Brief Announcement: Auditable Register Emulations

Vinicius Vielmo Cogo
LASIGE, Faculdade de Ciências, University of Lisbon, PT

Alysson Bessani
LASIGE, Faculdade de Ciências, University of Lisbon, PT

Abstract

We initiate the study of auditable storage emulations, which provide the capability for an auditor to report the previously executed reads in a register. We define the notion of `auditable register` and its properties, and establish tight bounds and impossibility results for auditable storage emulations in the presence of faulty base storage objects. Our formulation considers registers that securely store data using information dispersal (each base object stores only a block of the written value) and supporting fast reads (that complete in one communication round-trip). In such a scenario, given a maximum number f of faulty storage objects and a minimum number τ of data blocks required to recover a stored value, we prove that (R1) auditability is impossible if $\tau \leq 2f$; (R2) implementing a weak form of auditability requires $\tau \geq 3f + 1$; and (R3) a stronger form of auditability is impossible. We also show that (R4) signing read requests generically overcomes the lower bound of weak auditability, while (R5 and R6) totally ordering operations or using non-fast reads enables strong auditability. These results establish that practical storage emulations need f to $2f$ additional objects compared to their original lower bounds to support auditability.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms; Computer systems organization → Reliability; Security and privacy → Information accountability and usage control; Applied computing → Evidence collection, storage and analysis

Keywords and phrases Auditability, Secure Storage, Information Dispersal

Digital Object Identifier 10.4230/LIPIcs.DISC.2021.53

Related Version *Full Version*: `https://arxiv.org/abs/1905.08637`

Funding Work partially supported by the Fundação para a Ciência e Tecnologia (FCT, Portugal), through the LASIGE research unit (UIDB/00408/2020 and UIDP/00408/2020) and the IRCoC project (PTDC/EEI-SCR/6970/2014).

1 Introduction

Given a resilient storage system composed of n storage objects (e.g., [2, 11]), *information dispersal* techniques (e.g., erasure codes and secret sharing) traditionally split and convert a data value v into n coded blocks [7, 9, 10]. Each coded block b_{v_k} from value v is stored in a different base object o_k, and readers need to obtain only τ out of n coded blocks to effectively recover the original value v. In this type of solution, no base object stores the whole data value, which differentiates information dispersal from fully-replicated storage systems where each object retains a full copy of the value.

In this paper, we address the following question: *How to extend resilient storage emulations with the capability of auditing who has effectively read data from them?* More specifically, we intend to audit resilient storage systems for protecting them from readers trying to obtain data without being detected (i.e., audit completeness) and protecting correct readers from faulty storage objects trying to incriminate them (i.e., audit accuracy).

Model. We consider a system composed of an arbitrary number of *client processes* (writers, readers, and auditors) that interact with a set of n *base storage objects* by invoking operations (e.g., as remote procedure calls on top of asynchronous reliable, authenticated channels).

Faulty writers and *faulty auditors* are honest and can only fail by crashing. *Faulty readers* may be Byzantine, i.e., they can crash or request data from only a subset of objects, without complying with any read algorithm. This characterises an attack where they attempt to read data without being detected and reported by auditors.

Faulty storage objects can crash, omit their blocks to readers, omit read records to auditors, or record nonexistent read operations. More specifically, to isolate storage from logging, objects can fail by omission when accessing their blocks and arbitrarily when providing their log records. Omitting records to auditors means a faulty object may be helping a reader to avoid being detected by auditors. Producing records for nonexistent reads characterises an active attack where a faulty object may be trying to incriminate a reader. Furthermore, we assume no more than f storage objects are faulty.

Register Emulation. We consider a `dispersed register` as a *high-level shared storage object* that stores a value v using information dispersal schemes. This object provides two high-level operations: *a-write(v)* and *a-read()*. A high-level *a-write(v)* converts a value v, passed as an argument, into n coded blocks $b_{v_1}, b_{v_2}, ..., b_{v_n}$, and each coded block b_{v_k} is stored in the base object o_k. A high-level *a-read()* operation recovers the original value v from any subset consisting of a specific number τ of distinct blocks b_{v_k}.

Base objects in this work are `loggable R/W registers`, which is an object o_k that stores a data block b_{v_k} and has a log L_k to store records of every read operation that this base object responded to. This object o_k provides three low-level operations:

- *rw-write*(b_{v_k}): writes the data block b_{v_k}, passed as an argument, in this base object o_k and returns an *ack* to confirm that the operation succeeded.
- *rw-read()*: returns the data block b_{v_k} currently stored in this base object o_k (i.e., the block passed as argument in the latest preceding *rw-write*) or $\perp$ if no block has been written on it. It also creates a record $\langle p_r, label(b_{v_k})\rangle$[1] about this read in its log L_k.
- *rw-getLog()*: returns the log L_k of this base object o_k.

We extend the `dispersed register` emulation with a high-level operation *a-audit()*, which uses the fail-prone logs obtained from the *rw-getLog* operation in `loggable R/W registers` to compose an **`auditable register`** emulation. This emulation has access to a log $L \subseteq \bigcup_{k\in\{1..n\}} L_k$ from which auditors can infer who has effectively read a value from the register in the past. Four concepts are essential for our initial `auditable register` emulation: *providing sets*, *effective reads*, *fast reads*, and *available auditing quorums*.

First, we define a *providing set* based on the notion of an accepting set introduced by Lamport [8] to abstract the access to multiple base objects. In our work, a providing set $P_{p_r,v}$ is *maximal* if it contains all objects that have both stored a block associated with value v and returned this block to a reader p_r.

Second, we introduce the notion of an *effective read*, which characterises a providing set $P_{p_r,v}$ large enough (i.e., comprising at least τ base objects) that a reader p_r is able to effectively obtain value v from the received blocks.

---

[1] p_r is the identifier of the reader that invoked the *rw-read* operation and $label(b_{v_k})$ is an auxiliary function that, given a block b_{v_k}, returns a label (e.g., a unique identifier, hash, timestamp) associated to value v (from which the block b_{v_k} was derived). This label can be written in the first few bytes of each block and is critical for auditors to detect effective reads based on individual logs.

Third, to capture the most fundamental aspect of reading in dispersed storage, we consider (initially) that *high-level reads are fast* – i.e., each read completes in a single communication round-trip between the reader and the storage objects [4]. This faithfully represents a "stealthy" reader directly obtaining blocks from a number of storage objects without following any particular algorithm.

Fourth, we define an *available auditing quorum* A, where $|A| = n - f$, as the set of base objects from which the *a-audit* collects fail-prone individual logs to compose a *set of evidences* E_A about the effectively read values. Furthermore, we assume a threshold δ as the *minimal required number of collected records* obtained in the audit operation for an auditor create an evidence $\mathcal{E}_{p_r,v}$ of an effective read. Each *evidence* $\mathcal{E}_{p_r,v}$ contains at least δ records from different storage objects o_k proving that v was effectively read by reader p_r. This threshold δ is a configurable parameter that depends on the guarantees *a-audit* operations provide (defined below). A correct auditor receives E_A and reports all evidenced reads.

An `auditable register` provides an *a-audit* operation that guarantees *completeness* and at least one form of *accuracy* (i.e., weak or strong). The formal definitions of these properties are available in the full paper [3]. In summary, the completeness states that all effective reads that precede an *a-audit* are reported by auditors in this audit operation. The weak accuracy states that the *a-audit* operation never reports any effective read from a correct reader that has never tried to read any value. The strong accuracy states that the *a-audit* does not report any effective read of value v from a correct reader that has never effectively read this value v. In the remaining of this paper, we consider *weak auditability* when the storage system provides completeness and weak accuracy in audit operations and *strong auditability* when it provides completeness and strong accuracy.

2 Results

Several tight bounds and impossibility results are presented in the full paper [3] considering the previously mentioned model. We consider information dispersal as the primary form of `auditable register` emulations because alternative solutions that replicate the whole data can suffer from faulty base objects leaking data to readers without logging these read operations.

Our formulation stores data using information dispersal (each base object stores only a block of the written value) and initially supports fast reads (that complete in one communication round-trip). In such a scenario, given a maximum number f of faulty storage objects and a minimum number τ of data blocks required to recover a stored value, we prove that (R1) auditability is impossible if $\tau \leq 2f$; (R2) implementing a weak form of auditability requires $\tau \geq 3f + 1$; and (R3) a stronger form of auditability is impossible.

We also prove in [3] that (R4) signing read requests generically (i.e., without mentioning the value v to be read) overcomes the lower bound of weak auditability, (R5) totally ordering operations [5] or using non-fast reads (e.g., multi-round read algorithms [2]) enables strong auditability with $\tau \geq 3f + 1$, and (R6) combining non-fast reads with specific signed read requests (i.e., read requests with signatures valid only for a specific value v) overcomes the lower bound of strong auditability with $\tau \geq 2f + 1$. These results establish that practical storage emulations (e.g., [1, 2, 6]) need f to $2f$ additional objects compared to their original lower bounds to support auditability.

References

1. Soumya Basu, Alin Tomescu, Ittai Abraham, Dahlia Malkhi, Michael K. Reiter, and Emin Gün Sirer. Efficient verifiable secret sharing with share recovery in BFT protocols. In *Proc. of the 26th ACM Conference on Computer and Communications Security (CCS)*, page 2387–2402, 2019. doi:10.1145/3319535.3354207.
2. Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando Andre, and Paulo Sousa. DepSky: Dependable and secure storage in cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12:1–12:33, 2013. doi:10.1145/2535929.
3. Vinicius Vielmo Cogo and Alysson Bessani. Auditable register emulations. *CoRR*, abs/1905.08637, 2019. arXiv:1905.08637.
4. Rachid Guerraoui and Marko Vukolić. How fast can a very robust read be? In *Proc. of the 25th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 248–257, 2006. doi:10.1145/1146381.1146419.
5. Vassos Hadzilacos and Sam Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, Cornell University, 1994.
6. James Hendricks, Gregory R. Ganger, and Michael K. Reiter. Low-overhead Byzantine fault-tolerant storage. In *Proc. of the 21st ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 73–86, 2007. doi:10.1145/1294261.1294269.
7. Hugo Krawczyk. Secret sharing made short. In *Proc. of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 136–146, 1993. doi:10.1007/3-540-48329-2_12.
8. Leslie Lamport. Lower bounds for asynchronous consensus. *Distributed Computing*, 19(2):104–125, 2006. doi:10.1007/s00446-006-0155-x.
9. James S Plank. Erasure codes for storage systems: A brief primer. *The USENIX Magazine*, 38(6):44–50, 2013.
10. Michael Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*, 36(2):335–348, 1989. doi:10.1145/62044.62050.
11. Jason K. Resch and James S. Plank. AONT-RS: Blending security and performance in dispersed storage systems. In *Proc. of the 9th USENIX Conference on File and Storage Technologies (FAST)*, page 14, 2011.