Secure, Archival Storage with POTSHARDS

Mark W. Storer, Kevin M. Greenan, Ethan L. Miller, Kaladhar Voruganti

There are many factors motivating the need to securely store private data for long periods of time. These range from requirements on business data demanded by recent legislation to the push towards digital creation of cultural and family heritage data. This information often needs to be stored securely; data such as medical and legal documents that could be important to future generations must be kept indefinitely but must not be publicly accessible. Unfortunately, existing storage solutions have yet to prove their ability to store data securely for long periods of time. To this end, the goal of secure, archival storage is to provide long-term data security and reliability.

Archival storage has a unique usage model that is poorly served with techniques commonly found in conventional storage. The usage model of secure, long-term archival storage is write-once, read-maybe and thus stresses throughput over low-latency performance. This is quite different from the top storage tier of a hierarchical storage solution that stresses low-latency access or even bottom-tier backup storage. The usage model of long-term archives also have the unique property that the reader may have little knowledge of the system's contents and no contact with the original writer; while file lifetimes may be indefinite, user lifetimes certainly are not.

One of the common mechanisms used to provide data secrecy is encryption. This can be seen in systems such as OceanStore, FARSITE, SNAD, Plutus and many others. While sufficient for short-term protection, it is ill-suited for long data lifetimes as encryption is only computationally secure. To combat the problem of key loss a lot of attention has been payed to key management. Archival storage exacerbates this problem because in order to prevent a single archive from obtaining the unencrypted data, re-encryption must occur over the old encryption, resulting in a long key history for each file. Since these keys are all outside data, a problem with any of the keys in the key history can render the data inaccessible when it is requested. Additionally a compromised archive is a problem regardless of the encryption algorithm that is used. An adversary who compromises an archive need only wait for cryptanalysis techniques to catch up to the encryption used at the time of the compromise.

Another mechanism used to provide secrecy is secret splitting [1]. This is seen in systems such as PASIS, GridSharing and Cleversafe. Using this technique the data is split into a set of shares where, depending on the algorithm, either all or a subset of the secrets are recombined to reconstruct the data. While provably secure, the use of secret splitting in a storage system must consider the danger of an inside attacker that knows the location of each of the secret shares. Few systems are designed to prevent attacks by insiders at one or more of the archives who can determine which pieces they need from other archives and steal those specific blocks of data, enabling a breach of secrecy with relatively minor effort. This problem is particularly difficult given the long time that data must remain secret, since such breaches could occur over years, making detection of small-scale intrusions nearly impossible.

To address the many security requirements for long-term archival storage, we are designing and implementing POTSHARDS (Protection Over Time, Securely Harboring And Reliably Distributing Stuff). Since we are designing the system specifically for secure, long-term storage, we identified three basic design tenets to help focus our efforts. First, we assumed that encrypted data could be read by anyone given enough time and advanced cryptanalysis. Second, we aim to ensure that data must be recoverable without any information from outside the set of archives (such as encryption keys). Our third assumption is that individuals are more likely to be malicious than an aggregate. In other words, our system should trust the consensus of groups of archives.

The basic architecture of POTSHARDS is a client communicating with a number of independent archives which utilize secure, distributed RAID techniques for redundancy. The client utilizes secret sharing algorithms to produce a set of *shards* which are distributed to the archives in a manner that ensures that no single archive receives enough shards to rebuild any of the user's data. By using secret splitting techniques, the secrecy in POTSHARDS has a degree of future-proofing built into it—it can be proven that an adversary with infinite computational power cannot gain any of the original data, even if an entire archive is compromised. Each client privately maintains an index of their shards and the archive that they are stored upon. This index is used during a normal read operation in which the client requests shards from the archives and reconstructs their data. If the client loses their index they are still able to reconstruct their data, albeit more slowly, from the shards alone. Each shard includes an approximate pointer to the next shard. Unlike a traditional pointer which indicates an exact location in a namespace, an approximate pointer is less specific and only indicates a region. The use of approximate pointers provides a great deal of security by preventing an intruder who compromises an archive or an inside attacker from knowing exactly which shards to steal from other archives. An intruder would have to steal *all* of the shards the approximate pointer could refer to, and would have to steal all of the shards they refer to, and so on. All of this would have to bypass the authentication mechanisms of each archive, and archives would be able to identify the access pattern of a thief, who would be attempting to obtains shards that may not exist.

Our current implementation consists of roughly 15,000 lines of Java 5.0 code. While there is further optimization to perform, it is currently capable of the system's four basic operations: reading, writing, recovering from a lost archive, and recovering data from shards. With these four operations functioning, we have begun to explore other aspects of the system. One key area we are looking at is the behavior of approximate pointers and how they relate to user's namespace within POTSHARDS. Understanding this relationship is important to balancing the need between data security and data recovery.

References

[1] A. Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, Nov. 1979.