

# Auditable Register Emulations

Vinicius V. Cogo

Alysson Bessani

LASIGE, Faculdade de Ciências, Universidade de Lisboa, Lisboa, Portugal

## Abstract

The widespread prevalence of data breaches amplifies the importance of auditing storage systems. Here we initiate the study of auditable storage emulations, which provides the capability for an auditor to discover the previously executed reads in a register. We precisely define the notion of auditable register and its properties, and establish tight bounds and impossibility results for auditable storage emulations in a Byzantine setting. Our formulation considers read-write registers that securely store data using information dispersal and support fast reads. In such scenario, given a maximum number  $f$  of faulty storage objects and a minimum number  $\tau$  of data blocks required to recover a written value, we prove that (1) audibility is impossible if  $\tau \leq 2f$ ; (2)  $\tau \geq 3f + 1$  is required for implementing a weak form of audibility; and (3) a stronger form of audibility is impossible. We also show that totally ordering operations or using non-fast reads enables such strong audibility, albeit requiring more replicas.

## 1 Introduction

**Motivation.** Characteristics like cost-effectiveness, high scalability, and ease of use promoted the migration from private storage infrastructures to public multi-tenant clouds in the last decade. Security and privacy concerns were the main deterrents for this migration since the beginning [33]. Numerous secure storage systems have been proposing the use of advanced cryptographic primitives to securely disperse data across multiple clouds (i.e., independent administrative domains) to reduce the risk of data breaches [30].

Given a secure storage system composed of  $n$  storage objects, these *information dispersal* techniques split and convert the original data item into  $n$  coded blocks [21, 26, 30]. Each coded block is stored in a different object and clients need to obtain only  $\tau$  out of  $n$  coded blocks to effectively recover the original data. In this type of solution, no object contains the whole data item, which differentiates information dispersal from fully-replicated storage systems (e.g., [1, 4, 24])—where each object stores a full copy of the data.

Despite the advances in secure storage systems, the increasing severity of data breaches (e.g., [28]) and the tightening of privacy-related regulations (e.g., GDPR [11]) have been driving the demand for further improvements on this topic. For instance, *auditability* [19] may enable the systematic verification of who has effectively read data in secure storage systems. Notably, such verification will allow one to separate these users from the whole set of clients that are authorized to read data but have never done so. It is an important step to detect data breaches (including those caused by authorized users, e.g., Snowden’s case [15]), analyze leakages, and sanction misuses.

**Problem.** In this paper, we address the following question: *How to extend Byzantine-resilient storage emulations with the capability of auditing who has effectively read data?* The answer must encompass the techniques used in these emulations, such as information dispersal [21, 26, 30] and (available) Byzantine quorum systems [24], for providing a R/W register abstraction [22]. More

specifically, we address the problem of protecting storage systems 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).

**Related Work.** Several auditing schemes were proposed to verify the integrity of data stored in multi-tenant external infrastructures [20, 25, 32]. They mainly focus on cryptographic techniques to produce retrievability proofs without the need to fetch all data from the system (e.g., [18, 36]) or on providing public integrity verification (e.g., [34, 35]). However, to the best of our knowledge, *there is no previous work on auditing who has effectively read data in a dispersed storage.*

Another topic related to our work is the accountability, which focuses on making systems' components accountable in a way their actions become non-repudiable [13, 37]. Works in the accountability literature have discussed generic scenarios for networked systems [14], described the necessary cryptographic building blocks [13, 37], or how evidences should be stored and protected [14].

Several other works have explored the space complexity of fault-tolerant register emulations (e.g., [2, 7, 9, 10]), including disintegrated storage [5] (e.g., [3, 6, 31]). However, *none of these works focuses on the quorum requirements for auditing read accesses in a storage system despite the existence of Byzantine objects.*

**Contributions.** This paper initiates the study of auditing in Byzantine-resilient storage by presenting lower bounds and impossibility results related with the implementation of an *auditable register* on top of  $n$  base objects despite the existence of  $f$  faulty ones. Our results show that, given a minimum number  $\tau$  of data blocks required to recover a data item from information dispersal schemes, (1) auditability is impossible with  $\tau \leq 2f$  (or  $n \leq 4f$ ); (2) when fast reads (reads executed in a single communication round-trip [12]) are supported,  $\tau \geq 3f + 1$  (or  $n \geq 5f + 1$ ) is required for implementing a weak form of auditability, while a stronger form of auditability is impossible; and (3) totally ordering operations and using non-fast reads can still provide such strong auditability.

## 2 Preliminaries

**System Model.** Our system is composed of an arbitrary number of *client processes*  $\Pi = \{p_1, p_2, \dots\}$ , which interact with a set of  $n$  *storage objects*  $\mathcal{O} = \{o_1, o_2, \dots, o_n\}$ . Clients can be subdivided into three main classes: *writers*  $\Pi_W = \{p_{w1}, p_{w2}, \dots\}$ , *readers*  $\Pi_R = \{p_{r1}, p_{r2}, \dots\}$ , and *auditors*  $\Pi_A = \{p_{a1}, p_{a2}, \dots\}$ . These different roles do not necessarily mean they have to be performed by different processes.

A *configuration*  $C$  is a vector of the states of all entities (i.e., processes and objects) in the system. An *initial configuration* is a specific configuration where all entities of the system are in their initial states. An *algorithm*  $\mathcal{A}$  defines the behavior of processes in  $\Pi$  as deterministic state machines, which can modify the system's states through *actions* (e.g., invoke and response). An *execution segment* is a (finite or infinite) sequence of alternated configurations and actions. An *execution*  $\rho$  is an execution segment that begins in an initial configuration. An *event* is the occurrence of an action in an execution. Clients *invoke* operations in the storage objects and wait for *responses*. Low-level operations in objects are instantaneous, atomic actions.

A sequence of invocations and responses compose a *history*  $\sigma$  of the system. We consider the following relationships between operations within a history. First, if a history  $\sigma$  contains the invocation of an operation  $op_1$  and its response, then  $op_1$  is *complete* in  $\sigma$ . Otherwise, if  $\sigma$  contains only the former, then operation  $op_1$  is *pending*. Second, if the response of an operation  $op_1$  precedes the invocation of another operation  $op_2$  in  $\sigma$ , then  $op_1$  *precedes*  $op_2$ . Third, if operations  $op_1$  and

$op_2$  do not precede each other in  $\sigma$ , then they are *concurrent*. Fourth, a history  $\sigma$  that does not contain concurrent operations is considered *sequential*.

**Fault Model.** Clients (i.e., writers, readers, and auditors) and storage objects that obey their specifications are said to be *correct*. *Faulty readers* can be either malicious or honest. Malicious (i.e., Byzantine) faulty readers may behave arbitrarily, i.e., combining omissive and assertive behaviors. For instance, they may deviate from the protocol specification by invoking contradictory operations in all or only some storage objects. Alternatively, honest faulty readers may only crash.

*Faulty writers* and *faulty auditors* are honest and can only fail by crashing. Writers are trusted because they are the data owners, who are the most interested part in the auditable register we are proposing in this work. This is a common assumption in the BFT storage literature (e.g., [6, 24]) since malicious writers could write invalid values anyway, compromising the application’s state. Furthermore, auditors are also trusted because they are controlled either by the same entity as writers or third-party entities writers trust.

A *faulty storage object* may deviate from its specification arbitrarily—i.e., data can be created, corrupted, deleted, or leaked to unauthorized parties [23]. They can also present omissive behavior, intermittent or not. Although our system tolerates the aforementioned faults, it operates correctly only if no more than  $f$  storage objects are faulty.

**R/W Register Specification.** A *read-write (R/W) register* is an object that stores a data value  $v$  from any domain  $\mathbb{V}$  and provides two low-level operations [22]:

- $rw\text{-}write(v)$ : writes the data value  $v \in \mathbb{V}$ , passed as argument, in the data object and returns an *ack* to confirm the operation succeed.
- $rw\text{-}read()$ : returns the data value  $v \in \mathbb{V}$  currently stored in the data object or  $\perp$  if no value has been written on it.

The behavior of a R/W register is given by its sequential specification, where every low-level  $rw\text{-}read$  operation on it returns the value written by the last preceding  $rw\text{-}write$  operation on this object, or the special value  $\perp \notin \mathbb{V}$  if no such operation exists.

**Emulated Registers.** We emulate a *shared register* that stores a value  $v$  from domain  $\mathbb{V}$  using R/W registers. It exposes high-level  $s\text{-}read()$  and  $s\text{-}write(v)$  operations to processes in  $\Pi$ . We consider multi-writer multi-reader (MWMR) registers where multiple writers from  $\Pi_w$  invoke  $s\text{-}write$  operations and multiple readers from  $\Pi_R$  invoke  $s\text{-}read$  operations. Our safety requirement is the safe semantics: a read can return any value if it is concurrent with a write operation, and returns the value written by the most recent write operation if there is no write operation concurrent with the read [22]. Our liveness requirement is wait-freedom, where every operation invoked by a process  $p$  returns within a finite number of events [17]. High-level read operations (e.g.,  $s\text{-}read$ ) in our system are *fast reads*.

**Definition 1** (Fast read [12]). An  $s\text{-}read$  operation is *fast* if it completes in a single communication round-trip between the reader and the storage objects.

**Information Dispersal.** Storage objects in  $\mathcal{O}$  are untrusted independent entities. We assume they store data in blocks generated using information dispersal schemes (i.e., a special case of disintegrated storage [5]) such as erasure codes [26, 27] and secret sharing [21, 29].

In these schemes, a high-level  $s\text{-}write(v)$  operation  $w_v$  converts a value  $v \in \mathbb{V}$  into  $n$  coded blocks  $b_{v_1}, b_{v_2}, \dots, b_{v_n}$  from a domain  $\mathbb{B}$ . Each coded block  $b_{v_k}$  produced in  $w_v$  is marked with a unique

write label  $l_{w_v}$  and sent to the  $k^{th}$  storage object in the system. These techniques guarantee that no object  $o_k$  stores an entire copy of value  $v$  and no client process recovers  $v$  by accessing less than a certain fraction of these blocks. For simplicity, we assume every distinct value is written only once.

A high-level  $s\text{-read}()$  operation recovers the original value  $v$  from any subset of a specific number ( $\tau$ ) of correct blocks  $b_{v_k}$  written in  $w_v$ . It means readers do not need to fetch blocks from all ( $n$ ) storage objects. More specifically, we introduce the notion of an *effective read* as described in Definition 2.

**Definition 2** (Effective read). A value  $v \in \mathbb{V}$ , written in  $w_v$ , is *effectively read* by a reader  $p_r$ , in history  $\sigma$ , when  $p_r$  has read  $\tau$  correct blocks  $b_{v_k}$  of  $w_v$  from different objects  $o_k$ .

Note that an effective read depends only on the number of correct blocks (written in the same write  $w_v$ ) that a reader  $p_r$  has already obtained from different objects. Reader  $p_r$  does not necessarily obtain all these blocks on a single  $s\text{-read}$  operation. There might exist cases where it is accomplished only after receiving responses from many subsequent  $s\text{-read}$  operations.

We also consider information dispersal schemes have a maximum number of Byzantine faults  $f$  tolerated at the same time. As a consequence, the minimum number of blocks to recover the original data must comply with Proposition 1, which guarantees that  $f$  malicious objects cannot create nonexistent correct values.

**Proposition 1.** *Any information dispersal scheme requires  $\tau > f$  to tolerate up to  $f$  Byzantine faults [16].*

**Available  $f$ -Threshold Byzantine Quorum Systems.** An available ( $f$ -threshold) quorum system guarantees the consistency and availability of stored data by executing operations in only a subset (i.e., a *quorum*  $Q$ ) of storage objects, instead of requiring clients to communicate with all of them to execute high-level operations. We consider all protocols in this paper require Byzantine quorum systems, which contain *at least*  $f + 1$  objects at the intersection of the quorums of any two operations [24].

To provide wait-free register emulations using quorum systems in asynchronous environments, these systems need to be available.

**Definition 3** (Available  $f$ -threshold quorums [24]). Any available  $f$ -threshold quorum  $Q$  is composed of  $q = n - f$  replicas to tolerate up to  $f$  faults.

We assume available quorums are composed of  $q = n - f$  replicas (instead of  $q \leq n - f$  [24]) because we are interested in the limit case where quorums of maximum size are used. Using available quorums guarantees that any complete  $s\text{-write}$  operation  $w_v$  stores correctly labeled blocks in at least a quorum of storage objects. At any point in history  $\sigma$ , an object that participated in the quorum of the last preceding  $s\text{-write}$  is considered an *up-to-date* object, whereas an object that did not participate on it is considered *stale*.

Finally, we assume that information dispersal schemes (e.g., erasure codes) are implemented using available quorums and comply with Proposition 2 to tolerate up to  $f$  Byzantine objects. This proposition defines the required minimum number ( $n$ ) of objects to be present in the whole system.

**Proposition 2.** *Any available Byzantine fault-tolerant  $\tau$ -of- $n$  information dispersal scheme requires at least  $n \geq \tau + 2f$  replicas to tolerate up to  $f$  faulty objects [16].*

This bound comes from the fact that available quorums require the response from only  $n - f$  replicas, with up to  $f$  of these responses from malicious objects. Note that malicious readers can obtain  $\tau$  correct blocks without accessing a quorum of storage objects.

### 3 Auditable Register Emulations

We extend the aforementioned register emulation by adding an operation *getLog()* to the storage objects (i.e., R/W registers). Invoking this operation on object  $o_k$  returns the log  $L_k$  containing records of every preceding read executed in this object. A *record*  $\langle p_r, l_{w_v} \rangle_k \in L_k$  contains the identifier of the reader  $p_r$  and the write label  $l_{w_v}$  associated with the value  $v$  whose the block  $b_{v_k}$  was read by  $p_r$ .

Based on these individual fail-prone logs, we define an *auditable register emulation* that has access to a *virtual log*  $L \triangleq \bigcup_{k \in \{1..n\}} L_k$  from which we can infer who has effectively read a value from the register. This emulation provides three high-level operations: *a-write*( $v$ ), *a-read*(), and *a-audit*(). An *a-write*( $v$ ) is an unmodified *s-write*( $v$ ) described in the previous section. An *a-read*() is an extended *s-read*() that reads blocks  $b_{v_k}$  from a quorum of objects, causing  $\langle p_r, l_{w_v} \rangle_k$  to be added to  $L_k$  on each accessed correct storage object  $o_k$ .

Finally, the third operation is an *a-audit*(). It obtains records from  $L$  and produces a *set of evidences*  $E$  about the values read. Each *evidence*  $\mathcal{E}_{p_r, v}$  contains a set of at least  $\ell$  records from different storage objects, which proves that  $v$  was *effectively read* by each reader  $p_r$  in history  $\sigma$ .

We are interested in auditing effective reads because we intend to audit who has actually read a data value  $v$ —including malicious readers that do not follow the read protocol and leave operations pending. A correct auditor receives  $E$  and reports all evidenced reads.

An *auditable register* provides an *a-audit*() operation that guarantees *completeness* and at least one form of *accuracy*, as stated in Definitions 4–6.

**Definition 4** (Completeness). Every value  $v \in \mathbb{V}$  *effectively read* by a reader  $p_r$  before the invocation of an *a-audit* in history  $\sigma$  is reported in  $E$  resulting from this audit operation, i.e.,  $\mathcal{E}_{p_r, v} \in E$ .

**Definition 5** (Weak Accuracy). A correct reader  $p_r$  that has never invoked an *a-read* before the invocation of an *a-audit* in history  $\sigma$  will not be reported in  $E$  resulting from this audit operation, i.e.,  $\forall v \in \mathbb{V}, \mathcal{E}_{p_r, v} \notin E$ .

**Definition 6** (Strong Accuracy). A correct reader  $p_r$  that has never effectively read a value  $v \in \mathbb{V}$  before the invocation of an *a-audit* in history  $\sigma$  will not be reported in  $E$  resulting from this audit operation as having read  $v$ , i.e.,  $\mathcal{E}_{p_r, v} \notin E$ .

While completeness is intended to protect the storage system from readers trying to obtain data without being detected, accuracy focuses on protecting *correct readers* from malicious storage objects incriminating them.

The difference between the weak and the strong variants of the accuracy property is in the precision of the auditor’s report. With weak accuracy, the audit reports only readers that have invoked read operations, which already separates them from the whole set of clients that are authorized to read a specific register but have never done so. With strong accuracy, the audit reports only the values readers have effectively read. Strong accuracy implies weak accuracy.

Complementing the other side of the accuracy property, incorrect (honest or malicious) readers may be reported by the audit in pending, incomplete, or partial reads. It means they had the intention to read the data and aborted it or crashed while doing so.

### 4 Preliminary Results

Auditing *fully-replicated* systems is impossible in the presence of Byzantine objects because each object stores a full copy of data and can alone give it to readers without logging the operations.

Therefore, we consider information dispersal techniques as the basic form of disintegrated storage [5] for auditable register emulations.

**Records Available for Auditing Registers.** In this section, we identify the minimum number of records from every effective read that will be available for *a-audit* operations. This number is then used to define the required number of records  $\ell$  to produce an evidence  $\mathcal{E}_{p_r, v}$ .

The three operations (write, read, and audit) of auditable registers are possibly executed on different available quorums that need to intersect. The worst-case scenario is presented in Figure 1a, where we illustrate the minimum intersections between each of these three quorums, identified as groups  $G_{1-4}$ . In this figure, every storage object belongs to a group and all objects within the same group contain the same state.

**Lemma 1.** *Any available  $a\text{-audit}()$  operation obtains at least  $\tau - 2f$  records of every preceding effective read in history  $\sigma$ .*

*Proof.* Let us assume any available quorum system composed of  $n$  storage objects, where clients wait for replies from  $q = n - f$  of them (Definition 3). Let us assume that, in the worst-case scenario, the quorums  $Q_w$  for an  $a\text{-write}(x)$ ,  $Q_r$  for an  $a\text{-read}()$ , and  $Q_a$  for an  $a\text{-audit}()$  operation differ as depicted in Figure 1a. In this scenario, the size of the intersection  $|Q_w \cap Q_r \cap Q_a|$  is  $n - 3f$ . From these  $n - 3f$  objects,  $f$  of them are malicious and  $n - 4f$  are correct, up-to-date objects.

Let us assume a malicious reader that effectively reads a value  $v$  after accessing only  $\tau$  objects instead of contacting the whole quorum  $Q_r$ . In the worst-case scenario, the malicious reader obtains  $f$  correct blocks from malicious storage objects (that do not log the read);  $f$  blocks from correct, up-to-date objects in  $G_1$  (that log the read but do not participate in the audit quorum  $Q_a$ ); and  $n - 4f$  blocks from correct, up-to-date objects (that log the records and participate in  $Q_a$ ). Since these  $n - 4f$  correct, up-to-date objects are the only ones that have both logged the records and participate in  $Q_a$ , by applying Proposition 2, we have that  $\tau - 2f$  is the minimum number of available records in  $L$  from  $Q_a$  for detecting every value  $v$  effective read in the system.  $\square$

Evidences are created in  $a\text{-audit}()$  only after finding, in  $L$ , at least  $\ell$  records from different objects attesting the same read value and reader.

**Corollary 1.** *The required number ( $\ell$ ) of records to produce an evidence  $\mathcal{E}_{p_r, v}$  is  $\ell \leq \tau - 2f$ .*

## 5 Resilience Lower Bounds

We present impossibility results for the three properties of auditable registers: completeness (Definition 4), weak accuracy (Definition 5), and strong accuracy (Definition 6). From this point on, *correct storage objects* start a history  $\sigma$  with the initial configuration composed of blocks of a value  $v \in \mathbb{V}$  and an empty log  $L = \emptyset$ . For simplicity, we depict the log records  $\langle p_r, l_{w_v} \rangle$  as  $v_r$  in the figures.

**Lemma 2.** *It is impossible to satisfy the **completeness** of auditable registers with  $\tau \leq 2f$ .*

*Proof.* Without loss of generality, let us assume that  $\tau = 2f$ . According to Proposition 2, a system with  $\tau = 2f$  must have at least  $4f$  nodes. Consider an auditable register implemented using four subset groups of objects  $G_{1-4}$ , as depicted in Figure 1b. Each group is composed of  $f$  objects. Group  $G_2$  contains malicious objects that may behave arbitrarily.

A malicious reader  $p_{r1}$  obtains  $f$  coded blocks from the correct nodes in  $G_3$  (that log the operation) and  $f$  blocks from the malicious nodes in  $G_2$  (which do not log the operation). This

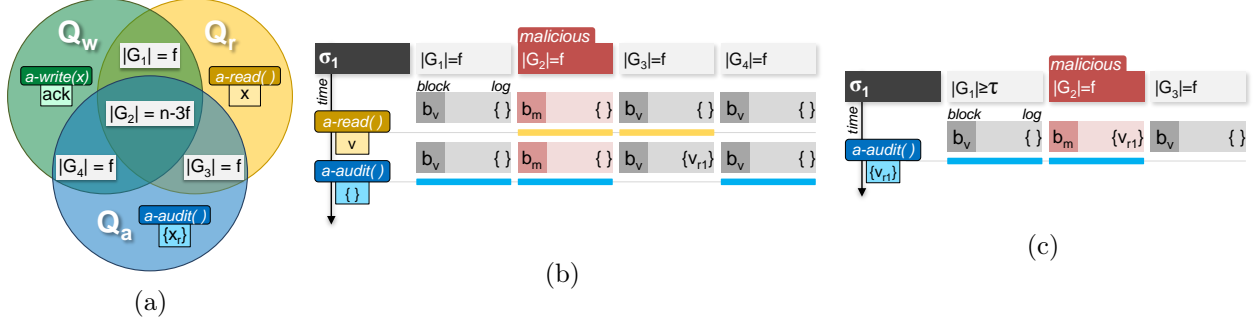


Figure 1: (a) The worst-case scenario of a triple quorum applied to auditable registers. (b) A scenario where  $\tau = 2f$  does not guarantee completeness. (c) A scenario where  $\ell = f$  does not guarantee weak accuracy.

malicious reader can decode the original value  $v$  after receiving the  $\tau = 2f$  correct blocks, performing thus an effective read. However, any audit quorum  $Q_a$  that does not include at least one object from group  $G_3$  will not receive any record for the read operation from reader  $p_{r1}$ . This history violates the completeness property since an evidence is never created if there is no record of this read to do so.  $\square$

**Lemma 3.** *It is impossible to satisfy the **weak accuracy** of auditable registers with  $\ell \leq f$ .*

*Proof.* Without loss of generality, let us assume that  $\ell = f$ . Consider an auditable register implemented using three subset groups of objects  $G_{1-3}$ , as depicted in Figure 1c, where the group  $G_1$  is composed of at least  $\tau$  objects and the remaining groups are composed of exactly  $f$  objects each. Group  $G_2$  contains malicious objects that may behave arbitrarily and log a nonexistent read operation. Any audit quorum  $Q_a$  that includes the  $f$  objects from group  $G_2$  will receive  $f$  records for a read operation that has never been invoked. If  $\ell = f$  is enough to create an evidence and report an effective read, then a correct auditor must report it, violating the weak accuracy property.  $\square$

The next theorem presents a scenario where these two properties are intended to be supported simultaneously.

**Theorem 1.** *It is impossible to satisfy both **completeness** and **weak accuracy** of auditable registers with  $\tau \leq 3f$ .*

*Proof.* Without loss of generality, let us assume that  $\tau = 3f$  is enough to satisfy both the weak accuracy and completeness properties. According to Proposition 2, a system with  $\tau = 3f$  must have at least  $5f$  nodes. Consider an auditable register implemented using five subset groups of objects  $G_{1-5}$ , as depicted in Figure 2, where each group is composed of  $f$  objects and group  $G_2$  contains only malicious objects.

In history  $\sigma_1$  (Figure 2a), a malicious reader  $p_{r1}$  obtains  $2f$  coded blocks from correct objects in  $G_3$  and  $G_4$  (that log the operation) and other  $f$  correct blocks from the malicious objects in  $G_2$  (which do not log the operation). Any audit quorum  $Q_a$  will receive at least  $f$  records either from the objects in group  $G_3$  or  $G_4$ . Evidences are created using these  $\ell \geq f$  records to report the read, satisfying completeness. However, as proved in Lemma 3 and illustrated in history  $\sigma_2$  (Figure 2b), it is impossible to guarantee weak accuracy with  $\ell \leq f$ .  $\square$

Now, we turn our attention to strong accuracy, i.e., the capability of an auditor to report exactly which value  $v$  each reader  $p_r$  has read.

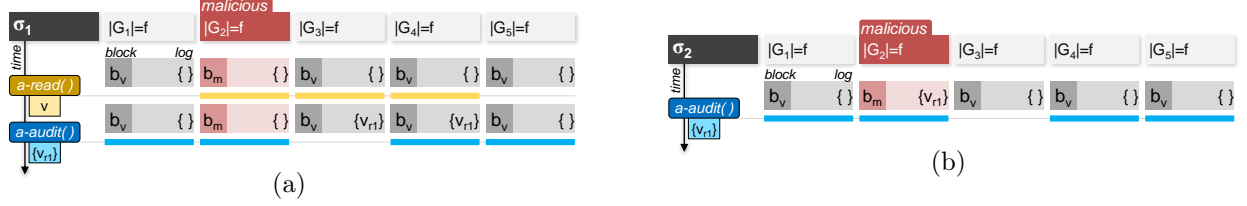


Figure 2: A scenario where  $\tau = 3f$  does not guarantee both completeness and weak accuracy.

**Lemma 4.** *It is impossible to satisfy the **strong accuracy** of auditable registers with  $\ell < \tau + f$ .*

*Proof.* Let us assume without loss of generality that  $\ell = \tau + f - 1$ . Consider an auditable register implemented using  $n \geq 2f + \tau$  objects divided in four subset groups of objects  $G_{1-4}$ , as depicted in Figure 3a, where group  $G_3$  is composed of  $\tau - 1$  objects, group  $G_4$  is composed of at least one object, and the remaining groups are composed of exactly  $f$  objects each. Group  $G_2$  contains only malicious objects.

A writer  $p_{w1}$  starts a write operation concurrently with a read operation from a correct reader  $p_{r1}$ . At a first moment, only objects from group  $G_4$  have received the write request and indeed wrote the blocks for value  $x$  on their registers. Then, the read request arrives in objects from groups  $G_{2-4}$ , where: the  $f$  malicious objects from  $G_2$  return blocks for the wrong value  $m$  (and log the read of  $v$  by  $p_{r1}$ );  $\tau - 1$  correct objects from group  $G_3$  return blocks for the value  $v$  (and log it); and all objects from group  $G_4$  correctly return blocks for the value  $x$  (and log the read of  $x$  by  $p_{r1}$ ). In this history  $\sigma_1$ , the reader  $p_{r1}$  has not received  $\tau$  correct data blocks for value  $v$ , which means it is unable to recover  $v$ .

However, any audit quorum  $Q_a$  that includes objects from groups  $G_2$  and  $G_3$  will return  $\tau + f - 1$  records  $\langle p_{r1}, l_{w_v} \rangle_k$ . If  $\ell = \tau + f - 1$  is enough to produce an evidence of an effective read, then the auditor will report this non-effective read, violating the strong accuracy property.  $\square$

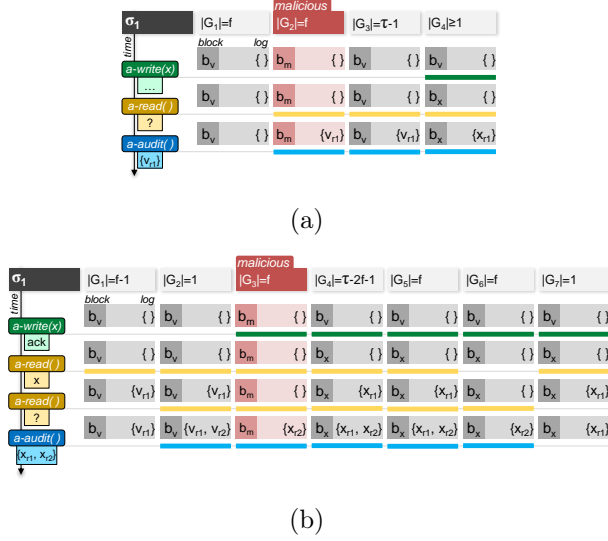
**Theorem 2.** *It is impossible to satisfy both **completeness** and **strong accuracy** of auditable registers.*

*Proof.* Without loss of generality, let us assume any number of blocks  $\tau \geq 2f + 1$  required to effectively read a value (Lemma 2). Consider an auditable register implemented using  $n \geq 2f + \tau$  objects (Proposition 2) divided in seven subset groups of objects  $G_{1-7}$ , as depicted in Figure 3b. Group  $G_1$  is composed of  $f - 1$  objects;  $G_2$  and  $G_7$  are composed of 1 object each;  $G_3$ ,  $G_5$ , and  $G_6$  are composed of  $f$  objects each; and  $G_4$  is composed of  $\tau - 2f - 1$  objects. Group  $G_3$  contains only malicious objects.

A writer  $p_{w1}$  executes a complete write operation in objects from groups  $G_{3-7}$ . Then, a correct reader  $p_{r1}$  executes a read operation in groups  $G_{1-5}$  and  $G_7$ . Malicious objects from  $G_3$  return their correct blocks for value  $x$  (but do not log the read operation). Reader  $p_{r1}$  receives exactly  $\tau$  correct data blocks and obtains the original value  $x$  (i.e., an effective read).

A second correct reader  $p_{r2}$  executes a read operation in groups  $G_{2-6}$ . Malicious objects from  $G_3$  do not return their correct blocks for value  $x$  (but they do log the read of  $x$  by  $p_{r2}$ ). Reader  $p_{r2}$  receives  $\tau - 1$  correct data blocks for value  $x$ , which is insufficient to recover the original value  $x$  (i.e., a non-effective read). After these two reads, any audit quorum  $Q_a$  receives the same number or more records  $\langle p_{r2}, l_{w_x} \rangle_k$  of a non-effective read than records  $\langle p_{r1}, l_{w_x} \rangle_k$  of an effective read. For instance, an audit quorum composed of objects  $G_{2-6}$  receives  $\tau + f - 1$  records  $\langle p_{r2}, l_{w_x} \rangle_k$  and  $\tau - f - 1$  records  $\langle p_{r1}, l_{w_x} \rangle_k$ . Defining  $\ell = \tau - f - 1$  is enough to report the effective read from





```

1: function  $a\text{-audit}()$ 
2:    $E \leftarrow \emptyset$ 
3:    $L[1..n] \leftarrow \emptyset$ 
4:   parallel for  $1 \leq k \leq n$  do
5:      $L[k] \leftarrow o_k.\text{getLog}()$ 
6:   end parallel for
7:   wait  $|\{k : L[k] \neq \perp\}| \geq q$ 
8:   for all  $p_r, l_{w_v}$  do
9:      $\mathcal{E}_{p_r, v} \leftarrow \bigcup_{k \in \{1..n\}} \{ \langle p_r, l_{w_v} \rangle_k \in L[k] \}$ 
10:    if  $|\mathcal{E}_{p_r, v}| \geq \ell$  then
11:       $E \leftarrow E \cup \{ \mathcal{E}_{p_r, v} \}$ 
12:    end if
13:  end for
14:  return  $E$ 
15: end function

```

(Algorithm 1)

Figure 3: (a) A scenario where  $\ell < \tau + f$  does not guarantee strong accuracy. (b) A generic scenario that proves it is impossible to satisfy both completeness and strong accuracy of auditable registers. (Algorithm 1) The  $a\text{-audit}()$  algorithm.

$p_{r1}$ , but it violates the strong accuracy by also reporting the non-effective read from  $p_{r2}$ . As a consequence, it is impossible to define a single reporting threshold  $\ell$  to be used to produce evidences without violating either completeness or strong accuracy.  $\square$

Remark: Adding any number of objects to the scenario of Figure 3b does not change the impossibility result of Theorem 2. The reason is that with the unlimited concurrency in our model, each additional object may have a value  $v_i$  different from all values stored in the other objects.

## 6 Audit Algorithm

We present a generic auditability algorithm for registers implemented on top of available  $f$ -threshold quorum systems to prove that all bounds from the previous section (Lemmata 2–4 and Theorem 1) are tight in our system model. In the present section, we prove that this algorithm satisfies the completeness property with  $\tau \geq 2f + 1$  (Lemma 5) and the weak accuracy with  $\ell \geq f + 1$  (Lemma 6). Then, we prove that it supports both completeness and weak accuracy with  $\tau \geq 3f + 1$  (Theorem 3). Finally, we prove it supports the strong accuracy property alone with  $\ell \geq \tau + f$  (Lemma 7).

The implementation for  $a\text{-audit}()$  is presented in Algorithm 1. It starts with an empty set  $E$  that will be used to store the evidences attesting the effective reads (Line 2). It then queries  $n$  storage objects to obtain the list of records on their logs (i.e.,  $o_k.\text{getLog}()$ ), waits the response from at least  $q$  of them, and stores these responses in the array of logs  $L$  (Lines 4–7).

For each previously seen record, it verifies if the number of different logs  $L[k]$  containing records for the same reader  $p_r$  and value  $v$  is at least  $\ell$ . In doing so, it adds the evidence  $\mathcal{E}_{p_r, v}$  to the reporting set of evidences  $E$  (Lines 8–11). After verifying all records, the audit operation returns the set  $E$  (Line 14), which is used by auditors to report that the detected readers have effectively read the mentioned data values.

**Lemma 5.** *Algorithm 1 satisfies the **completeness** property of auditable registers with  $\tau \geq 2f + 1$ .*

*Proof.* Let us assume  $\tau \geq 2f + 1$ . Then in this case, based on Corollary 1 (i.e.,  $\ell \leq \tau - 2f$ ), any audit quorum  $Q_a$  receives at least one record from a correct, up-to-date storage object that has participated on each effective read (i.e.,  $\ell \geq 1$ ). Consequently, Algorithm 1 satisfies completeness by obtaining the records from any audit quorum  $Q_a$  if  $\tau \geq 2f + 1$ .  $\square$

**Lemma 6.** *Algorithm 1 satisfies the **weak accuracy** of auditable registers with  $\ell \geq f + 1$ .*

*Proof.* To support the weak accuracy of auditable registers, Algorithm 1 simply needs to make the  $f$  records from malicious objects insufficient to create an evidence reporting an effective read if a correct reader has never invoked a read on any correct object. Therefore,  $\ell \geq f + 1$  ensures that at least one correct object has also received a read request from this client.  $\square$

**Theorem 3.** *Algorithm 1 satisfies both **completeness** and **weak accuracy** of auditable registers with  $\tau \geq 3f + 1$ .*

*Proof.* Assuming  $\tau \geq 3f + 1$  directly satisfies completeness (Lemma 5). Based on Corollary 1 (i.e.,  $\ell \leq \tau - 2f$ ), any audit quorum  $Q_a$  receives at least  $f + 1$  records from correct, up-to-date storage objects that have participated on each effective read. As proved in Lemma 6,  $\ell \geq f + 1$  is enough for Algorithm 1 to satisfy weak accuracy.  $\square$

A consequence of Theorem 3 and Proposition 2 is that existing information dispersal schemes that support fast reads, such as Hendricks *et. al* [16], require  $n \geq 5f + 1$  to support auditability.

**Lemma 7.** *Algorithm 1 satisfies the **strong accuracy** of auditable registers with  $\ell \geq \tau + f$ .*

*Proof.* To support the strong accuracy property of auditable registers, evidences must be created using at least  $\tau$  records for the same value from correct, up-to-date storage objects that have participate in the read—where  $\tau > f$  (Proposition 1). Since  $f$  malicious servers can also participate in any audit quorum  $Q_a$ , this number must be added to the minimum number of records required to create an evidence. These two requirements make strong accuracy satisfiable with  $\ell \geq \tau + f$  by accessing any audit quorum  $Q_a$  because, in this case, the  $f$  malicious objects make no difference when reporting a value effectively read by a correct reader.  $\square$

## 7 Alternative Models for the Algorithm

As already proved in Theorem 2, it is impossible to satisfy both completeness and strong accuracy of auditable registers in our model. In this section, we present two modifications on our assumptions that allow Algorithm 1 to overcome the lower bound in Lemma 4 and satisfy both completeness and strong accuracy.

### 7.1 Sequential Operations

Serializing operations using total order broadcast [8] allows our system to execute operations sequentially. By doing so, we limit the number of different values present in storage objects since high-level read and write operations are executed in total order. In the worst case, the system will have  $f$  objects with a correct but stale value and  $f$  malicious objects with arbitrary values.

With this limitation, strong accuracy becomes easier, and then it can be satisfied together with completeness.

**Lemma 8.** *Totally ordering operations in our model allows Algorithm 1 to satisfy **strong accuracy** with  $\ell \geq 2f + 1$ .*

*Proof.* If there is no concurrent operations in the system, there will be always at most  $f$  correct stale storage objects and  $f$  malicious ones. The worst case is when the  $f$  stale objects participate in a read quorum  $Q_r$  and the  $f$  malicious objects mimic them, resulting in  $2f$  records for the read of a stale value. Requiring  $\ell \geq 2f + 1$  to create an evidence guarantees that it is achieved only when a reader obtains at least  $4f + 1$  correct blocks from the same data value, which can only happen with the most up-to-date value.  $\square$

**Theorem 4.** *Totally ordering operations in our model allows Algorithm 1 to satisfy both **completeness** and **strong accuracy** with  $\tau \geq 4f + 1$ .*

*Proof.* With  $\tau \geq 4f + 1$  and based on Lemma 1, any audit quorum  $Q_a$  receives at least  $2f + 1$  records from correct, up-to-date storage objects that have participated on each effectively read value  $v$ . As proved in Lemma 8,  $\ell \geq 2f + 1$  is enough to satisfy strong accuracy in our model when there is no concurrency. Moreover, this limitation does not change the lower bound of Lemma 5, which means that  $\tau \geq 4f + 1$  is enough for Algorithm 1 to also satisfy completeness in our model.  $\square$

## 7.2 Non-fast Reads

There are read algorithms that use more than one communication round (i.e., a *non-fast read* [12]) to ensure correct readers will only fetch, from correct objects, the blocks of the most up-to-date value stored in the register. For instance, DepSky-CA [6] is a register emulation in which the first round of a read obtains the identifier of the most up-to-date value available in a quorum of objects, while the second round actually reads the coded blocks for that value. As a consequence, correct objects log reads from correct clients only if they hold the most up-to-date value available in a quorum of objects.

**Lemma 9.** *Applying Algorithm 1 to DepSky-CA protocol satisfies **strong accuracy** with  $\ell \geq f + 1$ .*

*Proof.* Let us assume registers follow DepSky-CA protocol [6] with two communication rounds in read operations and  $\tau \geq 3f + 1$ . No read operation will result in correct records for more than one data value and every read operation results in the creation of records in at least a quorum of objects. In the worst-case scenario, the  $f$  malicious storage objects can only forge at most  $f$  records for another value. Requiring  $\ell \geq f + 1$  to create an evidence and report an effective read guarantees that it is achieved only when the reported value was actually read in a quorum of objects.  $\square$

**Theorem 5.** *Applying Algorithm 1 to DepSky-CA protocol satisfies both **completeness** and **strong accuracy** with  $\tau \geq 3f + 1$ .*

*Proof.* Let us assume registers follow DepSky-CA protocol [6] with two communication rounds in read operations. Assuming  $\tau \geq 3f + 1$  and based on Lemma 1, any audit quorum  $Q_a$  receives at least  $f + 1$  records from correct, up-to-date storage objects that have participated on each effective read. As proved in Lemma 9,  $\ell \geq f + 1$  is enough to satisfy strong accuracy when using DepSky-CA algorithm. Moreover, this limitation does not change the lower bound of Lemma 5, which means that  $\tau \geq 3f + 1$  is also enough to satisfy completeness.  $\square$

## References

- [1] Ittai Abraham, Gregory V Chockler, Idit Keidar, and Dahlia Malkhi. Byzantine disk paxos: optimal resilience with Byzantine shared memory. In *Proc. of the 23th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 226–235, 2004.
- [2] Marcos K Aguilera, Burkhard Englert, and Eli Gafni. On using network attached disks as shared memory. In *Proc. of the 22nd Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 315–324, 2003.
- [3] Elli Androulaki, Christian Cachin, Dan Dobre, and Marko Vukolić. Erasure-coded Byzantine storage with separate metadata. In *Proc. of the 18th International Conference on Principles of Distributed Systems (OPODIS)*, pages 76–90, 2014.
- [4] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM (JACM)*, 42(1):124–142, 1995.
- [5] Alon Berger, Idit Keidar, and Alexander Spiegelman. Integrated bounds for disintegrated storage. In *Proc. of the 32nd International Symposium on Distributed Computing (DISC)*, pages 11:1–11:18, 2018.
- [6] Alysson Bessani, Miguel Correia, Bruno Quaresma, Fernando Andre, and Paulo Sousa. Dep-Sky: Dependable and secure storage in cloud-of-clouds. *ACM Transactions on Storage (TOS)*, 9(4):12:1–12:33, 2013.
- [7] Viveck R Cadambe, Zhiying Wang, and Nancy Lynch. Information-theoretic lower bounds on the storage cost of shared memory emulation. In *Proc. of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 305–313, 2016.
- [8] M. Castro and B. Liskov. Practical Byzantine Fault-Tolerance and Proactive Recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.
- [9] Gregory Chockler, Rachid Guerraoui, and Idit Keidar. Amnesic distributed storage. In *Proc. of the 21st International Symposium on Distributed Computing (DISC)*, pages 139–151, 2007.
- [10] Gregory Chockler and Alexander Spiegelman. Space complexity of fault-tolerant register emulations. In *Proc. of the 36th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 83–92, 2017.
- [11] Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation). *Official Journal of the European Union*, L119:1–88, 2016.
- [12] 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.
- [13] Andreas Haeberlen. A case for the accountable cloud. *ACM SIGOPS Operating Systems Review*, 44(2):52–57, 2010.

- [14] Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel. PeerReview: Practical accountability for distributed systems. In *Proc. of the 21th ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 175–188, 2007.
- [15] Luke Harding. *The Snowden files: The inside story of the world’s most wanted man*. Guardian Faber Publishing, 2014.
- [16] 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.
- [17] Maurice P. Herlihy. Impossibility and universality results for wait-free synchronization. In *Proc. of the 7th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, pages 276–290, 1988.
- [18] Ari Juels and Burton S Kaliski Jr. PORs: proofs of retrievability for large files. In *Proc. of the 14th ACM Conference on Computer and Communications Security (CCS)*, pages 584–597, 2007.
- [19] Ryan KL Ko, Bu Sung Lee, and Siani Pearson. Towards achieving accountability, auditability and trust in cloud computing. In *Proc. of the 1st International Conference on Advances in Computing and Communications (ACC)*, pages 432–444, 2011.
- [20] Manjur Kolhar, Mosleh M Abu-Alhaj, and Saied M Abd El-atty. Cloud data auditing techniques with a focus on privacy and security. *IEEE Security & Privacy*, 15(1):42–51, 2017.
- [21] Hugo Krawczyk. Secret sharing made short. In *Proc. of the 13th Annual International Cryptology Conference on Advances in Cryptology (CRYPTO)*, pages 136–146, 1993.
- [22] Leslie Lamport. On interprocess communication. *Distributed Computing*, 1(2):86–101, 1986.
- [23] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.
- [24] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.
- [25] Raluca Ada Popa, Jacob R Lorch, David Molnar, Helen J Wang, and Li Zhuang. Enabling security in cloud storage SLAs with CloudProof. In *Proc. of the USENIX Annual Technical Conference (ATC)*, pages 355–368, 2011.
- [26] Michael Rabin. Efficient dispersal of information for security, load balancing, and fault tolerance. *Journal of the ACM (JACM)*, 36(2):335–348, 1989.
- [27] Luigi Rizzo. Effective erasure codes for reliable computer communication protocols. *ACM SIGCOMM Computer Communication Review (CCR)*, 27(2):24–36, 1997.
- [28] Optimum Security. The 21 biggest data breaches of the 21st century. <https://optimumsecurity.ca/21-biggest-data-breach-of-21-century>, 2019.
- [29] Adi Shamir. How to share a secret. *Communications of the ACM (CACM)*, 22(11):612–613, 1979.

- [30] Daniel Slamanig and Christian Hanser. On cloud storage and the cloud of clouds approach. In *Proc. of the 7th International Conference for Internet Technology And Secured Transactions (ICITST)*, pages 649–655, 2012.
- [31] Alexander Spiegelman, Yuval Cassuto, Gregory Chockler, and Idit Keidar. Space bounds for reliable storage: Fundamental limits of coding. In *Proc. of the 35th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 249–258, 2016.
- [32] Emil Stefanov, Marten van Dijk, Ari Juels, and Alina Oprea. Iris: A scalable cloud file system with efficient integrity checks. In *Proc. of the 28th ACM Annual Computer Security Applications Conference (ACSAC)*, pages 229–238, 2012.
- [33] Hassan Takabi, James BD Joshi, and Gail-Joon Ahn. Security and privacy challenges in cloud computing environments. *IEEE Security & Privacy*, 8(6):24–31, 2010.
- [34] Cong Wang, Sherman SM Chow, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for secure cloud storage. *IEEE Transactions on Computers (TC)*, 62(2):362–375, 2013.
- [35] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *Proc. of the IEEE INFOCOM*, pages 1–9, 2010.
- [36] Qian Wang, Cong Wang, Kui Ren, Wenjing Lou, and Jin Li. Enabling public auditability and data dynamics for storage security in cloud computing. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 22(5):847–859, 2011.
- [37] Aydan R Yumerefendi and Jeffrey S Chase. The role of accountability in dependable distributed systems. In *1st Workshop on Hot Topics in System Dependability (HotDep)*, 2005.