# Improving the Hu-Toueg Construction of a Byzantine Linearizable SWMR Register

Ajay D. Kshemkalyani[1]*[0000−0003−2451−7306], Manaswini Piduguralla[2], Sathya Peri[3], and Anshuman Misra[4]

[1] University of Illinois Chicago, USA
ajay@uic.edu
[2] Indian Institute of Technology Hyderabad, India
cs20resch11007@iith.ac.in
[3] Indian Institute of Technology Hyderabad, India
sathya_p@cse.iith.ac.in
[4] Purdue University Fort Wayne, USA
misra47@pfw.edu

**Abstract.** Recently, Hu and Toueg gave an implementation of the SWMR register from SWSR registers in shared memory distributed systems. While their definition of register linearizability is consistent with the definition of Byzantine linearizability of a concurrent history of Cohen and Keidar, it has several drawbacks. A stronger definition of a Byzantine linearizable register that overcomes these drawbacks has been proposed. In this paper, we give a construction of a Byzantine linearizable SWMR atomic register from SWSR registers that meets the stronger definition. The construction is correct when $n > 3f$, where $n$ is the number of readers, $f$ is the maximum number of Byzantine readers, and the writer can also be Byzantine. The construction relies on a public-key infrastructure.

**Keywords:** Byzantine fault-tolerance · Register construction · Shared memory · SWMR Register

## 1 Introduction

We consider the problem of implementing a single-writer multi-reader (SWMR) register from single-writer single-reader (SWSR) registers in a system with Byzantine processes. Recently, Hu and Toueg gave an implementation of the SWMR register from SWSR registers [4,5]. While their definition of register linearizability is consistent with the definition of Byzantine linearizability of a concurrent history of Cohen and Keidar [2], the problem formulation and results in [2,4,5] have the following drawbacks, identified and described in [8].

1. If the writer is Byzantine, the register is vacuously linearizable no matter what values the correct readers return. Reads by correct processes can return any value whatsoever including the initial value while the register meets their

* Corresponding author

definition of linearizability. In particular, there is no view consistency. For example, in the Hu-Toueg algorithm, consider a scenario where a Byzantine writer writes a different data value associated with the same counter value to the various readers' SWSR registers. The correct readers will return different data values associated with the same counter value, thus having inconsistent views.
2. Their definition of register linearizability does not factor in, or ignores, those values written by a Byzantine writer, by honestly following the writer protocol for those values. We need a stronger notion of a *correct write operation* that factors in such values as being written correctly. Also, note that the Byzantine writer is in control of the execution both above and below the SWMR register interface and hence the value that it writes in a correct write operation can be assumed to be the value intended to be written (correctly) and not altered by Byzantine behavior.
3. Their definition of register linearizability allows a value written by a Byzantine writer to just a single reader's SWSR register to be returned by a correct process. In order to validate that the writer intended to write that value honestly, we would like a minimum threshold number of readers' SWSR registers to be written that same value to enable that value to become eligible for being returned to a correct reader. This validates the intention of the Byzantine writer to write that particular value.
4. In their definition of register linearizability, their notion of a "current" value returned by a correct reader is not related to the most recent value written by a correct write operation of a Byzantine writer. We need a more up to date version of the value that can be returned by a correct reader. This helps give a stronger guarantee of progress from the readers' perspective.

The definition of a Byzantine linearizable register given in [8] is stronger than that of [2, 4, 5] and overcomes the above drawbacks. Further, we are interested in implementing the SWMR register over SWSR registers directly in the shared memory model.

**Contributions.** We first review the stronger definition of a *Byzantine linearizable register* of [8] that overcomes all the above drawbacks of [2,4,5]. We examine their concept of a *correct write operation* by a Byzantine writer as one that conforms to the write protocol. We also examine their notion of a *pseudo-correct write operation* by a Byzantine writer, which has the effect of a correct write operation. Only correct and pseudo-correct writes may be returned by correct readers. The correct and pseudo-correct writes are totally ordered by the logical timestamps issued by the writer.

The main contribution is giving a construction of a Byzantine linearizable SWMR atomic register from SWSR atomic registers that meets the stronger definition of [8]. The construction is correct when $n > 3f$, where $n$ is the number of readers, $f$ is the maximum number of Byzantine readers, and the writer can also be Byzantine. The construction relies on a public-key infrastructure (PKI). The construction develops the idea of the readers validating the logical timestamp of the writing of the values set aside for them by the writer. A sufficient number of correct readers will validate this consistently, and that forms the basis of the total order used to ensure Byzantine register linearizability. As compared to the algorithm in [4, 5] which can tolerate any number of Byzantine readers, our algorithm requires $f < n/3$. Also, in the algorithm in [4, 5], a reader that stops reading also stops taking implementation steps whereas our algorithm requires a reader helper thread to take infinitely many steps even if it has no read operation to apply. The algorithm in [4, 5] as well as our algorithm use a PKI. The algorithm in [4, 5] uses $n^2$ shared SWSR registers, whereas our algorithm uses $2n^2 + 2n$ shared SWSR registers.

**Related Work.** The problem of implementing shared registers from weaker types of registers has been extensively studied, see [8] for a survey. The SWMR register in a Byzantine setting is of great importance in recent research. For example, Mostefaoui et al. [10] prove that in message-passing systems with Byzantine failures, there is a $f$-resilient implementation of a SWMR register if and only if $f < n/3$ processes are faulty, where $f$ is the number of Byzantine processes and $n$ is the total number of processes. It was the first to give the definition of a linearizable SWMR register in the presence of Byzantine processes and [2] generalized it to objects of any type. Aguilera et al. [1] use atomic SWMR registers to solve some agreement problems in hybrid systems subject to Byzantine process failures. Cohen and Keidar [2] give $f$-resilient implementations of three objects – asset transfer, reliable broadcast, atomic snapshots – using atomic SWMR registers in systems with Byzantine failures where at most $f < n/2$ processes are faulty. Their implementations were based on their definition of Byzantine linearizability of a concurrent history.

In other related work, a SWMR register was built above a message-passing system where processes communicate using send/receive primitives with the constraint that $f < n/3$ [6, 10]. These works do not use signatures. Unbounded history registers were required in [6] whereas [10] used $O(n^2)$ messages per write operation. Although building SWMR registers over SWSR registers or over message-passing systems is equivalent as SWSR registers can be emulated over send/receive and vice versa, this is a round-about and expensive solution and there are fundamental differences between the shared memory paradigm and the message-passing paradigm.

**Roadmap:** Section 2 gives the system model and preliminaries. Section 3 reviews the characterization and definition of a Byzantine linearizable register, from [8]. Section 4 gives our construction of the SWMR Byzantine linearizable register using SWSR registers. Section 5 gives the correctness proof. Section 6 gives a discussion.

## 2 System Model

**Model Basics.** We consider the shared memory model of a distributed system. The system contains a set of asynchronous processes. These processes access some shared memory objects. All inter-process communication is done through an API exposed by the objects. Processes invoke operations that return some response to the invoking process. We assume reliable shared memory but allow for an adversary to corrupt up to $f$ processes in the course of a run. A corrupted process is defined as being *Byzantine* and such a process may deviate arbitrarily from the protocol. A non-Byzantine process is *correct* and such a process follows the protocol and takes infinitely many steps.

We also assume a PKI. Each process has a public-private key pair used to sign data and verify signatures of other processes. A values $v$ signed by process $p$ is denoted $\langle v \rangle_p$.

We give an algorithm that emulates an object $O$, viz., a SWMR register from SWSR registers. We assume that there is adequate access control such that a SWSR register can be accessed only by the single writer and the single reader between whom the register is set up, and that another (Byzantine) process cannot access it. The algorithm is organized as methods of $O$. A method execution is a sequence of steps. It begins with the *invoke* step, goes through steps that access lower-level objects, viz., SWSR registers, and ends with a *return* step. The invocation and response delineate the method's execution interval. In an *execution* $\sigma$, each correct process invokes methods sequentially, and steps of different processes are interleaved. Byzantine processes take arbitrary steps irrespective of the protocol. The *history* $H$ of an execution $\sigma$ is the sequence of high-level invocation and response events of the emulated SWMR register in $\sigma$. A history $H$ defines a partial order $\prec_H$ on operations. $op_1 \prec_H op_2$ if the response event of $op_1$ precedes the invocation event of $op_2$ in $H$. $op_1$ is concurrent with $op_2$ if neither precedes the other.

In our algorithm, we assume that each reader process has a helper thread that takes infinitely many steps even if the reader stops reading the implemented register. These steps are outside the invocation-response intervals of the readers' own operations.

**Linearizability of a concurrent history.** *Linearizability*, a correctness condition for concurrent objects, is defined using an object's sequential specification.

**Definition 1.** *(Linearization of a concurrent history:) A* linearization *of a concurrent history $H$ of object $o$ is a sequential history $H'$ such that:*

1. *After removing some pending operations from $H$ and completing others by adding matching responses, it contains the same invocations and responses as $H'$,*
2. *$H'$ preserves the partial order $\prec_H$, and*
3. *$H'$ satisfies $o$'s sequential specification.*

A SWMR register as well as a SWSR register expose the *read* and *write* operations. The sequential specification of a SWMR and a SWSR register states that a read operation from register *Reg* returns the value last written to *Reg*. Following Cohen and Keidar [2], we manage Byzantine behavior in a way that provides consistency to correct processes. This is achieved by linearizing correct processes' operations and offering a degree of freedom to embed additional operations by Byzantine processes.

Let $H|_{correct}$ denote the projection of history $H$ to all correct processes. History $H$ is Byzantine linearizable if $H|_{correct}$ can be augmented by (some) operations of Byzantine processes such that the completed history is linearizable. Thus, there is another history with the same operations by correct processes as in $H$, and additional operations by at most $f$ Byzantine processes.

**Definition 2.** *(Byzantine linearization of a concurrent history [2]:) A history $H$ is Byzantine linearizable if there exists a history $H'$ such that $H'|_{correct} = H|_{correct}$ and $H'$ is linearizable.*

An object supports Byzantine linearizable executions if all of its executions are Byzantine linearizable. SWMR registers support Byzantine linearizable executions because before every read from such a register, invoked by a correct process, one can add a corresponding Byzantine write.

**Linearizability of Register Implementations.** Hu and Toueg defined register linearizability in a system with Byzantine processes as follows [4,5]. They let $v_0$ be the initial value of the implemented register and $v_k$ be the value written by the $k$th write operation by the writer $w$ of the implemented register.

**Definition 3.** *(*Register Linearizability [4,5]:*) In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following holds. If the writer is not malicious, then:*

- *(Reading a "current" value) If a read operation* `R` *by a process that is not malicious returns the value $v$ then:*
  - *there is a write $v$ operation that immediately precedes* `R` *or is concurrent with* `R`*, or*
  - *$v = v_0$ (the initial value) and no write operation precedes* `R`*.*
- *(No "new-old" inversion) If two read operations* `R` *and* `R'` *by processes that are not malicious return values $v_k$ and $v_{k'}$, respectively, and* `R` *precedes* `R'`*, then $k \leq k'$.*

## 3 Byzantine Register Linearizability

When a *high-level object (HLO)* is simulated or constructed using a *low-level object (LLO)*, there are two interfaces. A process interacts with the HLO through a *high-level interface (HLI)* through alternating invocations and matching responses. Between such a pair of matching invocation and response, the process interacts with the LLO through a *low-level interface (LLI)* using alternating invocations and responses. Such interactions are in software. For our problem, the HLO is the Byzantine-tolerant SWMR atomic register and the HLI is the read and write operation. The LLO is the SWSR atomic register and the LLI is also the read and write operation.

In the face of Byzantine readers as well as a Byzantine writer, a correct write operation has to be defined. We use $u$ or $v$ to refer to the data value written. If the $write(v)$ at the HLI is converted into multiple serial invocations of $write(v')$

(for different values of $v'$) and the protocol for each of these $v'$ is correctly followed, these various $write(v')$ are considered correct write operations because that sequence of write operations can be taken to be the values the writer writes or intended to write. This is because the invocation/response at the HLI is at a Byzantine process which controls the execution of code above the LLI and above the HLI. In a correct write operation, the code between the HLI and the LLI is followed correctly by the Byzantine process.

**Definition 4 ( [8]).** *A* correct (write) operation *is a (write) operation that follows the (write) protocol, but possibly with a different value than that passed down at the HLI.*

In the literature [2,4,5], any behavior of a Byzantine writer is allowable in the linearizability definition. A Byzantine writer is accommodated differently in [8] by introducing the concept of a *pseudo-correct write operation* (Definitions 5 and 10), which is a Byzantine write operation that has the effect of a correct write operation, i.e., whose actions that are visible to correct readers cannot be distinguished from the actions of a correct write operation by correct readers.

**Definition 5 ( [8]).** *A* pseudo-correct (write) operaton *is a (write) operation such that whatever steps the (writer) process performs in it and that results in a value being returned to correct readers, are indistinguishable to correct readers' executions below the HLI from an actual correct (write) operation's steps.*

A stronger definition of a Byzantine linearization of a concurrent history than Definition 2 of Cohen-Keidar is given in [8]. It tames the Byzantine behavior in a stronger way to provide consistency to correct processes. It linearizes the correct processes' operations and offers a *(more) limited* degree of freedom by way of embedding *only* correct and pseudo-correct write operations by Byzantine processes. History $H$ is Byzantine linearizable if $H|_{correct}$ can be augmented by (some) pseudo-correct and correct operations (and not any arbitrary operations) of Byzantine processes such that the completed history is linearizable.

**Definition 6 ( [8]).** *(Byzantine linearization of a concurrent history:) A history $H$ is Byzantine linearizable if there exists a history $H'$ containing correct and pseudo-correct write operations by Byzantine processes and writes and reads by correct processes such that $H'|_{correct} = H|_{correct}$ and $H'$ is linearizable.*

A counter-example to show that the Hu-Toueg algorithm does not have a Byzantine linearization for concurrent histories as per our Definition 6 is given in [8].

In general, there is a many-many mapping from HLO writes by a Byzantine process to pseudo-correct write operations. We define a *virtual invocation (response)* of a pseudo-correct write operation as occuring at the time of invocation (response) of the earliest (lastest) HLO write operation in which some step of the pseudo-correct write operation was taken. To construct $H'$ in Definition 6, we add a virtual invocation-response of a pseudo-correct write operation before an appropriately identified invocation-response of a read by a correct process.

Let $v^i$ be the value written by the $i$th correct or pseudo-correct write $W^i$ in a Byzantine linearization of a concurrent history (Definition 6), which is used in Definition 7, following the notation in [3]. Note that to determine $i$, $v^i$ and $W^i$ requires knowing what happened below the HLI and above the LLI because of the nature of pseudo-correct writes; but there is actually no need to determine $i$, $v^i$, and $W^i$.

**Definition 7 ( [8]).** *(Byzantine Linearizable Register:) In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following two properties are satisfied in a Byzantine linearization of a concurrent history.*

1. **Reading a current value:** *When a read operation* `R` *by a non-Byzantine process returns the value $v$:*
   (a) *if $v = v_0$ then no correct or pseudo-correct write operation precedes* `R`
   (b) *else if $v \neq v_0$ then $v$ was written by the correct or pseudo-correct write operation that immediately precedes* `R`*.*
2. **No "new-old" inversions:** *If read operations* `R` *and* `R'` *by non-Byzantine processes return values $v^i$ and $v^j$, respectively, and* `R` *precedes* `R'`*, then $i \leq j$.*

A pseudo-correct (write) operation looks like a correct write operation to correct readers; however the writer may still not follow the write protocol exactly. Taming a Byzantine write and making visible what a Byzantine write does as part of a pseudo-correct write is done by correct readers in their steps below the HLI and above the LLI. As the Byzantine writer may exit its write protocol prematurely, the linearization point of a pseudo-correct write may be after the invocation-response interval(s) of the HLI operations that triggered the pseudo-correct write.

## 4 The Algorithm

WLOG assume that there are $n$ SWSR registers $R\_init_{wi}$ writable by the single writer $w$ and readable by reader $i \in [1, n]$. The Byzantine writer can behave anyhow and can write different values to the SR registers, or write different values to different subsets of SR registers while not writing to some of them at all, or write multiple different values over time to some or all SR registers, as part of the same write operation. We assume that $f$ of the $n$ readers are Byzantine.

The writer writes $(k, u)$, where $k$ is a monotonically increasing sequence number and $u$ is a data value, to the various $R\_init_{wi}$. Although there is a many-many mapping from write operations issued to the HLI object interface to values written to the object, this does not pose any ambiguity because the different values that are returned to the correct readers have different logical timestamps, given by $k$.

Correct write operations are totally ordered in time. This total order is also the logical time ordering of their timestamps. *To be indistinguishable to correct readers below the HLI from correct write operations (to satisfy Definition 5), write operations whose values can be returned should (i) be written to a sufficient* *number of $R\_init_{wi}$, and (ii) be totally ordered along with the set of correct write operations, by their logical timestamps.* Based on this above *principle*, we proceed to redefine pseudo-correct write operations. (Note, different algorithms can instantiate Definition 5 in different ways based on their design of read and write operations below the HLI.)

**Definition 8 ( [8]).** *A* potential pseudo-correct write operation *of value $(k, v)$ is a write operation, timestamped $k$, that may not follow the write protocol but*

1. *there is a quorum of size $\geq n-2f$ indices $i$ of correct readers such that $(k, v)$ was written to $R\_init_{wi}$, and*
2. *$k > k'$ for all $(k', v')$ already read from these $R\_init_{wi}$.*

**Definition 9 ( [8]).** *A write operation* stabilizes *if its value can be returned by a correct reader.*

A correct write operation always stabilizes whereas a potential pseudo-correct write may stabilize, depending on run-time dynamic data races due to the asynchronous readers, steps of Byzantine readers and the Byzantine writer, and the algorithm. Only all write operations that stabilize have a linearization point.

**Definition 10 ( [8]).** *A* pseudo-correct write operation *is a potential pseudo-correct write operation that stabilizes.*

**Definition 11 ( [8]).** *(*Monotonicity/Total Order of stabilized write operation timestamps Property*:) The set of write operation timestamps that stabilize is totally ordered.*

If this property is satisfied, of any two potential pseudo-correct writes whose timestamps are the same, at most one can stabilize. Satisfying the property is key to prove that the algorithm implements a Byzantine linearizable register.

In the algorithm implementation, write of value $v$ is actually a write of tuple $\langle k, v\rangle$, where $k$ is a sequence number issued by the writer. Only correct and pseudo-correct writes may be returned by correct readers. A correct reader cannot distinguish between a correct and a pseudo-correct write operation. A pseudo-correct write operation $o_1(v_1)$ timestamped $k_1$ may lose a race due to asynchrony of process executions to a pseudo-correct or correct write operation timestamped $k_2$ where $k_1 < k_2$, in which case $v_1$ is not actually returned to any read operation and $o_1$ is deemed to have an *invisible* linearization point. The correct and pseudo-correct writes are totally ordered by their linearization points, and this order is the total order on the timestamp values $k$.

## 4.1 Basic Idea and Operation

Because the writer $w$ is Byzantine, a reader $i$ cannot unilaterally use the value in its register $R\_init_{wi}$ but needs to coordinate with other readers. But a reader may not invoke a read operation indefinitely. So we assume that each reader process has a reader helper thread that is always running and participates in this coordination. A correct write operation needs to wait for acknowledgements from the readers so that the write value is guaranteed to get written in the algorithm data structures and stabilize, and is not overwritten before it stabilizes. This guarantees progress by ensuring that the most recent correct write operation advances with time. The algorithm data structures are described in Algorithm 1. The reader helper thread is given in Algorithm 2. $w$ is the writer process and $P$ denotes the set of $n$ readers.

The writer writes $\langle k, u\rangle$, where $u$ is the value to be written and $k$ is a sequence number assigned by the writer, to all the reader registers $R\_init_{wi}$ and then waits for $n-f$ of the acknowledgement registers $R\_ack_{iw}$ to be written with this value.

The reader helper thread $(\forall p \in P)$ loops forever. In each iteration, it reads $R\_init_{wp}$ and if the value overwrites the earlier value $\langle k', u'\rangle$ where $k > k'$, it writes $\langle\langle k, u\rangle, p\rangle_p$ to all the registers $R\_witness_{pi}$. It then reads each $R\_witness_{ip}$ and if the logical time of an entry is larger than the previous logical time read, it stores the value in a local variable $T\_witness_i$. (If less than, or equal but the entry is different, $i$ is marked as Byzantine). If there are at least $n-f$ $T\_witness_q$ entries with identical $\langle k, u\rangle$ values then these $T\_witness_q$ entries are placed in the local set $Witness\_Set$. This $Witness\_Set$ is written to all $R\_final_{pi}$. $R\_ack_{pw}$ is updated with $\langle k, u\rangle$. All the $R\_final_{ip}$ are read and placed in set $Z$. All the elements in $Z$, i.e., $(\forall i)$ $R\_final_{ip}$ are totally ordered by a relation $\mapsto$, (to be defined in Definition 16), as will be proved in Theorem 5. The latest element in $Z$ is identified as $Y$ via a call to $Find\_Latest(Z)$ and if $Y$ is different from the local $Witness\_Set$, i.e., for a different $\langle k, u\rangle$ pair, (a) $Y$ is written to all $R\_final_{pi}$ and (b) $R\_ack_{pw}$ is updated with the common $\langle k, u\rangle$ occurring in the $T\_witness$ entries $\langle\langle k, u\rangle, l\rangle_l$ for $\geq n-f$ values of $l$ in the $Witness\_Set$ that is $Y$.

## 4.2 Instantiating Framework Definitions in the Algorithm

A reader sees the value written by the most recent correct or pseudo-correct write operation that precedes the read operation in a Byzantine linearization of the concurrent history. A write operation *stabilizes* (Definition 14) if the value can be returned by a correct read operation, i.e., when it is written to $R\_final_{p*}$ for all values of $*$ (in the form of an $Witness\_Set$ containing at least $n-f$ correctly signed $T\_witness_i$ entries).

A correct write operation always stabilizes because it waits for $n-f$ acknowledgements in $R\_ack_{*w}$ before completion, thereby allowing the value written to $R\_final_{p*}$ to be eligible for being returned by a read operation. A correct write operation corresponds to a sufficient condition for stabilization. Writing the same value to $n-2f$ correct readers' $R\_init_{wi}$ in a potential pseudo-correct operation is a necessary condition for stabilization. For the potential pseudo-correct write operation to become a pseudo-correct write operation, favorable outcome of run-time dynamic data races due to the asynchronous writer and readers and the collaboration by Byzantine readers are needed to allow the value being written to the other correct readers' $R\_init_{wi}$ to stabilize. The set of correct and

**Algorithm 1:** Constructing a linearizable SWMR atomic register. Data structures and READ, WRITE procedures. Code at process $p$.

```
Shared variables:
For all processes $i \in P$: $R\_init_{wi}$: atomic SWSR register $\leftarrow \langle 0, u_0 \rangle$
For all processes $i \in P$: $R\_ack_{iw}$: atomic SWSR register $\leftarrow \langle 0, u_0 \rangle$
For all processes $i$ and $j$ in $P$: $R\_witness_{ij}$: atomic SWSR register
  $\leftarrow \langle \langle 0, u_0 \rangle, i \rangle_i$
For all processes $i$ and $j$ in $P$: $R\_final_{ij}$: atomic SWSR register
  $\bigcup_{k \in L} \{ \langle \langle 0, u_0 \rangle, k \rangle_k \}$, where $|L| \geq n - f \wedge L \subseteq P$
Local variables:
variable of $w$: $c \leftarrow 0$
variable of $i \in P$: $k\_init\_latest \leftarrow 0$
variables of $j \in P$: For all processes $i \in P$: $T\_witness_i \leftarrow \langle \langle 0, u_0 \rangle, i \rangle_i$
variable of $j \in P$: $Witness\_Set \leftarrow \bigcup_{k \in L} \{ \langle 0, u_0 \rangle, k \rangle_k \}$, where
  $|L| \geq n - f \wedge L \subseteq P$
WRITE($u$):
$c = c + 1$
$W(\langle c, u \rangle)$
return
READ():
$R()$
return the value returned by the $R$ call
$W(\langle k, u \rangle)$:
for every process $i \in P$ do
    $R\_init_{wi} = \langle k, u \rangle$
$d = 0$
while $d < n - f$ do
    for each new $\langle k, u \rangle$ in $R\_ack_{iw}$ among $R\_ack$ registers do
        $d = d + 1$
return done
$R()$:
execute one iteration of the reader helper thread, Algorithm 2
return the value last written in $R\_ack_{pw}$
```

pseudo-correct writes is exactly the set of writes whose values stabilize (follows from Theorem 3).

We show (Corollary 2) that the set of all values that stabilize are totally ordered by the logical times of their writing to the readers' registers $R\_init_{w*}$. The timestamp of the writing of a value $\langle k, u \rangle$ that stabilizes is denoted $\langle k, u \rangle.\overline{T}$ and is defined as $k$. We say that for values $\langle k, v \rangle$ and $\langle k', v' \rangle$ that stabilize, $\langle k.v \rangle \mapsto \langle k., v' \rangle$ if and only if $\langle k, v \rangle.\overline{T} \mapsto \langle k', v' \rangle.\overline{T}$, i.e., $k < k'$. The total order $\mapsto$ on timestamps of values that stabilize is the total order on the linearization

**Algorithm 2:** Reader helper thread for constructing a linearizable SWMR atomic register. Code at process $p$.

```
1  reader helper thread:
2  while $true$ do
3      if $R\_init_{wp}(= \langle k,u \rangle)$ is newly written ($k > k\_init\_latest$) then
4          $k\_init\_latest \leftarrow k$
5          for each $i \in P$ do
6              $R\_witness_{pi} \leftarrow \langle \langle k,u \rangle, p \rangle_p$
7      for each $i \in P$ do
8          if $R\_witness_{ip}(= \langle \langle k,u \rangle, i \rangle_i)$ is newly written, i.e., $k > T\_witness_i.k$
             then
9              $T\_witness_i \leftarrow \langle \langle k,u \rangle, i \rangle_i$
10     if $\exists \geq n-f$ correctly signed latest elements $\langle \langle k,u \rangle, q \rangle_q$ $T\_witness_q$ then
11         add all these $\geq n-f$ elements $T\_witness_q$ to the emptyset
             $Witness\_Set$
12         for each $i \in P$ do
13             $R\_final_{pi} \leftarrow Witness\_Set$
14         $R\_ack_{pw} \leftarrow \langle k,u \rangle$
15     $Z \leftarrow \emptyset$
16     for each $i \in P$ do
17         $Z \leftarrow Z \cup \{R\_final_{ip}\}$
18     $Y \leftarrow Find\_Latest(Z)$
19     if $\langle k,u \rangle$ common to entries in $Y \neq \langle k,u \rangle$ common to entries in
         $Witness\_Set$ then
20         for each $i \in P$ do
21             $R\_final_{pi} \leftarrow Y$
22         $R\_ack_{pw} \leftarrow$ the common $\langle k,u \rangle$ value occurring in identical $T\_witness$
             entries $\langle \langle k,u \rangle, l \rangle_l$ for $\geq n-f$ values of $l$ in $Witness\_Set$ that is $Y$
23 Find_Latest($Z$):
24 for each $Witness\_Set$ $Z_i \in Z$ do
25     for each $T\_witness$ element $x$ in $Z_i$ that fails signature test do
26         $Z_i \leftarrow Z_i \setminus \{x\}$; if $|Z_i| < n-f$ then $Z \leftarrow Z \setminus \{Z_i\}$ and break()
27 while $|Z| > 1$ do
28     let $Z_i, Z_j$ be any two elements of $Z$; $Z_i$ ($Z_j$) has $\geq n-f$ entries and is the
         $Witness\_Set$ of some reader
29     For the $\geq n-2f$ readers $q \in Z_i \cap Z_j$, let $e^i_q = \langle \langle k,u \rangle, q \rangle_q$ and
         $e^j_q = \langle \langle k',u' \rangle, q \rangle_q$ be these $T\_witness$ entries in the $Witness\_Set$s that
         are $Z_i$ and $Z_j$, resp.
30     if $k \geq k'$ for any/all processes $q$ then
31         $Z \leftarrow Z \setminus Z_j$
32     else
33         $Z \leftarrow Z \setminus Z_i$
34 return(element in $Z$)
```

points of correct and pseudo-correct write operations. This follows from the *no "new-old" inversions* clause in Theorem 8.

More than one value can stabilize as part of the same HLO write operation. This can happen when, for example, some of the readers' registers could have been written to in earlier Write operations or a HLO invocation-response of a Byzantine write contains multiple pseudo-correct write operations.

With the introduction of the $\mapsto$ relation, our definition of Byzantine linearizable register (Definition 7) is adapted to the algorithm by rephrasing the *No "new-old" inversions* property as follows.

**Definition 12.** *(Byzantine Linearizable Register in our algorithm). In a system with Byzantine process failures, an implementation of a SWMR register is linearizable if and only if the following two properties are satisfied in a Byzantine linearization of a concurrent history.*

1. **Reading a current value:** *When a read operation* `R` *by a non-Byzantine process returns the value $v$:*
   *(a) if $v = v_0$ then no correct or pseudo-correct write operation precedes* `R`
   *(b) else if $v \neq v_0$ then $v$ was written by the correct or pseudo-correct write operation that immediately precedes* `R`*.*
2. **No "new-old" inversions:** *If read operations* `R` *and* `R'` *by non-Byzantine processes return values $\langle k, v\rangle$ and $\langle k', v'\rangle$, respectively, and* `R` *precedes* `R'`*, then $\langle k, v\rangle.\overline{T} \overset{=}{\mapsto} \langle k', v'\rangle.\overline{T}$, i.e., $k \leq k'$.*

## 5 Correctness Proof

**Definition 13.** *The witness set that is formed, denoted $WS(k, u)$, is the maximal set of at least $n - f$ $T\_witness$ entries $\langle\langle k, u\rangle, l\rangle_l$ in the $Witness\_Set$ $WS$.*

**Definition 14.** *The field/value $\langle k, u\rangle$ common to all the entries of a $Witness\_Set$ is defined to stabilize when the $Witness\_Set$ is written to all the $R\_final_{pi}$ for some process $p$.*

**Definition 15.** *For a value $\langle k, u\rangle$ of $Witness\_Set$ $WS$ that stabilizes, $\langle k, u\rangle.\overline{T}$ is the value $k$.*

Only a value that stabilizes may be returned by a correct reader.

**Theorem 1.** *A correct write operation is guaranteed to stabilize provided $n > 2f$.*

*Proof.* A correct write operation writes the same $\langle k, u\rangle$ to $R\_init_{wp}$ for all correct $p$ and will not complete unless it gets $n - f$ acks in $R\_ack_{pw}$. It may get $f$ acks from Byzantine processes but as $n > 2f$, it will need an ack from at least one correct process. A correct process $p$ gives the ack only after it has written the value to $R\_final_{p*}$, i.e., when the value has stabilized. We now show that at least one correct process will have the value stabilize, before which the value written to the $R\_init_{w*}$ will not be overwritten.

For all correct $p$, the witness timestamps $\langle\langle k,u\rangle, p\rangle$, signed by $p$, are correctly written to $R\_witness_{p*}$. At least $n-f$ correct reader helper threads of correct processes $p$ will eventually read from $R\_witness_{*p}$, form their $Witness_Set$s for that $\langle k,u\rangle$, and write their own $Witness\_Set$ to $R\_final_{p*}$. Such reader threads $p$ will then write $\langle k,u\rangle$ to $R\_ack_{pw}$. After $n-f$ acks have been written by the correct and/or Byzantine reader processes/threads, the correct write operation will complete. Thus, $\langle k,u\rangle$ is guaranteed to have stabilized as the $R\_init_{wi}(\forall i)$ will not be overwritten until then. □

**Theorem 2.** *A potential pseudo-correct write operation may stabilize provided* $n > 2f$.

*Proof.* A value written by a potential pseudo-correct operation writes the same value to at least $n-2f$ correct readers' $R\_init_{wp}$, across possibly multiple prior write operations and the current write operation. These reader processes write that value, if not overwritten, to $R\_witness_{p*}$. Let the $f$ Byzantine processes $b$ read the value from their $R\_witness_{*b}$ and thereafter behave as though the value had been written to their $R\_init_{wb}$ and thenceforth behave correctly. There is now a way that the value may stabilize if the Byzantine writer does not overwrite the values in $R\_init_{wp}$ $(\forall p)$ until at least one process $q$ forms its $Witness\_Set$ of correctly signed $Witness\_Set$ entries for that value $\langle k,u\rangle$ and writes the $Witness\_Set$ to $R\_final_{q*}$. This condition will be satisfied as per the logic in the second paragraph of the proof of Theorem 1. □

**Theorem 3.** *If a value stabilizes, it must have been written by a correct write operation or by a potential pseudo-correct write operation.*

*Proof.* A correct write operation stabilizes (Theorem 1). So we need to only prove the following contrapositive, namely that if a write is not a potential pseudo-correct write operation, it will not stabilize.

If a write is not a potential pseudo-correct operation, it is not written to at least $n-2f$ correct processes' $R\_init_{wi}$ across possibly multiple write operations. Then there is no way a $Witness\_Set$ of $n-f$ correctly signed $Witness\_Set$ entries can form at any process, correct or Byzantine, and can be written to any $R\_final_{p*}$. If a Byzantine process attempts to write a fake $Witness\_Set$ in $R\_final_{p*}$, that will be detected by correct processes as the entries in that $Witness\_Set$ written in $R\_final_{p*}$ will not pass the signature test. Hence that value is deemed not to have stabilized. □

We next formalize the liveness/progress conditions for the advancement of the latest correct/pseudo-correct write operation. Define an *epoch* to be the interval in which each correct reader helper thread executes an iteration of its main *while* loop at least once.

**Theorem 4.** *From the time of writing the same* $\langle k,u\rangle$ *to at least* $n-f$ $R\_init_{wi}$ *for correct processes* $i$ *and such that* $k > k'$ *for all previously read* $\langle k',u'\rangle$ *from these* $R\_init_{wi}$*, if the Byzantine writer does not overwrite these* $n-f$ $R\_init_{wi}$ *for two epochs then the* $\langle k,u\rangle$ *tuple is guaranteed to stabilize.*

*Proof.* In the first epoch, the correct readers $i$ read $R\_init_{wi}$ and write the read tuples (after signing the corresponding tuple) to $R\_witness_{i*}$. In the second epoch, the correct readers $i$ read $R\_witness_{*i}$, form the respective $Witness_Set$s, and write it to $R\_final_{i*}$. At this time the $\langle k, u\rangle$ has stabilized (Definition 14). □

**Observation.** A correct process forms its successive $Witness_Set$s only in increasing order of source witness timestamps for the at least $n-2f$ common witnesses as it writes these $Witness_Set$s to $R\_final$.

**Definition 16.** *Given* $WS1(= WS(k,u))$ *and* $WS2(= WS(k',u'))$*,* $WS1 \mapsto WS2$ *iff* $(k,u).\overline{T} < (k',u').\overline{T}$*, i.e.,* $k < k'$*.*

If $WS(k,u) \mapsto WS(k',u')$, we also interchangeably say that for the corresponding values, $\langle k,u\rangle \mapsto \langle k',u'\rangle$.

**Definition 17.** *Given distinct* $WS1(= WS(k,u))$ *and* $WS2(= WS(k',u'))$*,* $WS1||WS2$ *iff* $WS1 \not\mapsto WS2 \wedge WS2 \not\mapsto WS1$*.*

**Theorem 5.** *For* $Witness\_Sets$ $WS1 = WS(k,u)$ *and* $WS2 = WS(k',u')$ *at any two (possibly different) reader processes,* $WS1 \mapsto WS2 \vee WS2 \mapsto WS1$*, i.e.,* $\neg(WS1\,||\,WS2)$*, provided* $n > 3f$*.*

*Proof.* There are $n-f$ signed $T\_witness$ entries in each of $WS1$ and $WS2$. For these to be valid, at least one must be from a correct reader, requiring $n-f > f$, implying $n > 2f$. However this is not sufficient.

When $n = 2f+1$, to prevent $f$ Byzantine processes from giving same witness timestamp inputs but for different values of $v$ ($v_\alpha$, for $\alpha = [1, n-f]$), forming $n-f = f+1$ $Witness_Set$s with identical common timestamps but different values $v_\alpha$ corresponding to each of $f+1$ correct processes, that were written to $R\_init_{wp}$, implying $f+1$ mutually concurrent $Witness_Set$s, we require that at least one correct process provide witness timestamps for both of any pair of $Witness_Set$s. This will totally order the $Witness_Set$s. To achieve this, the minimum number of common reader processes that provided inputs for any pair of $Witness_Set$s, $n-2f$ (as $f$ do not provide input to $WS1$ and some other $f$ do not provide input to $WS2$), should be greater than $f$, implying $n > 3f$. Thus $\neg(WS1||WS2)$.

For any two $Witness_Set$s that form (and stabilize), there is at least one correct process that provides witness timestamp inputs to both sets. From the pseudo-code, it will necessarily do so in increasing order of timestamps. For either $WS(k,u)$ or $WS(k',u')$, inputs from all processes must necessarily be for the same input ($(k,u)$ or $(k',u')$, respectively). Thus if $WS1$ and $WS2$ form, all processes that provide input witness timestamps to both $WS1$ and $WS2$ provide first to $WS1$ and then to $WS2$ or all provide first to $WS2$ and then to $WS1$. Furthermore, all such processes provide input first for the smaller of $\{k, k'\}$ and then for the larger. □

**Corollary 1.** *$WS1 \mapsto WS2$ implies that for all the at least $n-2f$ readers $z$ that witnessed values in both $WS1$ and $WS2$, $z$'s $WS1$ timestamp is less than $z$'s $WS2$ timestamp.*

Only value $\langle k, v\rangle$ corresponding to an $Witness\_Set$ that is written to $R\_final_{p*}$ could be returned by a correct process if the signed $Witness\_Set$ entries pass the signature test. Each new value returned by a correct reader is a new value genuinely written to at least $n-2f$ correct reader $i$'s $R\_init_{wi}$ and not falsely reported by Byzantine readers, in addition to that same value being the most recent value in a total of $n-f$ readers $j$'s $R\_init_{wj}$ registers as per $\langle k, u\rangle.\overline{T}$.

From Theorem 5, all the $\langle k, v\rangle.\overline{T}$ form a total order based on $\mapsto$. This leads to the following corollary.

**Corollary 2.** *The timestamps $\langle k, u\rangle.\overline{T}$ of $\langle k, u\rangle$ values that stabilize are totally ordered, provided $n > 3f$.*

We give a linearization of an execution to show that the SWMR register we constructed supports Byzantine linearizability of executions (Definition 6). First, read operations of correct readers are linearized in the order of the return values, as per the $\mapsto$ relation. Then an update of a value $\langle k, v\rangle$ by the Byzantine writer is added just before the first read operation that reads that value. In general, the value returned by a read operation is based on the values written by one or more than one HLI write operation as the writer is Byzantine. Further, one HLI write operation by the Byzantine writer may result in different read values being returned by multiple correct readers. To differentiate among the multiple updates of different values over time to the SWMR register, we treat each update of $\langle k, v\rangle$, which has stabilized, as an independent *Byzantine (correct or pseudo-correct) write operation*, associated with its timestamp $\langle k, v\rangle.\overline{T} = k$.

**Theorem 6.** *Let reads $R$ and $R'$ return $\langle k, v\rangle$ and $\langle k', v'\rangle$, respectively, and let $R$ precede $R'$. Then $\langle k, v\rangle.\overline{T} \leq \langle k', v'\rangle.\overline{T}$, i.e., $k \leq k'$.*

*Proof.* When $R$ returns $\langle k, v\rangle$, the $WS(k, v)$ has been written to all $R\_final_{p*}$ for some $p$. From the Algorithm 2 pseudo-code, when $R'$ is invoked, it will be guaranteed to read $WS(k, v)$ into $Z$ and hence the $(k', v')$ returned to $R'$ will be such that $k' \geq k$. □

**Theorem 7.** *The SWMR register implemented by Algorithm 1 satisfies Byzantine linearizability of executions.*

*Proof.* Let $L_R$ be a linearization of the correct readers' read operations such that (i) the order of all non-overlapping reads is preserved, and (ii) if read `R` returns $\langle k, v\rangle$, read `R'` returns $\langle k', v'\rangle$, and $\langle k, v\rangle.\overline{T} \mapsto \langle k', v'\rangle.\overline{T}$ then `R` precedes `R'` in $L_R$. Such an $L_R$ is guaranteed to exist because Theorem 6 ensures that there is no ordering conflict between (i) and (ii).

For the pseudo-correct writes to form, i.e., be totally ordered along with correct writes, $n > 3f$ by Theorem 5. For any $\langle k, v\rangle$ returned by a read, the value must have been written by a correct or a pseudo-correct write operation. In the linearization $L_R$, place a (Byzantine) pseudo-correct or correct write operation writing a value $\langle k, v\rangle$ and assigned timestamp $\langle k, v\rangle.\overline{T}$ immediately before the first read operation R in $L_R$ that reads $\langle k, v\rangle$.

The resulting linearization is seen to be a Byzantine linearization that considers all the read operations of correct readers, and includes some correct and pseudo-correct write operations. □

**Theorem 8.** *Algorithm 1 implements a Byzantine linearizable SWMR register using SWSR registers, provided $n > 3f$.*

*Proof.* A Byzantine linearization of the concurrent history was identified in the proof of Theorem 7, and it required $n > 3f$. We now show that the value returned by a read operation satisfies "reading a current value" and "no new-old inversions" with respect to this Byzantine linearization.

- **Reading a current value:** As per the construction of the Byzantine linearization of the concurrent history, a read R of $\langle k, v\rangle$ is preceded by the correct or pseudo-correct write operation that immediately precedes R.
- **No "new-old" inversions:** Let read R by $x$ return $\langle k, u\rangle$ and let read R' by $y$ return $\langle k', u'\rangle$, where R precedes R'. R will write $\langle k, u\rangle$ in $R\_final_{x*}$ before returning. R' will read from $R\_final_{*y}$, add these elements to $Z$ and invoke $Find\_Latest(Z)$ which will return the most recent value $\langle k, u\rangle^{max}$ as per $\mapsto$. It is guaranteed that if $\langle k, u\rangle^{max} \neq \langle k, u\rangle$ then $\langle k, u\rangle.\overline{T} \mapsto \langle k, u\rangle^{max}.\overline{T}$ because all the values in $Z$ are totally ordered by $\mapsto$ (Theorem 5, Corollary 2) and from Definition 16, $\langle k, u\rangle.\overline{T} \mapsto \langle k, u\rangle^{max}.\overline{T}$ implies that the write of $\langle k, u\rangle^{max}$ had a greater witness timestamp than the write of $\langle k, u\rangle$ for the at least $n-2f$ common processes that witnessed both writes. The value $\langle k', u'\rangle$ returned by R' is $\langle k, u\rangle^{max}$. As $\langle k, u\rangle.\overline{T} \mapsto \langle k, u\rangle^{max}.\overline{T}$, $i \leq j$ (as defined in Definition 7) as per the Byzantine linearization of the concurrent history. Thus there are no inversions. □

**Space Complexity.** The algorithm uses $2n^2$ shared SWSR registers: $n^2$ $R\_witness$ registers of size $O(1)$, $n^2$ $R\_final$ registers of size $O(n)$. It also uses $2n$ shared SWSR registers: $n$ $R\_init$ registers of size $O(1)$, and $n$ $R\_ack$ registers of size $O(1)$. The local space at each reader process can be seen to be $O(n)$.

## 6 Discussion

We gave an algorithm to construct a Byzantine tolerant SWMR atomic register from SWSR atomic registers, that meets the definition of Byzantine register linearizability from [8], and overcomes the drawbacks of the Hu-Toueg algorithm.

One limitation of the proposed algorithm is that if the Byzantine writer uses a high (highest) value of $k$, no further writes will thenceforth be returnable to the readers, affecting liveness. A solution that overcomes this drawback using vector time [9] is given in [7].

## References

1. Aguilera, M.K., Ben-David, N., Guerraoui, R., Marathe, V.J., Zablotchi, I.: The impact of RDMA on agreement. In: Robinson, P., Ellen, F. (eds.) Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 409–418. ACM (2019), https://doi.org/10.1145/3293611.3331601
2. Cohen, S., Keidar, I.: Tame the wild with byzantine linearizability: Reliable broadcast, snapshots, and asset transfer. In: Gilbert, S. (ed.) 35th International Symposium on Distributed Computing, DISC 2021, October 4-8, 2021, Freiburg, Germany (Virtual Conference). LIPIcs, vol. 209, pp. 18:1–18:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021), https://doi.org/10.4230/LIPIcs.DISC.2021.18
3. Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming. Morgan-Kaufmann (2008)
4. Hu, X., Toueg, S.: On implementing SWMR registers from SWSR registers in systems with byzantine failures. In: Scheideler, C. (ed.) 36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA. LIPIcs, vol. 246, pp. 36:1–36:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2022), https://doi.org/10.4230/LIPIcs.DISC.2022.36
5. Hu, X., Toueg, S.: On implementing SWMR registers from SWSR registers in systems with byzantine failures. Distributed Comput. **37**(2), 145–175 (2024), https://doi.org/10.1007/s00446-024-00465-5
6. Imbs, D., Rajsbaum, S., Raynal, M., Stainer, J.: Read/write shared memory abstraction on top of asynchronous byzantine message-passing systems. J. Parallel Distributed Comput. **93-94**, 1–9 (2016), https://doi.org/10.1016/j.jpdc.2016.03.012
7. Kshemkalyani, A.D., Piduguralla, M., Peri, S., Misra, A.: Construction of a byzantine linearizable SWMR atomic register from SWSR atomic registers. CoRR **abs/2405.19457** (2024), https://doi.org/10.48550/arXiv.2405.19457
8. Kshemkalyani, A.D., Piduguralla, M., Peri, S., Misra, A.: On constructing a byzantine linearizable SWMR atomic register from SWSR atomic registers. In: Korman, A., Chakraborty, S., Peri, S., Boldrini, C., Robinson, P. (eds.) Proceedings of the 26th International Conference on Distributed Computing and Networking, ICDCN 2025, Hyderabad, India, January 4-7, 2025. pp. 239–243. ACM (2025), https://doi.org/10.1145/3700838.3700839
9. Kshemkalyani, A.D., Singhal, M.: Distributed Computing: Principles, Algorithms, and Systems. Cambridge University Press (2011), https://doi.org/10.1017/CBO9780511805318
10. Mostéfaoui, A., Petrolia, M., Raynal, M., Jard, C.: Atomic read/write memory in signature-free byzantine asynchronous message-passing systems. Theory Comput. Syst. **60**(4), 677–694 (2017), https://doi.org/10.1007/s00224-016-9699-8