POTSHARDS: Secure Long-Term Storage Without Encryption

Mark W. Storer Kevin M. Greenan Ethan L. Miller
University of California, Santa Cruz
Kaladhar Voruganti
Network Appliance†

Abstract

Users are storing ever-increasing amounts of information digitally, driven by many factors including government regulations and the public's desire to digitally record their personal histories. Unfortunately, many of the security mechanisms that modern systems rely upon, such as encryption, are poorly suited for storing data for indefinitely long periods of time—it is very difficult to manage keys and update cryptosystems to provide secrecy through encryption over periods of decades. Worse, an adversary who can compromise an archive need only wait for cryptanalysis techniques to catch up to the encryption algorithm used at the time of the compromise in order to obtain "secure" data.

To address these concerns, we have developed POTSHARDS, an archival storage system that provides long-term security for data with very long lifetimes without using encryption. Secrecy is achieved by using provably secure secret splitting and spreading the resulting shares across separately-managed archives. Providing availability and data recovery in such a system can be difficult; thus, we use a new technique, approximate pointers, in conjunction with secure distributed RAID techniques to provide availability and reliability across independent archives. To validate our design, we developed a prototype POTSHARDS implementation, which has demonstrated "normal" storage and retrieval of user data using indexes, the recovery of user data using only the pieces a user has stored across the archives and the reconstruction of an entire failed archive.

1 Introduction

Many factors motivate the need for secure long-term archives, ranging from the relatively short-term (for archival purposes) requirements on preservation, retrieval and security properties demanded by recent legislation [1, 20] to the indefinite lifetimes of cultural and family heritage data. As users increasingly create and store images, video, family documents, medical records and legal records digitally, the need to securely preserve this data for future generations grows correspondingly. This information often needs to be stored securely; data such as medical records and legal documents that could be important to future generations must be kept indefinitely but must not be publicly accessible.

The goal of a secure, long-term archive is to provide security for relatively static data with an indefinite lifetime. There are three primary security properties that such archives aim to provide. First, the data stored must only be viewable by authorized readers. Second, the data must be available and accessible to authorized users within a reasonable amount of time, even to those who might lack a specific key. Third, there must be a way to confirm the integrity of the data so that a reader can be reasonably assured that the data that is read is the same as the data that was written.

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 has 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. For digital "time capsules" that must last for decades or even centuries, the writer is assumed to be gone soon after the data has been written.

There are many novel storage problems [3, 32] that result from the potentially indefinite data lifetimes found in long-term storage. This is partially due to mechanisms such as cryptography that work well in the short-term but are less effective in the long-term. In long-term applications, encryption introduces the problems of lost

†Work performed while a member of IBM Almaden Research

keys, compromised keys and even compromised cryptosystems. Additionally, the management of keys becomes difficult because data will experience many key rotations and cryptosystem migrations over the course of several decades; this must all be done without user intervention because the user who stored the data may be unavailable. Thus, security for archival storage must be designed explicitly for the unique demands of long-term storage.

To address the many security requirements for long-term archival storage, we designed and implemented POTSHARDS (Protection Over Time, Securely Harboring And Reliably Distributing Stuff), which uses three primary techniques to provide security for long-term storage. The first technique is secret splitting [28], which is used to provide secrecy for the system's contents. Secret splitting breaks a block into *n* pieces, *m* of which must be obtained to reconstitute the block; it can be proven that any set of fewer than *m* pieces contains *no* information about the original block. As a result, secret splitting does not require the same updating as encryption, which is only computationally secure. By providing data secrecy without the use of encryption, POTSHARDS is able to move security from encryption to the more flexible and secure authentication realm; unlike encryption, authentication need not be done by computer, and authentication schemes can be easily changed in response to new vulnerabilities. Our second technique, *approximate pointers*, makes it possible to reconstitute the data in a reasonable time even if all indices over a user's data have been lost. This is achieved without sacrificing the secrecy property provided by the secret splitting. The third technique is the use of secure, distributed RAID techniques across multiple independent archives. In the event that an archive fails, the data it stored can be recovered without the need for other archives to reveal their own data.

We implemented a prototype of POTSHARDS and conducted several experiments to test its performance and resistance to failure. The current, CPU-bound implementation of POTSHARDS can read and write data at 2.5–5 MB/s on commodity hardware but is highly parallelizable. It also survives the failure of an entire archive with no data loss and little effect seen by users. In addition, we demonstrated the ability to rebuild a user's data from all of the user's stored shares without the use of a user index. These experiments demonstrate the system's suitability to the unique usage model of long-term archival storage.

2 Background

Since POTSHARDS was designed 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 sufficient CPU cycles and advances in cryptanalysis. This means that, if all of an archives encrypted contents are obtained, an attacker can recover the original information. Second, data must be recoverable without any information from outside the set of archives. Thus, fulfilling requests in a reasonable time cannot require anything stored outside the archives, including external indexes or encryption keys. If this assumption is violated, there is a high risk that data will be unrecoverable after sufficient time has passed because the needed external information has been lost. Third, we assume that individuals are more likely to be malicious than an aggregate. In other words, our system can trust a group of archives even though it may not trust an individual archive. The chances of every archive in the system colluding maliciously is small; thus, we designed the system to allow rebuilding of stored data if all archives cooperate.

In designing POTSHARDS to meet these goals, we used concepts from various research projects and developed additional techniques. There are many existing storage systems that satisfy some of the design tenets discussed above, ranging from general-purpose distributed storage systems to distributed content delivery systems, to archival systems designed for short-term storage and archival systems designed for very specific uses such as public content delivery. A representative sample of these systems is summarized in Table 1. The remainder of this section discusses each of these primary tenets within the context of the related systems. Since these existing systems were not designed with secure, archival storage in mind, none has the combination of long-term data security and proof against obsolescence that POTSHARDS provides.

2.1 Archival Storage Models

Storage systems such as Venti [23] and Elephant [26] are concerned with archival storage, but tend to focus on the near-term time scale. Both systems are based on the philosophy that inexpensive storage makes it feasible to store many versions of data. Other systems, such as Glacier [13], are designed to take advantage of the underutilized client storage of a local network. These systems, and others that employ "checkpoint-style" backups, address neither the security concerns of the data content nor the needs of long-term archival storage. Venti and commercial systems such as the EMC Centera [12] use content-based storage techniques to achieve their goals, naming blocks based on a secure hash of their data. This approach increases reliability by providing an easy way to verify the content of a block against its name. As with the short-term storage systems described above, security

System	Secrecy	Authorization	Integrity	Blocks for Compromise	Migration
FreeNet	encryption	none	hashing	1	access based
OceanStore	encryption	signatures	versioning	m (out of n)	access based
FarSite	encryption	certificates	Merkle trees	1	continuous relocation
Publius	encryption	password (delete)	retrieval based	m (out of n)	
SNAD / Plutus	encryption	encryption	hashing	1	
GridSharing	secret splitting		replication	1	
PASIS	secret splitting		repair agents, auditing	m (out of n)	
CleverSafe	information dispersal	unknown	hashing	m (out of n)	none
Glacier	user encryption	node auth.	signatures	n/a	
Venti	none		retrieval	n/a	
LOCKSS	none		vote based checking	n/a	site crawling
POTSHARDS	secret splitting	pluggable	algebraic signatures	$O(R^{m-1})$	device refresh

Table 1: *Capability overview of the storage systems described in Section 2. "Blocks to compromise" lists the number of data blocks needed to brute-force recover data given advanced cryptanalysis; for POTSHARDS, we assume that an approximate pointer points to R shard identifiers. "Migration" is the mechanism for automatic replication or movement of data between nodes in the system.*

is ensured by encrypting data using standard encryption algorithms.

Some systems, such as LOCKSS [18] and Intermemory [10], are aimed at long-term storage of open content, preserving digital data for libraries and archives where file consistency and accessibility are paramount. These systems are developed around the core idea of very long-term access for public information; thus file secrecy is explicitly not part of the design. Rather, the systems exchange information about their own copies of each document to obtain consensus between archives, ensuring that a rogue archive does not "alter history" by changing the content of a document that it holds.

2.2 Storage Security

Many storage systems seek to enforce a policy of secrecy for their contents. Two common mechanisms for enforcing data secrecy are encryption and secret splitting.

2.2.1 Secrecy via Encryption

Many systems such as OceanStore [25], FARSITE [2], SNAD [19], Plutus [16], and e-Vault [15] address file secrecy but rely on the explicit use of keyed encryption. While this may work reasonably well for short-term secrecy needs, it is less than ideal for the very long-term security problem that POTSHARDS is addressing. Encryption is only computationally secure and the struggle between cryptography and cryptanalysis can be viewed as an arms race. For example, a DES encrypted message was considered secure in 1976; just 23 years later, in 1999, the same DES message could be cracked in under a day [29]; future advances in quantum computing have the potential to make many modern cryptographic algorithms obsolete.

The use of long-lived encryption implies that re-encryption must occur to keep pace with advances in cryptanalysis in order to ensure secrecy. 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 external data, a problem with any of the keys in the key history can render the data inaccessible when it is requested.

Keyed cryptography is only computationally secure, so compromise of an archive of encrypted data is a potential problem regardless of the encryption algorithm that is used. An adversary who compromises an encrypted archive need only wait for cryptanalysis techniques to catch up to the encryption used at the time of the compromise. If an insider at a given archive gains access to all of its data, he can decrypt any desired information even if the data is subsequently re-encrypted by the archive, since the insider will have access to the new key by virtue of his internal access. This is unacceptable, since the data's existence on a secure, long-term archive suggests that data will still be valuable even if the malicious user must wait several years to read it.

Some content publishing systems utilize encryption, but its use is not motivated solely by secrecy. Publius [34] utilizes encryption for write-level access control. Freenet [6] is designed for anonymous publication and encryption is used for plausible deniability over the contents of a users local store. As with secrecy, the use of encryption to enforce long-lived policy is problematic due to the mechanism's computationally secure nature.

2.2.2 Secrecy via Splitting

To address the issues resulting from the use of encryption, several recent systems including PASIS [11, 36] and GridSharing [33] have used or suggested the use [31] of *secret splitting* schemes [5, 22, 24, 28]; a related approach used by Mnemosyne [14] and CleverSafe [7] uses encryption followed by information dispersal (IDA) to attempt to gain the same security. In secret splitting, a secret is distributed by splitting it into a set number n of