked to an adversary through vulnerabilities elsewhere in the system, such as ones due to defective security policies.

To manage the confidentiality of the stored data in an efficient way, we consider how the cryptographic keys must be stored at the storage service. Cryptographic keys must be stored at the storage service to facilitate sharing access to data by multiple authorized users. Obviously, the keys must be stored at the storage service in a confidential way without using additional keys. This is achieved [4] by storing the cryptographic keys using perfect secret sharing

schemes [6, 3], which encode the keys into *shares* such that only certain valid combinations of shares can be used to reconstruct the encoded keys, while invalid combinations of shares give no information on the encoded keys. By storing these shares at different storage servers, the encoded keys are kept confidential as long as not more than a threshold servers are compromised - an assumption used to also provide availability and integrity of the storage service.

Our work explores the possibility of using perfect secret sharing schemes on the data itself, as opposed to encrypting it. The challenges involved in managing the encryption keys securely (especially in shared-data environments) are thus avoided. We use replication of each share to provide availability and integrity guarantees. This is a novel approach to realizing a data store, where cryptography intersects dependable computing. Also, designing a data store using this approach provides a comprehensive solution to the problem of fault-tolerant data storage. Section 2 describes the architecture of the data store. Section 3 gives preliminary results on the feasibility of this approach, and Section 4 describes future research directions in this context.

2. Data Store Architecture

The architecture of the data store is shown in Figure 1. A distributed set of storage servers store the clients' data. A separate set of servers, or a subset of the storage servers, are used to provide the authentication, authorization, and the metadata service. The clients and servers connect to each other over a reliable and asynchronous network.

Clients upload or download data objects to / from the data storage service. The authentication component authenticates clients's requests and the authorization component issues capabilities, which the clients forward to the storage servers during uploads and downloads.

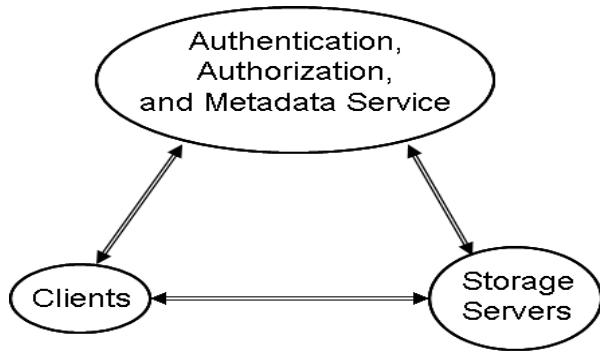


Figure 1. The Data Store Architecture

3. The New Approach: Using replication with XOR perfect secret sharing for fault tolerant data storage

The use of perfect secret sharing schemes to encode data directly has not been widely considered earlier because of the high computation overheads involved. In [7], we showed that the computation times to encode and decode 8 KB using standard perfect secret sharing schemes are over 3000 times slower than the AES encryption algorithm. All timing measurements were carried out on an Intel Pentium4 3GHz processor with 256 MB RAM running Linux 2.6.9. [8] uses perfect secret sharing to store data, but does not address the problem of high computation overheads.

In [7], we reduce the computation overheads dramatically by using XOR perfect secret sharing, and replicating each share sufficiently to meet availability and integrity requirements. In XOR perfect secret sharing, to encode a data bit into, say, ten shares, ten random bits are generated such that the XOR of the ten bits gives the data bit. These ten bits are the ten shares. Knowledge of upto nine shares gives no information on the data bit to the adversary. The computation overheads of this scheme are in the order of a few hundred μ s and are comparable to those of the AES symmetric-key encryption algorithm.

To combine perfect secret sharing and replication, we proposed the GridSharing framework in [7]. The storage servers are arranged in a logical grid. Secret sharing is done across rows, and shares are replicated along rows. Figure 2 shows a possible arrangement of the servers. To reason about the availability, integrity, and confidentiality requirements, we introduced a new fault model in [7]. Servers can exhibit crash, Byzantine, or *leakage-only* faults, corresponding to the availability, integrity, and confidentiality metrics. Byzantine-faulty servers can also leak its shares to an adversary. We use the threshold fault model, where we assume that not more than c servers can crash, not more than b servers can become Byzantine-

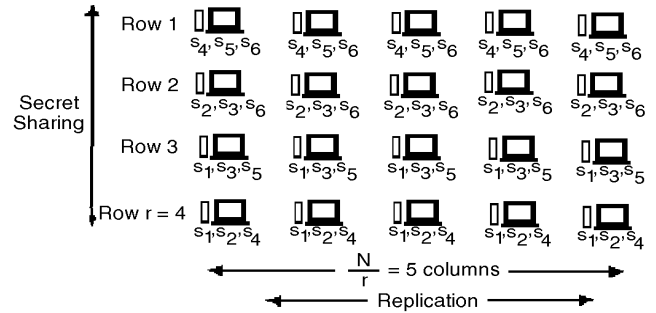


Figure 2. The *GridSharing* framework: N servers are arranged in a logical grid with r rows. Secret sharing is done across rows, and shares are replicated along rows. Setup shown for $N = 20$, $l = b = 1$, and $c = 6$.

faulty, and not more than l servers can exhibit leakage-only faults. We believe this fault model will be useful in the analysis of other works that take into account confidentiality requirements.

Figure 2 shows a possible server arrangement and share allocation scheme when $l = b = 1$ and $c = 6$. The depicted arrangement requires users' data to be encoded into six shares, denoted by s_1, \dots, s_6 . Thus, if upto $l + b = 2$ servers become faulty, an adversary can obtain only upto five shares, which give no information on the encoded data. All six shares are needed to decode the data. Similarly, availability and integrity requirements are met by replicating each share at at least $3b + c + 1 = 10$ servers.

A simple description of the clients' read-write protocols for Figure 2 is as follows: To write a data object, a client generates six shares such that the bitwise XOR of the six shares gives the original data object. The client writes to each server its assigned shares as indicated in Figure 2. To read a data object, a client fetches at least $(2b+1)$ values for each of the six shares. For each share, the value returned by at least $(b+1)$ servers must be the correct value. The bitwise XOR of the six shares gives the encoded data object.

The number of rows in the GridSharing framework can be varied to tradeoff between the minimum number of servers required (for a given l , b , and c) and the storage space required at each server. This also leads to tradeoffs in other performance metrics, such as encoding and decoding times, and the overall read and write latencies. A full analysis of these tradeoffs is given in [7]. More importantly, we showed that the read and write latencies of GridSharing are comparable to approaches that use encryption, thus paving the way to realizing a practical data store that uses perfect secret sharing techniques to store data directly.

4. Future Directions

The GridSharing framework in [7] establishes that an efficient fault-tolerant data storage service can be constructed using perfect secret sharing schemes and replication. This section gives an overview of future directions in our research.

4.1. Long-term fault tolerance (proactive security)

Storage servers can become faulty and be subsequently repaired. Given sufficient time, all the storage servers could have become faulty at some point. All the shares could be leaked to an adversary, or enough replicas of a share could have been destroyed or corrupted, leading to the data store not being fault-tolerant in the long term.

The concept of proactive security or long-term fault tolerance was introduced in [5]. Real time is divided into time periods, and the servers execute distributed *share verification and repair* and *share renewal* protocols. In the *share verification and repair* protocol, servers check if they have any corrupted or missing shares, and repair themselves by fetching copies from other servers. In the *share renewal* protocol, servers distributively change the shares such that the encoded data is unchanged and never revealed in the process.

The challenge here is to design protocols that scale with large volumes of data. The overheads in standard share renewal protocols are the high computation overheads due to the underlying secret sharing schemes in use. Further, executing the share renewal protocol for each data item drains processor and network resources. Since we use XOR secret sharing, where each bit is individually encoded, we have found that using stream ciphers offers a practical solution to this problem. A storage server could specify the key to be used for the stream cipher to the other storage servers holding different shares, and it can compute the bit stream it must use so that the net result of XORing the different key streams nullify. Implementation and performance measurements of this technique are in progress.

Regarding the *share verification and repair* protocol, we have each share managed using replication-based protocols. We have found that the problem can be reduced to that of purely-replicated systems (no confidentiality), and therefore our work has broader applicability. A protocol such as [1] along with a tool like tripwire can be used for this purpose. However, the protocol in [1] does not scale with the number of writes in a given time window. We are investigating two approaches to solve this problem: 1) Execute the protocol in [1] for batches of writes and use trees to identify corrupted shares, and 2) use threshold signatures and have the servers check data integrity locally. The right approach

will depend on the read-write access patterns.

4.2. Distributed computations on stored data

Homomorphic secret sharing schemes [2], such as Shamir's scheme [6], allow computations over data objects to be performed via computations over their shares. This allows a whole class of applications, such as databases, to make use of our storage model. The fact that we use replication-and-voting instead of mathematical methods allows straightforward use of our approach. Even data obfuscation, used for privacy maintenance, can be done by perturbing the shares. Such applications are not possible with data stored using symmetric-key encryption techniques. We are actively investigating this area of research.

References

- [1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Lazy verification in fault-tolerant distributed storage systems. In *Proceedings of the International Symposium on Reliable Distributed Systems*, 2005.
- [2] J. C. Benaloh. Secret sharing homomorphisms: keeping shares of a secret secret. In *Crypto*, 1987.
- [3] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the National Computer Conference*, 1979.
- [4] M. Herlihy and J. D. Tygar. How to make replicated data secure. In *Crypto*, 1987.
- [5] R. Ostrovsky and M. Yung. How to withstand mobile virus attacks. In *Proceedings of the 10th Symposium on the Principles of Distributed Computing*, 1991.
- [6] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.
- [7] A. Subbiah and D. M. Blough. An approach for fault tolerant and secure data storage in collaborative work environments. In *Proceedings of the ACM International Workshop on Storage Security and Survivability (StorageSS)*, 2005.
- [8] T. M. Wong. *Decentralized recovery for survivable storage systems*. PhD thesis, Carnegie Mellon University, 2004.