HAIL: A High-Availability and Integrity Layer for Cloud Storage

Kevin D. Bowers
RSA Laboratories
kbowers@rsa.com

Ari Juels
RSA Laboratories
ajuels@rsa.com

Alina Oprea
RSA Laboratories
aoprea@rsa.com

Abstract

We introduce HAIL (High-Availability and Integrity Layer), a distributed cryptographic system that permits a set of servers to prove to a client that a stored file is intact and retrievable. HAIL strengthens, formally unifies, and streamlines distinct approaches from the cryptographic and distributed-systems communities. Proofs in HAIL are efficiently computable by servers and highly compact—typically tens or hundreds of bytes, irrespective of file size. HAIL cryptographically verifies and reactively reallocates file shares. It is robust against an active, mobile adversary, i.e., one that may progressively corrupt the full set of servers. We propose a strong, formal adversarial model for HAIL, and rigorous analysis and parameter choices. We show how HAIL improves on the security and efficiency of existing tools, like Proofs of Retrievability (PORs) deployed on individual servers. We also report on a prototype implementation.

1 Introduction

Cloud storage denotes a family of increasingly popular on-line services for archiving, backup, and even primary storage of files. Amazon S3 [1] is a well known example. Cloud-storage providers offer users clean and simple file-system interfaces, abstracting away the complexities of direct hardware management. At the same time, though, such services eliminate the direct oversight of component reliability and security that enterprises and other users with high service-level requirements have traditionally expected.

To restore security assurances eroded by cloud environments, researchers have proposed two basic approaches to client verification of file availability and integrity. The cryptographic community has proposed tools called proofs of retrievability (PORs) [24] and proofs of data possession (PDPs) [2]. A POR is a challenge-response protocol that enables a prover (cloud-storage provider) to demonstrate to a verifier (client) that a file F is retrievable, i.e., recoverable without any loss or corruption. The benefit of a POR over simple transmission of F is efficiency. The response can be highly compact (tens of bytes), and the verifier can complete the proof using a small fraction of F. Roughly speaking, a PDP provides weaker assurances than a POR, but potentially greater efficiency.

As a standalone tool for testing file retrievability against a single server, though, a POR is of limited value.[1] Detecting that a file is corrupted is not helpful if the file is irretrievable and thus the client has no recourse. Thus PORs are mainly useful in environments where F is distributed across multiple systems, such as independent storage services. In such environments, F is stored in redundant form across multiple servers. A verifier (user) can test the availability of F on individual servers via a POR. If it detects corruption within a given server, it can appeal to the other servers for file recovery. To the best of our knowledge, the application of PORs to distributed systems has remained unexplored in the literature.

A POR uses file redundancy *within a server* for verification. In a second, complementary approach, researchers have proposed distributed protocols that rely on queries *across servers* to check file availability [26, 35]. In a distributed file system, a file F is typically spread across servers with redundancy—often via an erasure code. Such redundancy supports file recovery in the face of server errors or failures. It can also enable a verifier (e.g., a client) to check the integrity of F by retrieving fragments of F from individual servers and cross-checking their consistency.

In this paper, we explore a unification of the two approaches to remote file-integrity assurance in a system that we call *HAIL (High-Availability and Integrity Layer)*.

HAIL manages file integrity and availability across a collection of servers or independent storage services. It makes use of PORs as building blocks by which storage resources can be tested and reallocated when failures are detected.

[1] A standalone POR is useful for quality-of-service testing. The *speed* of the verifier's response gives an upper bound on expected delivery throughput for F. We don't treat QoS issues in this paper.

HAIL does so in a way that transcends the basic single-server design of PORs and instead exploits both within-server redundancy and cross-server redundancy.

HAIL relies on a single trusted verifier—e.g., a client or a service acting on behalf of a client—that interacts with servers to verify the integrity of stored files. (We do not consider a clientless model in which servers perform mutual verification, as for distributed information dispersal algorithms such as [16, 18, 8, 21].)

HAIL offers the following benefits:

- *Strong file-intactness assurance:* HAIL enables a set of servers to prove to a client via a challenge-response protocol that a stored file F is fully intact—more precisely, that the client can recover F with overwhelming probability. HAIL protects against even small, e.g., single-bit, changes to F.
- *Low overhead:* The per-server computation and bandwidth required for HAIL is comparable to that of previously proposed PORs. Apart from its use of a natural file sharing across servers, HAIL improves on PORs by eliminating check values and reducing within-server file expansion.
- *Strong adversarial model:* HAIL protects against an adversary that is *active*, i.e., can corrupt servers and alter file blocks and *mobile*, i.e., can corrupt every server over time.
- *Direct client-server communication:* HAIL involves one-to-one communication between a client and servers. Servers need not intercommunicate—or even be aware of other servers' existence. (In comparison, some information dispersal algorithms involve server-to-server protocols, e.g., [16, 18, 8, 21].) The client stores just a secret key.
- *Static / dynamic file protection:* In this paper, we show how HAIL protects static stored objects, such as backup files and archives. But our tools and framework can be modified with little added overhead to accommodate file updates, i.e., to provide integrity assurance for dynamically changing objects. We briefly explain this direction in the paper conclusion.

Our two broad conceptual contributions in HAIL are:

Security modeling We propose a strong, formal model that involves a *mobile adversary*, much like the model that motivates proactive cryptographic systems [23, 22]. A mobile adversary is one capable of progressively attacking storage providers—and in principle, ultimately corrupting all providers at different times.

None of the existing approaches to client-based file-integrity verification treats the case of a mobile adversary. We argue that the omission of mobile adversaries in previous work is a serious oversight. In fact, we claim that *a mobile adversarial model is the only one in which dynamic, client-based verification of file integrity makes sense*. The most common alternative model is one in which an adversary (static or adaptive) corrupts a bounded number of servers. As real-world security model for long-term file storage, this approach is unduly optimistic: It assumes that some servers are *never* corrupted. More importantly, though, an adversarial model that assumes a fixed set of honest servers for all time does not require dynamic integrity checking at all: A robust file encoding can guarantee file recovery irrespective of whether or not file corruptions are detected beforehand.

HAIL design strategy: Test-and-Redistribute (TAR) HAIL is designed like a proactive cryptographic system to withstand a mobile adversary. But HAIL aims to protect integrity, rather than secrecy. It can therefore be *reactive*, rather than proactive. We base HAIL on a new protocol-design strategy that we call TAR (Test-And-Redistribute). With TAR, the client uses PORs to detect file corruption and trigger reallocation of resources when needed—and only when needed. On detecting a fault in a given server via challenge-response, the client communicates with the other servers, recovers the corrupted shares from cross-server redundancy built in the encoded file, and resets the faulty server with a correct share.

Our TAR strategy reveals that for many practical applications, PORs and PDPs are *overengineered*. PORs and PDPs assume a need to store explicit check values with the prover. In a distributed setting like that for HAIL, it is possible to obtain such check values from the collection of service providers itself. On the other hand, distributed protocols for checking file availability are largely *underengineered*: Lacking robust testing and reallocation, they provide inadequate protection against mobile adversaries.

Three main coding constructions lie at the heart of HAIL:

- *Dispersal code:* In HAIL, we use what we call a *dispersal code* for robustly spreading file blocks *across servers*. For the dispersal code in HAIL, we propose a new cryptographic primitive that we call an *integrity-protected error-correcting code* (IP-ECC). Our IP-ECC construction draws together PRFs, ECCs, and universal hash functions (UHFs) into a single primitive. This primitive is an error-correcting code that is, at the same time, a corruption-resilient MAC on the underlying message. The additional storage overhead is minimal—basically just one extra codeword symbol.

  In a nutshell, our IP-ECC is based on three proper-

ties of (certain) universal hash function families h: (1) h is linear, i.e., $h_\kappa(m) + h_\kappa(m') = h_\kappa(m + m')$ for messages m and m' and key κ; (2) For a pseudorandom function family (PRF) g, the function $h_\kappa(m) + g_{\kappa'}(m)$ is a cryptographic message-authentication code (MAC) on m; and (3) $h_\kappa(m)$ may be treated as a parity block in an error-correcting code applied to m.

- *Server code:* File blocks *within* each server are additionally encoded with an error-correcting code that we call a *server code*. This code protects against the low-level corruption of file blocks that may occur when integrity checks fail. (For efficiency, our server code is a computational or "adversarial" error-correcting code as defined in Bowers et al. [6].)

- *Aggregation code:* HAIL uses what we call an *aggregation code* to compress responses from servers when challenged by the client. It acts across multiple codewords of the dispersal code. One feature of the aggregation code is that it combines / aggregates multiple MACs in our IP-ECC into a single *composite MAC*. This composite MAC verifies correctly only if it represents a combination of valid MACs on each of the aggregated codewords.

  Note that while the aggregation code is built on an error-correcting code, it is computed as needed, and thus imposes no storage or file-encoding overhead.

Organization We review related work in section 2. We give an overview of the HAIL construction and its main technical ingredients in Section 3. We present our adversarial model in section 4 and describe technical building blocks for HAIL in section 5. We detail the HAIL protocol in section 6, and analyze its security and discuss parameter choices in Section 7. Finally, we give implementation results in section 8 and conclude in section 9.

2 Related Work

HAIL may be viewed loosely as a new, service-oriented version of RAID (Redundant Arrays of Inexpensive Disks). While RAID manages file redundancy dynamically across hard-drives, HAIL manages such redundancy across cloud storage providers. Recent multi-hour failures in S3 [17] illustrate the need to protect against basic service failures in cloud environments. In view of the rich targets for attack that cloud storage providers will present, HAIL is designed to withstand Byzantine adversaries. (RAID is mainly designed for crash-recovery.)

Information dispersal Distributed information dispersal algorithms (IDA) that tolerate Byzantine servers have been proposed in both *synchronous networks* [16], as well as *asynchronous* ones [18, 8, 21]. In these algorithms, file integrity is enforced within the pool of servers itself. Some protocols protect against faulty clients that send inconsistent shares to different servers [18, 8, 21]. In contrast, HAIL places the task of file-integrity checking in the hands of the client or some other trusted, external service and avoids communication among servers. Unlike previous work, which verifies integrity at the level of individual file blocks, HAIL provides assurance at the granularity of a full file. This difference motivates the use of PORs in HAIL, rather than block-level integrity checks.

Universal Hash Functions Our IP-ECC primitive fuses several threads of research that have emerged independently. At the heart of this research are *Universal Hash-Functions* (UHFs). (In the distributed systems literature, common terms for variants of UHFs are *algebraic signatures* [28, 35] or *homomorphic fingerprinting* [21].) UHFs can be used to construct *message-authentication codes (MAC)* [4, 20, 14] (see [31] for a performance evaluation of various schemes). In particular, a natural combination of UHFs with pseudorandom functions (PRFs) yields MACs; these MACs can be aggregated over many data blocks and thus support compact proofs over large file samples.

PORs and PDPs Juels and Kaliski (JK) [24] propose a POR protocol and give formal security definitions. The main JK protocol supports only a limited number of challenges, whose responses are precomputed and appended to the encoded file. Shacham and Waters (SW) [36] use an implicit MAC construction that enables an unlimited number of queries, at the expense of larger storage overhead. Their MAC construction is based on the UHF + PRF paradigm, but they construct a UHF based on a random linear function, rather than a more efficient, standard error-correcting code.

In concurrent and independent work, Bowers et al. [6] and Dodis et al. [13] give general frameworks for POR protocols that generalize both the JK and SW protocols. Both papers propose the use of an error-correcting code in computing server responses to client challenges with the goal of ensuring file extraction through the challenge-response interface. The focus of [13] is mostly theoretical in providing extraction guarantees for adversaries replying correctly to an arbitrary small fraction of challenges. In contrast, Bowers et al. consider POR protocols of practical interest (in which adversaries with high corruption rates are detected quickly) and show different parameter tradeoffs when designing POR protocols.

Ateniese et al. [2] propose a closely related construction called a *proof of data possession* (PDP). A PDP detects a large fraction of file corruption, but does not guarantee