Mobile Byzantine Agreement in a Trusted World

Bo Pan
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Maria Potop-Butucaru
Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

Abstract

In this paper, we address the Byzantine Agreement problem in synchronous systems where Byzantine agents can move from process to process, corrupting their host. We focus on two representative models: *Garay's* and *Buhrman's* models. In *Garay's model*, when a process has been left by the Byzantine agent, it enters a *cured* state, is aware of its condition, and can remain silent for a round to prevent the dissemination of incorrect information. In *Buhrman's model*, a Byzantine agent moves together with the message. It has been shown that solving Byzantine Agreement requires at least $4t + 1$ processes in *Garay's model*, and at least $3t + 1$ in *Buhrman's model*. In this paper, we aim to increase the tolerance to mobile Byzantine agents by integrating a trusted counter abstraction into both models. This abstraction prevents nodes from equivocating. In the new models, we prove that at least $3t + 1$, respectively $2t + 1$ processors are needed to tolerate t mobile Byzantine agents. Furthermore, we propose novel Mobile Byzantine Agreement algorithms that match these new lower bounds for both *Garay's* and *Buhrman's* models, achieving agreement in $\mathcal{O}(n)$ synchronous rounds.

2012 ACM Subject Classification Theory of computation → Distributed algorithms

Keywords and phrases Byzantine Agreement, Mobile Faults, Trusted Abstractions

Digital Object Identifier 10.4230/LIPIcs.OPODIS.2025.7

1 Introduction

Byzantine Agreement, introduced by Lamport et al. [23, 28], is a fundamental building block in distributed systems. Since its introduction, Byzantine Agreement has been studied from different angles in various models (from synchronous [23, 28, 29] to asynchronous [16, 25], from authenticated [19] to anonymous [26]). Byzantine Agreement has been addressed in both deterministic [23, 28] and probabilistic [9, 20] settings. The *resiliency* (i.e. the fraction of faulty nodes the protocol can tolerate) has been the core research around Byzantine Agreement for decades. Notably, Lamport et al. [23] fix this bound to one third in environments with no authentication and further Borderding [8] proves that this bound holds even for models with local authentication. In all the previous mentioned works it is assumed that the faulty behavior is not mobile. That is, once a node is affected by a Byzantine behavior it will stay affected all along the system execution.

The newly emergent area of blockchains and more generally distributed ledger technologies brought to the light new types of faults and attacks that cannot be predicted accurately and may affect the system at various moments during its execution (e.g. [12, 22, 1]). Some of these behaviors can be modeled as malicious "agent" that can move through the network and corrupt the nodes they occupy. In the literature this model is known as *mobile Byzantine model*.

Related works

This paper investigates the Byzantine Agreement problem in the *mobile Byzantine model* enhanced with trusted abstraction. The literature focusing on the *mobile Byzantine model* investigates two main types of mobility: constrained and unconstrained. Like the classical

distributed systems, the main case study problem in mobile Byzantine faults model has been Byzantine Agreement. When faults are mobile, the problem is known as Mobile Byzantine Agreement and requires special attention to *preserving* agreement even after it has been reached.

In *constrained mobility* settings, Buhrman et al. [10] consider that malicious agents move from one node to another only when protocol messages are sent (similar to how viruses would propagate). In that model, they prove a tight bound for Mobile Byzantine Agreement ($n > 3t$, where t is the maximal number of simultaneously faulty processes) and propose a time-optimal protocol that matches this bound.

In *unconstrained mobility*, the mobility of malicious agents is *not* constrained by message exchanges [30, 27, 21, 4, 31]. In Garay [21], Banu et al. [4], and Sasaki et al. [31], the authors consider that processes execute synchronous rounds composed of three phases: *send*, *receive*, *compute*. Between two consecutive rounds, malicious agents can move from one host to another, hence the set of faulty processes has a bounded size, although its membership can change from one round to the next.

In Garay's model, a process can detect its infection after the fact. More precisely, during the first round following the leave of the malicious agent, a process enters a *cured* state, during which it can take preventive actions to avoid sending messages based on a corrupted state. Under this assumption, Garay [21] proposes an algorithm that solves Mobile Byzantine Agreement provided that $n > 6t$. In the same model, Banu et al. [4] propose a Mobile Byzantine Agreement algorithm for $n > 4t$. Sasaki et al. [31] investigate the problem in a different model where processes do not have this ability to detect when malicious agents move. This work extends the tight bounds ($n > 3t$ and $d > 2t$) for Byzantine Agreement of Dolev [18] in arbitrary networks with static faults. In [6] the authors prove that the Mobile Byzantine Agreement problem has no solution if the size of the network is $n < 5t$ (where t is an upper bound on the number of faulty agents) and propose an algorithm that matches this bound in a variant of Sasaki's model [31].

In order to further increase the resilience of systems to Byzantine behaviors, one recent direction is the use of a trusted environment that provides a simple and limited set of trusted services. In this line of research, several works investigated the use of trusted hardware or software. Correia et al. introduce TTCB wormhole in [14, 15] proved that BFT can support a fraction of half Byzantine nodes. Chun et al. proposed Attested Append-Only Memory (A2M) [13], a trusted system that removes from the faulty nodes the ability to equivocate (i.e. a faulty node may send different messages to different nodes). They introduce PBFT-EA, a modified PBFT [11] that uses A2M for each message exchanged; the message is appended to a log and the attestation produced is sent with the message. The use of this abstraction increases the resilience from one third to one half. Levin et al. propose TrInc [24], a trusted monotonic counter that deals with equivocation in large distributed systems by providing a primitive: once-in-a-lifetime attestations. They also prove that TrInc can implement A2M. Monotonic trusted counter is used by Veronese et al. in [32]. They propose USIG (Unique Sequential Identifier Generator) a local service available in each node that signs a message and assigns it the value of a counter.

Furthermore, in the context of blockchains in [33], the authors enhance HotStuff with small trusted environments to tolerate a minority of corruptions. *Damysus* (in [17]) proposes a BFT protocol that uses trusted environments to improve Hotstuff resilience. In [5], the authors proposed enhance Tendermint protocol with a specific authenticated broadcast service (called A2M-broadcast) to tolerate a minority of faulty processes. The above works do not address the tolerance to Mobile Byzantine faults and the trusted abstractions contain and make use of trusted memory (logs).

In this paper we follow the line of research recently opened by [2] where the authors introduce a simple trusted abstraction, the *Trusted Monotonic Counter*, defined in terms of liveness and safety. Integrating this simple trusted counter into Bracha's algorithms increases their resilience to $t+1$ in the case of reliable broadcast and $2t+1$ in the case of probabilistic consensus [3]. Differently from [2] and [3], we address Mobile Byzantine settings.

Our contribution

In this paper, we address the Mobile Byzantine Agreement problem in Garay's model [21] and Buhrman's model [10], both enhanced with the Trusted Monotonic Counter (TMC) abstraction [2]. For each of these models, we establish tight lower bounds on the number of processes required to solve Byzantine Agreement: in Section 3, we show that no solution exists for Garay's model if $n \leq 3t$, and for Buhrman's model if $n \leq 2t$, where t is an upper bound on the number of faulty agents. Then we propose algorithms for both models that match these bounds in Section 5 and Section 6. Each algorithm guarantees agreement within $\mathcal{O}(n)$ synchronous rounds. It should be noted that compared to the previous algorithm proposed in [6], our algorithm shares a similar three-phase structure (proposing, collecting, deciding), but crucially differs in the way messages are authenticated and validated.

In our work, by integrating a Trusted Monotonic Counter (TMC), our design prevents equivocation: a faulty process cannot send different messages without being detected. This stronger consistency guarantee allows us to lower the validating thresholds, thus improving the fault tolerance for Garay's model from $n \geq 4t+1$ [4] to $n \geq 3t+1$ processes, and for Buhrman's model from $n \geq 3t+1$ [10] to $n \geq 2t+1$ processes.

2 Model and Definition

2.1 System model

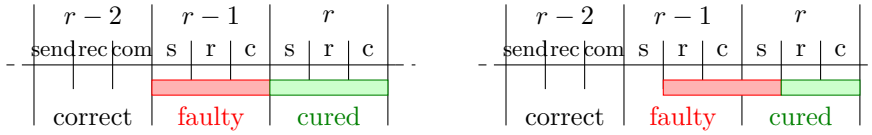
Processes

We consider a synchronous message-passing system consisting of n processes $p_0, p_1, \ldots, p_{n-1}$ where $\Pi = \{0, \ldots, n-1\}$ denotes the set of process indices. Each process is an automaton whose state evolves following the execution of its local algorithm. All processes execute the same algorithm.

The network is fully connected: all pairs of processes are directly linked with a reliable bidirectional channel; i.e. there is no loss, duplication, or alteration of messages. The system evolves in synchronous rounds and all processes start simultaneously at round 0. There is a round counter accessible to the algorithm executed by each process. Each round consists of three steps; *send*, *receive*, and *compute*. Based on its current local state, a process (1) computes and *sends* a message to all processes (including itself); (2) *receives* messages sent by all processes (including itself); and (3) *computes* its new state based on its current state and the set of received messages.

Mobile malicious agents

Faults are represented by malicious mobile agents that can move from process to process between rounds. There are at most t malicious agents, with $t < n$, and any process can be occupied by an agent. A process is said to be *faulty* in a given round if it is occupied by an agent in that round. A process which is not occupied by a malicious agent, but was occupied in the previous round is called a *cured* process.



(a) Garay's model [21, 4]. **(b)** Buhrman et al. model [10].

Figure 1 Graphical representation of the fault models.

A process which is neither faulty nor cured is called a *correct* process. $\mathcal{F}_r$, $\mathcal{C}o_r$, and $\mathcal{C}u_r$ denote respectively the set of faulty, correct, and cured processes at round r. For ease of writing, we also consider the combined sets of correct/cured processes as the set of non-faulty processes $\mathcal{C}c_r = \mathcal{C}o_r \cup \mathcal{C}u_r = \Pi \setminus \mathcal{F}_r$.

Malicious agents are mobile. The behavior of a faulty process is controlled by the malicious agent. In particular, the agent can corrupt the local state of its host process, and force it to send arbitrary messages (potentially different messages to different processes). However, a malicious agent cannot corrupt the identity of that process (i.e., it cannot send messages using another identity), and is unable to modify the code of the algorithm (i.e., the process resumes executing the correct algorithm after the malicious agent moves away). So we assume a secure, tamper-proof read-only memory where the identity and the code are stored. While it is possible for each non-faulty process to rejuvenate its code at the beginning of each round, local variables may still be corrupted (and of course cannot be recovered on its own).

In this paper we consider two variants of mobility illustrated in Figure 1:

Garay's model [21]: Malicious agents can move between the compute step of a round and the send step of the next round. When a process is in the *cured* state, it is aware of its condition and thus can remain silent for a round to prevent the dissemination of wrong information.

Buhrman's model [10]: Malicious agents move together with the message, i.e., they move between the send step and the receive round of the same round. When a process is in the *cured* state it is aware of that.

Initial state corruptions. At most t processes may begin their execution in a *corrupted initial state*. In order to keep a consistent notation and the presentation elegant, we simply represent this case by considering an imaginary round -1 in which that process is occupied. Thus, a process starting its execution in a *corrupted initial state* is either faulty or, by an abuse of terminology, cured in round 0. Therefore, just like in any other subsequent round, it is possible for round 0 to have at most t cured processes in addition to t faulty processes.

Notation. In the formal definitions and proofs, var_i^r denotes the value of variable var in process p_i at the end of round r. We also use the notation $\#w(\mathcal{W})$ to refer to the number of occurrences of w in tuple $\mathcal{W}$.

In this work, we assume that the round number counter is stored in a tamper-resistant register and cannot be modified by Byzantine agents. This counter ensures that messages can be uniquely tagged with the corresponding round identifier. Note that while we adopt this assumption for simplicity, prior work [6] has shown that this constraint can be relaxed. Specifically, they propose techniques to emulate round numbers securely without requiring a trusted round counter. Our protocol can be extended using such techniques to remove this assumption without compromising correctness.

2.2 Mobile Byzantine Agreement problem

Each initially-correct process p_i has an initial value w_i. All processes must *decide*[1] a value *dec* such that the following properties hold:

1. *BA-Termination*: Eventually, all non-faulty processes during a round terminate the round with a non-bottom decided value.

$$\exists r, \quad \forall r' > r \quad \forall i \in \mathcal{C}c_{r'} \quad dec_i^{r'} \neq \perp$$

2. *BA-Agreement*: No two non-faulty processes decide different values:

$$\forall r, r' \quad \forall i \in \mathcal{C}c_r \quad \forall j \in \mathcal{C}c_{r'} \quad \left(dec_i^r \neq \perp \ \wedge \ dec_j^{r'} \neq \perp\right) \Rightarrow \left(dec_i^r = dec_j^{r'}\right)$$

3. *BA-Validity*: If all initially-correct processes propose the same value w, correct processes can decide only w.

$$\forall w \quad (\forall i \in \mathcal{C}o_0 \quad w_i = w) \Rightarrow (\forall r \quad \forall i \in \mathcal{C}c_r \quad dec_i^r \in \{\perp, w\})$$

After some round, all non-faulty processes must decide the same non-bottom value at every round. Temporary undecided states (denoted by $\perp$) are permitted during the convergence phase, but not after agreement is reached.

We assume that at least one process remains correct for $\Omega(n)$ rounds of communication. This assumption originates from the impossibility result of [21], and is acknowledged in [6], as one way of overcoming the impossibility of solving the Mobile Byzantine Agreement problem. The assumption is expressed as $\Omega(n)$ for generality, our upper bounds rely only on the fact that one process remains correct for $3n$ rounds.

2.3 Trusted Monotonic Counter Object

We enhance the system with the Trusted Monotonic Counter Oracle abstraction TMC-Object defined below. We adopt the specification of the Trusted Monotonic Counter presented in [3].

This abstraction allows us, in Garay's model, to design a Mobile Byzantine Agreement algorithm that supports t Byzantine failures among n processes, where $n \geq 3t + 1$, which improves the $n \geq 4t + 1$ bound [4]. In Buhrman's model, we reduce the bound to $n \geq 2t + 1$, improving the $n \geq 3t + 1$ bound [10].

Informally, a TMC provides each process with a tamper-proof counter that attaches a unique, strictly increasing number and an unforgeable certificate to every message. This **prevents equivocation**, ensuring that a process cannot send conflicting messages or reuse old counters. Each process operates its local TMC independently, so parallel broadcasts proceed without interference or the need for centralized trusted infrastructure.

TMC-Object provides each process with a read-only local variable, called trustedCounter. Whenever the TMC-Object is invoked, it returns a value for trustedCounter strictly greater than any previous value it returned. The difference between two successive counter values is exactly 1.

The TMC-Object supports the operation **get_certificate()**. A process p invokes **get_certificate(m)** with a message m. The object returns a *certificate* and the value of the *counter*. The certificate object certifies that the returned value was created by

[1] We use a terminology consistent with the classical definition of Byzantine agreement. However, the action "decide" does not in itself guarantee a permanent decision. Indeed, due to the mobility of the malicious agents, non-faulty processes must *re-decide* the decision at the end of each round.

the tamper-proof TMC-Object object for the message m. The unique identifier is essentially a reading of the monotonic counter trustedCounter, which is incremented whenever **get_certificate(m)** is called.

The TMC-Object object guarantees the following properties:

- **Uniqueness:** The **get_certificate()** operation of TMC-Object will never assign the same identifier to two different messages.
- **Sequentiality:** The **get_certificate()** operation of TMC-Object will always assign an identifier that is the successor of the previous one.

To send a message u certified by TMC-Object, a process p invokes the **get_certificate()** operation of TMC-Object. This invocation creates a certificate $\mathcal{C}_{(p,u)}$ corresponding to the value of the trustedCounter c_p. Then the process sends the tuple $(u, \mathcal{C}_{(p,u)}, c_p)$, which can be verified by any other process receiving the message via the **check_certificate()** operation of the local TMC-Object, which confirms whether the given certificate was produced for the given message. Each invocation of the **get_certificate()** operation of TMC-Object increments the value of the trustedCounter c_p of process p. We call that sequence of operations TMC-Object-Send u.

When receiving a message $(u, \mathcal{C}_{(p,u)}, c_p)$, a process must check if the certificate $\mathcal{C}_{(p,u)}$ for message u corresponds to the value of the counter c_p. If not, the message is considered invalid and is ignored. If they correspond, the message is said to be valid according to the TMC-Object.

Note that Trusted Monotonic Counters are already provided by real-world real implementations such as TrInc [24] and USIG [32], which realize sequential counters with minimal trusted functionality. These systems show that TMCs can be efficiently implemented in real-world distributed settings without requiring expensive trusted computing bases.

3 Lower bound

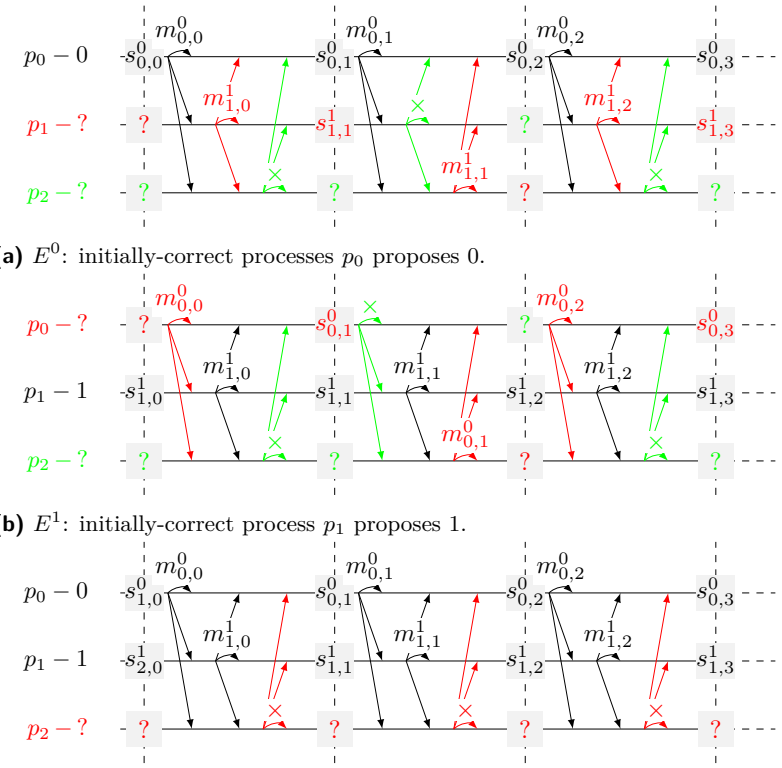
In this section, we prove that in environments where nodes are equipped with equivocation detection mechanisms, in the presence of t mobile Byzantine agents, $3t + 1$ nodes are necessary to solve Mobile Byzantine Agreement problem in Garay's model, and $2t + 1$ nodes are necessary in Buhrman's model.

3.1 Garay's model

▶ **Theorem 1.** *If $n \leq 3t$, there is no deterministic algorithm that solves the Mobile Byzantine Agreement problem in a synchronous n-process system with equivocation prevention, in the presence of (1) t mobile Byzantine agents following Garay's model and (2) t processes with an initially-corrupted state (even with a permanently correct process).*

Proof. Without loss of generality, it suffices to prove the case of 3-process system with $t = 1$. The generalization of this result for $n \leq 3t$, can be done by replacing each process appearing in the proof by a group of processes of size at most t. Since the adversary can control t agents, it can corrupt one group per round, preserving the argument.

Given a system consisting of 3 processes $\{p_0, p_1, p_2\}$, where at least one is permanently correct in one execution. The proof is by contradiction. Let us suppose that there exists a deterministic algorithm that can solve the BA problem in the presence of a single malicious mobile agent. In this algorithm, with equivocation prevention, processes send the same message to all processes.



(a) E^0: initially-correct processes p_0 proposes 0.

(b) E^1: initially-correct process p_1 proposes 1.

(c) E^{01}: initially-correct process p_0 proposes value 0 and p_1 proposes value 1. Process p_2 is faulty.

Figure 2 A contradiction in a 3-process system for Garay's model. Arrows with "×" means "send nothing / stay silent".

General idea. We consider three executions of this algorithm: E^0, E^1 and E^{01}. In executions E^0 and E^1, all correct processes propose the same value; 0 and 1 respectively. The BA properties imply that, eventually, non-faulty processes respectively decide 0 and 1 in these two executions. The third execution, called E^{01}, brings a contradiction: some processes decide 0 while others decide 1.

The three executions are illustrated in Figure 2. Arrows correspond to messages exchanged. Gray boxes contain the new local state computed by each process at the end of each round, which is then used to send messages in the following round. Red (resp. green) indicates actions taken by the faulty (resp. cured) processes. Vertical dashed lines separate successive rounds.

Executions E^0 and E^1. In E^0, the malicious agent alternates between processes p_1 and p_2, and process p_0 is initially correct and proposes 0 in E^0. In E^1, the malicious agent alternates between processes p_0 and p_2, and process p_1 is initially correct and proposes 1 in E^1.

For non-faulty processes, the messages sent during these executions are computed by the algorithm based on the local states of processes. For correct processes (i.e., excluding cured ones), let us denote by $s^0_{i,r}$ (resp. $s^1_{i,r}$) the local state of process p_i at the beginning of the round r in execution E^0 (resp. E^1). Based on this local state, let $m^0_{i,r}$ (resp. $m^1_{i,r}$) denote the message computed and sent by a correct process p_i at round r in execution E^0 (resp. E^1).

We now define the behavior of the malicious agent. In E^0, for the faulty process p_i (either p_1 or p_2) at round r of execution E^0, we choose that p_i sends the message $m^1_{1,r}$ (i.e., the message p_1 would have sent at the same round in E^1) and we choose that when p_1 is faulty,

it updates its local state to $s^1_{1,r+1}$ at the end of the round (i.e., the same state p_1 would have computed in E^1). Similarly we choose that the faulty process p_i (either p_0 or p_2) at round r of execution E^1 sends the message $m^0_{0,r}$, and when p_0 is faulty, it updates its state to $s^0_{0,r+1}$. (The state of p_2 is irrelevant as it is never correct in all three executions.)

Execution E^{01}. In execution E^{01}, the malicious agent always occupies process p_2. The two other processes are initially (and forever) correct. As in E^0, process p_0 propose 0. As in E^1, process p_1 propose 1. In this execution, the faulty process p_2 always stays silent to all processes.

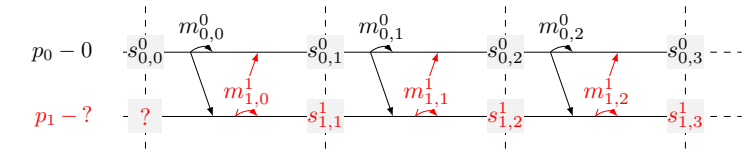
Indistinguishability. An induction on the round number r shows that: *E^0 and E^{01} are indistinguishable for p_0 (which is always correct in these two executions), and E^1 and E^{01} are indistinguishable for p_1 (which is always correct in these two executions).* The complete induction proof is given in Appendix A.

The BA properties imply that, p_0 eventually decides 0 in E^0, it also decides 0 in E^{01}. Since p_1 eventually decides 1 in E^1, it also decides 1 in E^{01}. This violates the BA-Agreement property in E^{01}, therefore contradiction. ◀

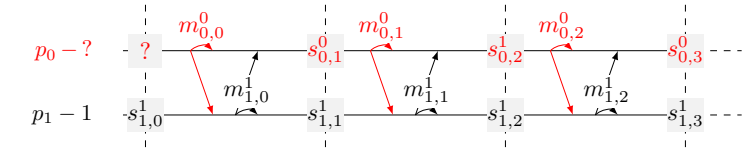
3.2 Buhrman's model

For Buhrman's model, in fact it is sufficient to show that $2t + 1$ nodes are necessary to solve even the static Byzantine Agreement, let alone any mobile model.

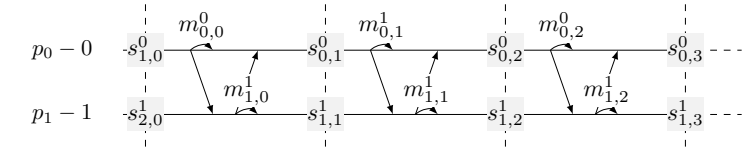
▶ **Lemma 2.** *There is no deterministic algorithm that solves the static Byzantine Agreement problem in a synchronous n-process system with equivocation prevention if $n \leq 2t$.*



(a) E^0: initially-correct processes p_0 proposes 0.



(b) E^1: initially-correct processes p_1 proposes 1.



(c) E^{01}: initially-correct p_0 proposes value 0 and p_1 proposes value 1.

Figure 3 A contradiction in a 2-process system for static malicious agents.

Proof. Similarly to the proof of Theorem 1, consider the three executions illustrated in Figure 3. *E^0 and E^{01} are indistinguishable for p_0 (which is always correct in these two executions), and E^1 and E^{01} are indistinguishable for p_1 (which are always correct in these two executions).* The BA properties imply that, p_0 eventually decide 0 in E^0, it also decides 0 in E^{01}. Since p_1 eventually decides 1 in E^1, it also decides 1 in E^{01}. This violates the BA-Agreement property in E^{01}, therefore contradiction. ◀

▶ **Theorem 3.** *If $n \leq 2t$, there is no deterministic algorithm that solves the Mobile Byzantine Agreement problem in a synchronous n-process system with equivocation prevention, in the presence of (1) t mobile Byzantine agents following Buhrman's model and (2) t processes with an initially-corrupted state (even with a permanently correct process).*

4 Byzantine Reliable Broadcast with Trusted Monotonic Counter

Several approaches to reliable broadcast have been proposed. Some, such as [7], address the problem in a different model by relying on a local failure oracle to detect past faults, leading to a bound of $n > 5t$. Our work instead builds upon the broadcast primitive introduced in [3], which uses a Trusted Monotonic Counter to eliminate equivocation by design, and requires only $n > t$. This leads to a simpler and more efficient construction.

Algorithm 1 Byzantine Reliable Broadcast with a unique TMC-Object(from [3]).

procedure TMC-Object_Broadcast(v)
  Step 0
    if p is the initiator **then**
      TMC-Object-Send $\langle initial, v, p\rangle$ to all
  Step 1
    `Upon reception` of $(\langle initial, u, id\_initiator\rangle, \mathcal{C}, 1)$ message
      Send $(\langle initial, u, id\_initiator\rangle, \mathcal{C}, 1)$ to all
      Deliver u

Algorithm 1 invokes `get_certificate()` of the TMC-Objectwith the counter initialized at 0. When a process p (the *initiator*) broadcasts a value v, it uses TMC-Object-Send to send $\langle initial, v, p\rangle$ together with its certificate and counter to all processes. Upon receiving a valid initial message, each process forwards it and then delivers v. Validity is ensured because only counter value 1 is accepted for the first broadcast, and more generally, each new value must increase the counter by exactly one. This prevents equivocation, as even a Byzantine initiator can commit to only one value per broadcast.

In our synchronous round-based model with mobile Byzantine faults, the same guarantees hold: within each round faults are static, and all broadcasts complete in the send phase. Hence each initiator produces at most one certified message that reaches all processes, and the primitive implements Byzantine Reliable Broadcast for any $t < n$.

▶ **Theorem 4** (adapted from [3])**.** *Let n be the number of processes, and let t be an upper bound on the number of Byzantine agents. In a synchronous system with $n > t$, the TMC-Object-Broadcast primitive implements Byzantine Reliable Broadcast with $O(n^2)$ messages and a single call to `get_certificate()`. This guarantee holds even in the presence of mobile Byzantine faults, provided that broadcasts are invoked during the send step of each round.*

The use of TMC-Object guarantees that even a faulty process cannot equivocate – that is, it cannot send conflicting messages to different recipients without being detected. Each faulty process is restricted to committing to a single value per round. This stronger consistency guarantee allows us to lower the validation thresholds used in our Mobile Byzantine Agreement algorithms, thereby improving fault tolerance while simplifying message validation logic.

5 Mobile BA algorithm for Garay's model with Trusted Monotonic Counter

Building on the primitives introduced above, we now present our solution for Garay's model. Algorithm 2 uses TMC-Object and solves Mobile Byzantine Agreement under the following two conditions: (1) there are at least $3t + 1$ processes in total, and (2) some process remains uncorrupted for the first $3n$ rounds. This improves the fault tolerance bound from $n \geq 4t + 1$ (in [4]) to $n \geq 3t + 1$ processes.

5.1 Description of the MBA algorithm

Algorithm 2 MBA-TMC-Garay (code for p_i, Garay's model).

```
1:  procedure MBA-TMC-Garay(w_i)
2:      v_i ← w_i
3:      for s from 0 to n − 1 do
          Round 3s: Proposing
4:        if Not Cured then TMC-Object-Broadcast(p_i, v_i, 3s)        ▷ Silent if cured in this round
5:        Prop[0..n − 1] ← [⊥, ..., ⊥]
6:        for each unique valid v_j received from p_j do Prop[j] ← v_j
7:        if ∃v : (#v(Prop) ≥ n − 2t) and (#v(Prop) + #⊥(Prop) ≥ n − t) then v_i ← v
8:        else v_i ← ⊥

          Round 3s + 1: Collecting
9:        if Not Cured then TMC-Object-Broadcast(p_i, v_i, 3s + 1)
10:       Rec[0..n − 1] ← [⊥, ..., ⊥]
11:       for each valid v_j received from p_j do Rec[j] ← v_j

          Round 3s + 2: Deciding
12:       if Not Cured then TMC-Object-Broadcast(p_i, Rec, 3s + 2)
13:       Echo[0..n − 1][0..n − 1] ← [[⊥, ..., ⊥], ..., [⊥, ..., ⊥]]
14:       for each valid Rec_j received from p_j do Echo[j] ← Rec_j
15:       Cand[0..n − 1] ← [⊥, ..., ⊥]
16:       for each k from 0 to n − 1 do
17:           if ∃w ≠ ⊥ : #w(Echo[·, k]) > t then Cand[k] ← w
18:       if ∃w ≠ ⊥ : #w(Cand) > t then
19:           v_i ← w
20:       else
21:           c ← s                                                  ▷ Index of current coordinator p_c
22:           if ∃w ≠ ⊥ : #w(Echo[c][·]) > t then v_i ← w
23:           else v_i ← 0

24:     dec_i ← v_i

25:     for r from 3n to ∞ do
          Round r: Maintaining
26:       if Not Cured then TMC-Object-Broadcast(p_i, dec_i, r)
27:       dec_i ← the value received at least n − 2t times
```

The MBA algorithm consists of four types of rounds. During the first n phases, each composed of 3 rounds, it follows a repeating sequence of proposing, collecting, and deciding rounds. At the end of round $3n - 1$, all non-faulty processes are guaranteed to hold the same value v and decide on it (line 24). From round $3n$ onward, it continuously executes the maintaining round.

In accordance with Garay's model, a process that was occupied by a Byzantine agent in the previous round – i.e., a process that is *cured* in the current round – must remain silent during this round.

- **Proposing round:** The goal is to filter out conflicting values and retain only those with strong support. Line 7 ensures that a value is accepted only if it is well supported (at least $n - 2t$ processes) and not contradicted by too many (t) conflicting values. These thresholds guarantee that all non-faulty processes (at least $n - t$) end the round with at most one non-bottom value v. Consequently, all correct processes (at least $n - 2t$) in the next (collecting) round start with at most one non-bottom value v.
- **Collecting round:** The goal of this round is to gather and stabilize the values selected in the previous round. Each process broadcasts its current value and collects values from others into an array *Rec*. This array represents the current opinions of the system and will be used to compute candidate values in the next step. For all non-faulty processes, according to the TMC-Object, only valid values are stored.
- **Deciding round:** In this round processes attempt to reach agreement using the rotating coordinator paradigm. Each process broadcasts its *Rec* array (gathered in the previous round) and collects the *Rec* arrays from others into a matrix *Echo*, where $Echo[j][k]$ stores the value reported by process p_j for position k. Processes then analyze each column of *Echo* to identify consistent values. Each such value is stored in a candidate array *Cand*, where $Cand[k]$ records the majority-supported value (if any) for position k. If enough support for a single value w is observed, it is selected. Otherwise, the coordinator p_c (chosen as process p_s in phase s) helps break the tie. If the coordinator of the current round is correct during the entire phase, all non-faulty processes are guaranteed to terminate the phase with the same value. By assumption, such a coordinating round is guaranteed to exist. This property persists in all subsequent phases even if the corresponding coordinators in those phases are faulty.
- **Maintaining round:** This part repeats indefinitely starting from round $3n$. The goal is to allow cured processes to recover the decided value from correct ones. All processes exchange their current decided values *dec* and update their variable *dec* to the value that has been received at least $n - 2t$ times. In each of these round, all the $n - 2t$ correct processes send the same value (guaranteed by the algorithm), all non-faulty processes receive $n - 2t$ messages containing this same value, thus decide accordingly.

5.2 Proof of correctness

▶ **Lemma 5.** *For any $s \leq n-1$, at the end of the round $3s$, all the $n-t$ non-faulty processes can have at most one non-bottom value. Formally:*

$$\forall s \leq n-1 \quad \forall i,j \in \mathcal{C}c_{3s} \quad \Big((v_i^{3s} \neq \perp) \wedge (v_j^{3s} \neq \perp)\Big) \Rightarrow (v_i^{3s} = v_j^{3s})$$

Proof. We prove the lemma by contradiction. Suppose that there exists two different non-bottom value v_1 and v_2, and two different non-faulty processes p_1 and p_2, such that p_1 (resp. p_2) ends round $3s$ with v_1 (resp. v_2). We show that this is impossible under the threshold conditions of line 7. With the use of TMC-Object, in fact the array *Prop* are the same for all non-faulty processes since there is no equivocation. According to line 7 we have $\#v_1(Prop) + \#\perp(Prop) \geq n - t$ and $\#v_2(Prop) \geq n - 2t$. So $\#v_1(Prop) + \#\perp(Prop) + \#v_2(Prop) \geq (n - t) + (n - 2t) = n + (n - 3t) \geq n + 1$, which is impossible since the size of *Prop* is n. Therefore contradiction. ◀

▶ **Lemma 6.** *There exists a phase such that all non-faulty processes end with the same non-bottom value. Formally:*

$$\exists s \leq n-1 \quad \exists w \neq \perp \quad \forall i \in \mathcal{C}c_{3s+2} \quad v_i^{3s+2} = w$$

Proof. As we assume that at least one process remains non-faulty for at least $3n$ rounds. Let p_c be this process, and we consider the cth phase where p_c is the coordinator (line 22).

In round $3c$, by Lemma 5, all non-faulty processes can have at most one non-bottom value. Formally:

$$\forall i, j \in \mathcal{C}c_{3c} \quad \Big((v_i^{3c} \neq \perp) \wedge (v_j^{3c} \neq \perp)\Big) \Rightarrow (v_i^{3c} = v_j^{3c})$$

If any non-faulty process has a non-bottom value, denote w.

In round $3c+1$, all correct processes in this round were non-faulty in the previous round, so these correct processes send either $\perp$ or w (if exists). Cured ones don't send. Only faulty ones may send other values. Thus all non-faulty processes receive at most t values different from w and $\perp$. Therefore, the arrays Rec of all non-faulty processes contain at most t elements different from w and $\perp$ and these elements are all located at the same indexes. Formally:

$$\forall i, i' \in \mathcal{C}c_{3c+1} \quad \forall k \in \mathcal{C}o_{3c+1} \quad Rec_i^{3c+1}[k] = Rec_{i'}^{3c+1}[k] \in \{w, \perp\}$$

$$\forall i \in \mathcal{C}c_{3c+1} \quad \forall k \in \mathcal{C}u_{3c+1} \quad Rec_i^{3c+1}[k] = \perp$$

In round $3c+2$, all correct processes (non-faulty in round $3c+1$) send an array Rec satisfying the previous conditions, and then all non-faulty processes update $Echo$ such that:

$$\forall i \in \mathcal{C}c_{3c+2} \quad \forall j, j' \in \mathcal{C}o_{3c+2} \quad \forall k \in \mathcal{C}o_{3c+1} \quad Echo_i[j][k] = Echo_i[j'][k] \in \{w, \perp\}$$

$$\forall i \in \mathcal{C}c_{3c+2} \quad \forall j \in \mathcal{C}o_{3c+2} \quad \forall k \in \mathcal{C}u_{3c+1} \quad Echo_i[j][k] = \perp$$

As there is at most t faulty processes, for all such indexes k, the test at line 17 can be true only for the value w:

$$\forall i \in \mathcal{C}c_{3c+2} \quad \forall k \in \mathcal{C}o_{3c+1} \quad Cand_i^{3c+2}[k] \in \{w, \perp\}$$

The test line 18 can be true only for the value w. Starting from the test of line 18, non-faulty processes update their variable v_i by line 19, 22, or 23:

- If no non-faulty process updates its v_i by line 19: This means that all of them update their v_i by testing line 22. As we have assumed that p_c is correct, it sends the same array Rec to all processes in round $3c+1$, then all non-faulty processes have the same line $Echo[c][\cdot]$, which means that all non-faulty processes terminate the phase with the same non-bottom value, either all at line 22, or all at line 23.
- If at least one non-faulty process updates its variable v_i at line 19: Denote p_m such a process. Similarly, it remains to prove that the remaining non-faulty processes which update their v_i from the test of line 22 also update to the same value w.
  As p_m updates at line 19, its $Cand$ contains more than t times the value w. Denote $\mathcal{W}$ the set of indexes k such that $Cand[k] = w$.
  As p_m has executed line 17, for all k in $\mathcal{W}$, the column $Echo[\cdot][k]$ contains more than t times the value w. Since there are at most t faulty processes, at least one correct process is the source of these, which means: for all k in $\mathcal{W}$, there exists a p_j correct in round $3c+2$, such that p_j sends Rec_j with $Rec_j[k] = w$. Formally:

  $$\forall k \in \mathcal{W} \quad \exists j \in \mathcal{C}o_{3c+2} \quad Rec_j^{3c+2}[k] = w$$

  At the point of view of p_j (correct in round $3c+2$ thus non-faulty in round $3c+1$), this value comes from the line 10, which means p_j in the round $3c+1$ received w from p_k.

With the equivocation prevention guaranteed by TMC, p_k broadcasts w in round $3c+1$, no matter whether it is faulty or not. Thus all processes of $\mathcal{W}$ broadcast the value w in the round $3c+1$.
Therefore, all non-faulty processes of round $3c+1$ have received these values w from processes of $\mathcal{W}$. It includes the process p_c which is always correct by assumption. Then p_c sends in round $3c+2$ an array Rec_c that contains at least $|\mathcal{W}| \geq t$ times of value w. Consequently during round $3c+2$, each non-faulty process which has not updated its v_i from line 19 update from line 22 (since the test of line 22 is true). It means that all non-faulty processes update their v_i to w, which concludes the proof. ◀

▶ **Lemma 7.** *If all correct processes start a phase s with the same variable v, then all non-faulty processes terminate the phase with this same value. Formally:*

$$\forall s \leq n-1 \quad \forall w \neq \perp \quad \left(\forall i \in \mathcal{C}o_{3s} \quad v_i^{3s-1} = w\right) \Rightarrow \left(\forall i \in \mathcal{C}c_{3s+2} \quad v_i^{3s+2} = w\right)$$

where v_i^{-1} corresponds to the value initially proposed by process p_i.

Proof. We consider the three rounds of phase s:

Round $3s$. All correct processes (at least $n-2t$) start the round with the same value w. They all send this w at line 4. Cured ones stay silent. Only faulty ones sends other values. Then all non-faulty processes receive w at least $n-2t$ times. and the sum of the number of occurrences of v and $\perp$ in $Prop$ is at least $n-t$. Then all of them update their v_i to w.

Round $3s+1$. All correct processes were non-faulty in the previous round, then all of them have value w at the beginning of this round and send w at line 9. Then all non-faulty processes receive w at least from all correct processes, thus form an array Rec containing at least $n-2t$ times the value w at the indexes corresponding to correct processes.

Round $3s+2$. All correct processes were non-faulty in the round $3s+1$, then all of them send an array at line 12 containing at least $n-2t$ times the value w. Therefore all non-faulty processes obtain a $Echo$ satisfying:

$$\forall i \in \mathcal{C}c_{3s+2} \quad \forall j \in \mathcal{C}o_{3s+2} \quad \forall k \in \mathcal{C}o_{3s+1} \quad Echo_i[j][k] = w$$

As there are more than $2t$ correct processes, all non-faulty processes execute line 17 for all indexes corresponding to correct processes of the round $3s+1$:

$$\forall i \in \mathcal{C}c_{3s+2} \quad \forall k \in \mathcal{C}o_{3s+1} \quad Cand_i[k] = w$$

Then for all non-faulty processes, the array $Cand$ contains at least $|\mathcal{C}o_{3s+1}| \geq n-2t > t$ times of w, thus all non-faulty processes get positive test at line 18, which means all of them update their v_i to w, which concludes the proof. ◀

▶ **Theorem 8.** *Let n be the number of processes, and t be an upper bound of the Byzantine processes following Garay's model. If (1) $n \geq 3t+1$ and (2) there is at least one process remaining correct for the first $3n$ rounds, Algorithm 2 solves the Mobile Byzantine Agreement problem in a synchronous n-process system.*

Proof. Lemma 6 and 7 guarantee that all non-faulty processes of round $3n-1$ terminate the round with the same non-bottom value w in variable v. At line 24, all these non-faulty processes decide the value w. Starting from round $3n$, since there are at least $n-2t$ correct processes during each round, all non-faulty processes will always decide this same value w. It proves the BA-Termination and BA-Agreement properties. To prove the BA-Validity, it is sufficient to apply Lemma 7 from the first round. This concludes the proof. ◀

5.3 Round Complexity

The following complexity bound follows directly from the proof of Lemma 6, which identifies the first phase where the coordinator remains correct throughout.

▶ **Theorem 9.** *In Algorithm 2, all non-faulty processes reach agreement by the end of round* $3c + 2$*, where* c *is the index of the first phase with a correct coordinator. In the worst case,* $c = n - 1$*, resulting in a round complexity of* $3n$ *rounds.*

This bound holds under the assumption that at least one process remains correct for $3n$ rounds, as stated in the system model. In our model, up to $n - 1$ processes may be faulty over time. Therefore, this bound of $\mathcal{O}(n)$ also corresponds to $\mathcal{O}(f)$ when compared to the early Byzantine Agreement result of Reischuk [30], where f denotes the number of processes that are ever faulty. In practice, if coordinators are scheduled uniformly or the fault pattern is favorable, the index c can be significantly smaller than n, leading to faster agreement.

6 Mobile BA algorithm for Buhrman's model with Trusted Monotonic Counter

Algorithm 3 MBA-TMC-Buhrman (code for p_i, Buhrman's model).

```
1: procedure MBA-TMC-BUHRMAN(w_i)
2:     v_i ← w_i
3:     for s from 0 to n − 1 do
       Round 3s: Proposing
4:         TMC-Object-Broadcast(p_i, v_i, 3s)
5:         Prop[0..n − 1] ← [⊥, ..., ⊥]
6:         for each unique valid v_j received from p_j do Prop[j] ← v_j
7:         if ∃v : #v(Prop) ≥ n − t then v_i ← v
8:         else v_i ← ⊥

       Round 3s + 1: Collecting
9:         TMC-Object-Broadcast(p_i, v_i, 3s + 1)
10:        Rec[0..n − 1] ← [⊥, ..., ⊥]
11:        for each valid v_j received from p_j do Rec[j] ← v_j

       Round 3s + 2: Deciding
12:        TMC-Object-Broadcast(p_i, Rec, 3s + 2)
13:        Echo[0..n − 1][0..n − 1] ← [[⊥, ..., ⊥], ..., [⊥, ..., ⊥]]
14:        for each valid Rec_j received from p_j do Echo[j] ← Rec_j
15:        Cand[0..n − 1] ← [⊥, ..., ⊥]
16:        for each k from 0 to n − 1 do
17:            if ∃w ≠ ⊥ : #w(Echo[·, k]) > t then Cand[k] ← w
18:        if ∃w ≠ ⊥ : #w(Cand) > t then
19:            v_i ← w
20:        else
21:            c ← s
22:            if ∃w ≠ ⊥ : #w(Echo[c][·]) > t then v_i ← w
23:            else v_i ← 0

24:    dec_i ← v_i

25:    for r from 3n to ∞ do
       Round r: Maintaining
26:        TMC-Object-Broadcast(p_i, dec_i, r)
27:        dec_i ← the value received at least n − t times
```

We now consider Buhrman's model. We introduce Algorithm 3 that uses TMC-Object and solves Mobile Byzantine Agreement under the following two conditions: (1) there are at least $2t+1$ processes in total, and (2) some process remains uncorrupted for the first $3n$ rounds. This improves the fault tolerance bound from $n \geq 3t+1$ (in [10]) to $n \geq 2t+1$ processes.

We recall that, in Buhrman's model, the mobility of malicious agents is constrained: an agent may only move *together with messages* – that is, from the *send* step of one process to the *receive* step of another, within the same round. As a result, the set of faulty processes is determined at the time messages are received. Cured processes are never faulty in the next round's *send* step, and they are aware that they are no longer infected and resume active participation immediately. Therefore, any process that is non-faulty during the *receive* step behaves as a correct process in that round, regardless of whether it was faulty in the previous round. As a consequence, up to $n-t$ processes can be trusted to send the correct decision value in each maintaining round.

The structure and correctness proof follow similar principles to the previous section, but are adapted to Buhrman's mobility model and thresholds made possible by the use of TMC. Our algorithm adjusts thresholds across all rounds to reflect this increased reliability. In particular, a process accepts a value if it is received from at least $n-t$ distinct sources, rather than $n-2t$. These relaxed thresholds are safe due to two key properties: (1) cured processes resume participation immediately and contribute to communication, and (2) the Trusted Monotonic Counter abstraction ensures that faulty processes cannot equivocate.

▶ **Lemma 10.** *For any $s \leq n-1$, at the end of the round $3s$, all the $n-t$ non-faulty processes can have at most one non-bottom value. Formally:*

$$\forall s \leq n-1 \quad \forall i,j \in \mathcal{C}c_{3s} \quad \Big((v_i^{3s} \neq \perp) \wedge (v_j^{3s} \neq \perp)\Big) \Rightarrow (v_i^{3s} = v_j^{3s})$$

Proof. The argument follows the same structure as Lemma 5, but under the modified threshold condition of line 7. Suppose two distinct values $v_1 \neq v_2$ satisfy the condition in line 7, we have $\#v_1(Prop) \geq n-t$ and $\#v_2(Prop) \geq n-t$. So $\#v_1(Prop) + \#v_2(Prop) \geq (n-t)+(n-t) = n+(n-2t) \geq n+1$, which is impossible since the size of $Prop$ is n. Therefore, at most one value can satisfy the condition. ◀

▶ **Lemma 11.** *There exists a phase such that all non-faulty processes end with the same non-bottom value. Formally:*

$$\exists s \leq n-1 \quad \exists w \neq \perp \quad \forall i \in \mathcal{C}c_{3s+2} \quad v_i^{3s+2} = w$$

Proof. The proof is Similar to Lemma 6. The complete proof is in Appendix B. ◀

▶ **Lemma 12.** *If all non-faulty processes at the end of phase $s-1$ terminate with the same variable v, then all non-faulty processes at the end of phase s terminate with this same value. Formally:*

$$\forall s \leq n-1 \quad \forall w \neq \perp \quad \big(\forall i \in \mathcal{C}c_{3s-1} \quad v_i^{3s-1} = w\big) \Rightarrow \big(\forall i \in \mathcal{C}c_{3s+2} \quad v_i^{3s+2} = w\big)$$

where v_i^{-1} corresponds to the value initially proposed by process p_i.

Proof. The complete proof is in Appendix C. ◀

The following results hold in Buhrman's model by analogy with Theorems 8 and 9:

▶ **Theorem 13.** *Let n be the number of processes, and t be an upper bound of the Byzantine processes following Buhrman's model. If (1) $n \geq 2t+1$ and (2) there is at least one process remaining correct for the first $3n$ rounds, Algorithm 3 solves the Mobile Byzantine Agreement problem in a synchronous n-process system.*

▶ **Theorem 14.** *In Algorithm 3, all non-faulty processes reach agreement by the end of round* $3c + 2$*, where* c *is the index of the first phase with a correct coordinator. In the worst case,* $c = n - 1$*, yielding a round complexity of* $3n$ *rounds.*

7 Conclusion

In this paper, we propose novel solutions for Mobile Byzantine Agreement in an environment where processes are equipped with a Trusted Monotonic Counter abstraction that provides a non-falsifiable, verifiable, unique, monotonic, and sequential counter. We investigate the problem under two different mobile Byzantine models: Garay's and Buhrman's. In the case of Garay's model, our deterministic algorithm tolerates t mobile Byzantine processes with $n \geq 3t + 1$ while in the original model the best known bound was $n \geq 4t + 1$. Our second algorithm for Buhrman's model tolerates t Byzantine processes with $n \geq 2t + 1$ while in the original model the bound was $n \geq 3t + 1$. We also prove that these two bounds are tight. Both algorithms share a common three-phase structure and achieve agreement within $\mathcal{O}(n)$ synchronous rounds. An interesting direction for future work is to design a generic transformer from mobile Byzantine model to mobile Byzantine with trusted components model as suggested in [5] for the static case.

References

1 Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. Dfinity consensus, explored. *IACR Cryptol. ePrint Arch.*, page 1153, 2018. URL: https://eprint.iacr.org/2018/1153.

2 Yackolley Amoussou-Guenou, Lionel Beltrando, Maurice Herlihy, and Maria Potop-Butucaru. Byzantine reliable broadcast with one trusted monotonic counter. In Toshimitsu Masuzawa, Yoshiaki Katayama, Hirotsugu Kakugawa, Junya Nakamura, and Yonghwan Kim, editors, *Stabilization, Safety, and Security of Distributed Systems - 26th International Symposium, SSS 2024, Nagoya, Japan, October 20-22, 2024, Proceedings*, volume 14931 of *Lecture Notes in Computer Science*, pages 360–374. Springer, 2024. doi:10.1007/978-3-031-74498-3_26.

3 Yackolley Amoussou-Guenou, Maurice Herlihy, and Maria Potop-Butucaru. Asynchronous byzantine consensus with trusted monotonic counters. In Ulrich Schmid and Roman Kuznets, editors, *Structural Information and Communication Complexity - 32nd International Colloquium, SIROCCO 2025, Delphi, Greece, June 2-4, 2025, Proceedings*, volume 15671 of *Lecture Notes in Computer Science*, pages 23–38. Springer, 2025. doi:10.1007/978-3-031-91736-3_2.

4 Nazreen Banu, Samia Souissi, Taisuke Izumi, and Koichi Wada. An improved byzantine agreement algorithm for synchronous systems with mobile faults. *International Journal of Computer Applications*, 43(22):1–7, April 2012.

5 Lionel Beltrando, Maria Potop-Butucaru, and José Alfaro. Tendertee: Increasing the resilience of tendermint by using trusted environments. In *24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, January 4-7, 2023*, pages 90–99. ACM, 2023. doi:10.1145/3571306.3571394.

6 François Bonnet, Xavier Défago, Thanh Dang Nguyen, and Maria Potop-Butucaru. Tight bound on mobile byzantine agreement. *Theor. Comput. Sci.*, 609:361–373, 2016. doi:10.1016/J.TCS.2015.10.019.

7 Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Reliable broadcast despite mobile byzantine faults. In Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi, editors, *27th International Conference on Principles of Distributed Systems, OPODIS 2023, December 6-8, 2023, Tokyo, Japan*, volume 286 of *LIPIcs*, pages 18:1–18:23. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023. doi:10.4230/LIPICS.OPODIS.2023.18.

8 Malte Borderding. Levels of authentication in distributed agreement. In Özalp Babaoglu and Keith Marzullo, editors, *Distributed Algorithms, 10th International Workshop, WDAG '96, Bologna, Italy, October 9-11, 1996, Proceedings*, volume 1151 of *Lecture Notes in Computer Science*, pages 40–55. Springer, 1996. doi:10.1007/3-540-61769-8_4.

9 Gabriel Bracha. An o(log n) expected rounds randomized byzantine generals protocol. *J. ACM*, 34(4):910–920, 1987. doi:10.1145/31846.42229.

10 Harry Buhrman, Juan A. Garay, and Jaap-Henk Hoepman. Optimal resiliency against mobile faults. In *Digest of Papers: FTCS-25, The Twenty-Fifth International Symposium on Fault-Tolerant Computing, Pasadena, California, USA, June 27-30, 1995*, pages 83–88. IEEE Computer Society, 1995. doi:10.1109/FTCS.1995.466995.

11 Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In Margo I. Seltzer and Paul J. Leach, editors, *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999. URL: https://dl.acm.org/citation.cfm?id=296824.

12 Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. doi:10.1016/J.TCS.2019.02.001.

13 Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In Thomas C. Bressoud and M. Frans Kaashoek, editors, *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 189–204. ACM, 2007. doi:10.1145/1294261.1294280.

14 Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*, pages 174–183. IEEE Computer Society, 2004. doi:10.1109/RELDIS.2004.1353018.

15 Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. BFT-TO: intrusion tolerance with less replicas. *Comput. J.*, 56(6):693–715, 2013. doi:10.1093/COMJNL/BXS148.

16 Miguel Correia, Giuliana Santos Veronese, and Lau Cheuk Lung. Asynchronous byzantine consensus with 2f+1 processes. In Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung, editors, *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, pages 475–480. ACM, 2010. doi:10.1145/1774088.1774187.

17 Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. DAMYSUS: streamlined BFT consensus leveraging trusted components. In Yérom-David Bromberg, Anne-Marie Kermarrec, and Christos Kozyrakis, editors, *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 1–16. ACM, 2022. doi:10.1145/3492321.3519568.

18 Danny Dolev. The byzantine generals strike again. *J. Algorithms*, 3(1):14–30, 1982. doi:10.1016/0196-6774(82)90004-9.

19 Danny Dolev, Michael J. Fischer, Robert J. Fowler, Nancy A. Lynch, and H. Raymond Strong. An efficient algorithm for byzantine agreement without authentication. *Inf. Control.*, 52(3):257–274, 1982. doi:10.1016/S0019-9958(82)90776-8.

20 Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997. doi:10.1137/S0097539790187084.

21 Juan A. Garay. Reaching (and maintaining) agreement in the presence of mobile faults (extended abstract). In Gerard Tel and Paul M. B. Vitányi, editors, *Distributed Algorithms, 8th International Workshop, WDAG '94, Terschelling, The Netherlands, September 29 - October 1, 1994, Proceedings*, volume 857 of *Lecture Notes in Computer Science*, pages 253–264. Springer, 1994. doi:10.1007/BFB0020438.

22 Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018. arXiv:1805.04548.

23 Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.

24 Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. In Jennifer Rexford and Emin Gün Sirer, editors, *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*, pages 1–14. USENIX Association, 2009. URL: http://www.usenix.org/events/nsdi09/tech/full_papers/levin/levin.pdf.

25 Jean-Philippe Martin and Lorenzo Alvisi. Fast byzantine consensus. *IEEE Trans. Dependable Secur. Comput.*, 3(3):202–215, 2006. doi:10.1109/TDSC.2006.35.

26 Michael Okun and Amnon Barak. Efficient algorithms for anonymous byzantine agreement. *Theory Comput. Syst.*, 42(2):222–238, 2008. doi:10.1007/S00224-007-9006-9.

27 Rafail Ostrovsky and Moti Yung. How to withstand mobile virus attacks (extended abstract). In Luigi Logrippo, editor, *Proceedings of the Tenth Annual ACM Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 19-21, 1991*, pages 51–59. ACM, 1991. doi:10.1145/112600.112605.

28 Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980. doi:10.1145/322186.322188.

29 Michel Raynal. *Fault-tolerant Agreement in Synchronous Message-passing Systems*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2010. doi:10.2200/S00294ED1V01Y201009DCT003.

30 Rüdiger Reischuk. A new solution for the byzantine generals problem. *Inf. Control.*, 64(1-3):23–42, 1985. doi:10.1016/S0019-9958(85)80042-5.

31 Toru Sasaki, Yukiko Yamauchi, Shuji Kijima, and Masafumi Yamashita. Mobile byzantine agreement on arbitrary network. In Roberto Baldoni, Nicolas Nisse, and Maarten van Steen, editors, *Principles of Distributed Systems - 17th International Conference, OPODIS 2013, Nice, France, December 16-18, 2013. Proceedings*, volume 8304 of *Lecture Notes in Computer Science*, pages 236–250. Springer, 2013. doi:10.1007/978-3-319-03850-6_17.

32 Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *IEEE Trans. Computers*, 62(1):16–30, 2013. doi:10.1109/TC.2011.221.

33 Sravya Yandamuri, Ittai Abraham, Kartik Nayak, and Michael K. Reiter. Communication-efficient BFT protocols using small trusted hardware to tolerate minority corruption. *IACR Cryptol. ePrint Arch.*, page 184, 2021. URL: https://eprint.iacr.org/2021/184.

A Induction proof for indistinguishability argument

We now provide the full inductive proof of the property $\mathcal{P}(r)$, used in the proof of Theorem 1.

We define the predicate $\mathcal{P}(r)$ as:

$$\mathcal{P}(r) = \begin{cases} s^{01}_{0,r} = s^{0}_{0,r} & (p_0 \text{ starts round } r \text{ in } E^0 \text{ and } E^{01} \text{ with the same local state}) \\ s^{01}_{1,r} = s^{1}_{1,r} & (p_1 \text{ starts round } r \text{ in } E^1 \text{ and } E^{01} \text{ with the same local state}) \end{cases}$$

Base case ($r = 0$)

The initial local states in E^0 and E^{01} are the same for p_0, since both propose 0, and p_0 is correct in both. Similarly, p_1 is correct and proposes 1 in both E^1 and E^{01}, so:

$$s^{01}_{0,0} = s^{0}_{0,0}, \quad s^{01}_{1,0} = s^{1}_{1,0}$$

Thus, $\mathcal{P}(0)$ holds.

Inductive step

Assume $\mathcal{P}(r)$ holds. We now show that $\mathcal{P}(r+1)$ holds as well.

- In E^0 and E^{01}, p_0 starts round r in the same state and is correct in both. Hence, it sends the same message $m^0_{0,r}$ to all processes.
- In E^1 and E^{01}, p_1 starts round r in the same state and is also correct in both. Hence, it sends $m^1_{1,r}$ in both.
- For p_2, in E^{01} it is faulty and always silent. In E^0, p_2 is either (1) faulty, in which case the malicious agent forces p_2 to send $m^1_{1,1}$), or (2) cured, in which case it remains silent.
- Therefore, p_0 receives the same set of messages in round r in E^0 and E^{01}. Likewise for p_1 between E^1 and E^{01}.
- Since both p_0 and p_1 are correct, they apply the same algorithm to identical input messages and compute the same next state:

  $$s^{01}_{0,r+1} = s^0_{0,r+1}, \quad s^{01}_{1,r+1} = s^1_{1,r+1}$$

  Hence, $\mathcal{P}(r+1)$ holds. By induction, $\mathcal{P}(r)$ is true for all r.

B Proof for Lemma 11

We consider the cth phase were p_c remains non-faulty for at least $3n$ rounds.

In round $3c$, by Lemma 10, all non-faulty processes can have at most one non-bottom value. If any non-faulty process has a non-bottom value, denote w.

In round $3c+1$, all processes, which were non-faulty in round $3c$, send at line 9 of round $3c+1$ either $\perp$ or w (if exists), then all non-faulty processes of round $3c+1$ receive at most t values different from w and $\perp$. Therefore, the arrays Rec of all non-faulty processes contain at most t elements different from w and $\perp$ and these elements are all located at the same indexes.

$$\forall i, i' \in \mathcal{C}c_{3c+1} \quad \forall k \in \mathcal{C}c_{3c} \quad Rec_i^{3c+1}[k] = Rec_{i'}^{3c+1}[k] \in \{w, \perp\}$$

In round $3c+2$, all processes which were non-faulty in round $3c+1$, send at line 12 of round $3c+2$ an array Rec satisfying the previous conditions, then all non-faulty processes of round $3c+2$ update $Echo$ such that:

$$\forall i \in \mathcal{C}c_{3c+2} \quad \forall j, j' \in \mathcal{C}c_{3c+1} \quad \forall k \in \mathcal{C}c_{3c} \quad Echo_i[j][k] = Echo_i[j'][k] \in \{w, \perp\}$$

The remaining of the proof follows the same as in the round $3c+2$ part of Lemma 6.

C Proof for Lemma 12

Round $3s$. All non-faulty processes (at least $n-t$) of round $3s-1$ start the round $3s$ with the same value w. They all send this w at line 4. Then all non-faulty processes of round $3s$ receive w at least $n-t$ times. Then all of them update their v_i to w.

Round $3s+1$. All non-faulty processes of round $3s$ send w at line 9 in round $3s+1$. Then all non-faulty processes of round $3s+1$ receive w at least $n-t$ times, thus form an array Rec containing at least $n-t$ times the value w at the indexes corresponding to $\mathcal{C}c_{3s}$.

Round $3s+2$. All non-faulty processes of round $3s+1$ send an array at line 12 in round $3s+2$ containing at least $n-t$ times the value w. Therefore all non-faulty processes obtain a $Echo$ satisfying:

$$\forall i \in \mathcal{C}c_{3s+2} \quad \forall j \in \mathcal{C}c_{3s+1} \quad \forall k \in \mathcal{C}c_{3s} \quad Echo_i[j][k] = w$$

As $|\mathcal{C}c_{3s+1}| \geq n-t > t$, all non-faulty processes execute line 17 for all indexes k corresponding to $\mathcal{C}c_{3s}$:

$$\forall i \in \mathcal{C}c_{3s+2} \quad \forall k \in \mathcal{C}c_{3s} \quad Cand_i[k] = w$$

Then for all non-faulty processes, the array $Cand$ contains at least $|\mathcal{C}c_{3s}| \geq n-t > t$ times of w, thus all non-faulty processes get positive test at line 18, which means all of them update their v_i to w, which concludes the proof.