Mobile Byzantine Agreement in a Trusted World

Bo Pan[a], Maria Potop-Butucaru[a]

[a]*LIP6, Sorbonne University, 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 three representative models: *Garay's*, *Bonnet's* and *Buhrman's* models. In *Garay's model* when a process has been left by the Byzantine, it is in the *cured* state and it is aware of its condition and thus can remain silent for a round to prevent the dissemination of wrong information. In *Bonnet's model* a cured process may send messages (based on a state corrupted by the malicious agent), however it will behave correctly in the way it sends those messages: i.e., send messages according to the algorithm. In *Buhrman's model* Byzantine agents move together with the message. It has been shown that in order to solve Byzantine Agreement in the *Garay's model* at least $4t+1$ processors are needed, for *Bonnet's model* at least $5t+1$ processors are needed, while for *Buhrman's model* at least $3t+1$ processors are needed. In this paper we target to increase the tolerance to mobile Byzantines by integrating a trusted counter abstraction to the above models. This abstraction prevents nodes to equivocate. In the new models we prove that at least $3t+1$, respectively $4t+1$, and $2t+1$ processors are needed to tolerate t mobile Byzantine agents. Furthermore, we propose novel Mobile Byzantine Agreement algorithms that match these new lower bounds for *Garay's*, *Bonnet's* and *Buhrman's* models.

Keywords:
Byzantine agreement, Mobile faults, Trusted abstractions

1. Introduction

Byzantine Agreement, introduced by Lamport *et al.* [1, 2], is a fundamental building block in distributed systems. Since its introduction, Byzantine Agreement was studied under different angles in various models (from synchronous [1, 2, 3] to asynchronous [4, 5], from authenticated [6] to anonymous [7]). Byzantine Agreement has been addressed in both deterministic [1, 2] and probabilistic [8, 9] settings. The core research around Byzantine Agreement is its *resiliency* (i.e. the fraction of faulty nodes the protocol can tolerate). Interestingly, in [10] the authors fix this bound to one third in environments with no authentication and further in [11] the authors prove 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 investigation of the agreement problem in dynamic transmission failure models was opened by Santoro *et al.* [12, 13], and later continued by Schmid *et al.* [14] in both complete and arbitrary networks. These models assume that different communication links may randomly fail at different times. Santoro and Widmayer [12] study the k-agreement problem, where the system reaches a k-agreement if, in finite time, k processes choose the same value, either 0 or 1, with $k > \lceil n/2 \rceil$,[1] where n is the total number of processes. Based on the bivalent argument of Fischer *et al.* [15], they state that $(\lceil n/2 + 1 \rceil)$-agreement is impossible in a synchronous system if at each time there is one processor whose messages may be corrupted. This impossibility applies to the mobile Byzantine model. Therefore, results on Mobile Byzantine Agreement rely on the assumption that at least one process remains uncorrupted for $\Omega(n)$ rounds of communication. In the current work we make the same assumption.

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. Some of these behaviours can be model as malicious "agent" that can move through the network and corrupt the nodes they occupy. In the literature this model is know as *mobile byzantine model*.

In the *mobile byzantine model* two main directions have been investigated: *constrained* and *unconstrained mobility* and, as for the classical distributed systems, the main case study problem in mobile byzantine faults model was Byzantine Agreement. When faults are mobile the problem is known as *Mobile Byzantine Agreement* and requires special attention for *preserving* agreement even after it has been reached. Reischuk [16] proposed a first sub-optimal solution for *Mobile Byzantine Agreement* under an additional hypothesis on the stability/stationarity of malicious agents for a given period of time. Later, Ostrovsky and Yung [17] introduced the notion of an adversary that can inject and distribute faults in the system at a constant rate in every round and proposed solutions (mixing randomization and self-stabilization) for tolerating the attacks of mobile viruses.

In *constrained mobility* settings Buhrman *et al.* [18], 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 [16, 17, 19, 20, 21]. In Garay [19], Banu *et al.* [20] and Sasaki *et al.* [21] the authors consider that processes execute synchronous rounds composed of three phases: *send*, *receive* and *compute*. Between two consecutive

[1] If $k \leq \lceil n/2 \rceil$ the k-agreement problem is trivial.

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. Their study focus the synchronous networks mobile agent can move from one process to another one in order to arbitrarily change the algorithm and cause the failures in occupied host. The movement of agents is completed in a round. With this movement, in every round, the set of faulty processes could be changed but the size of the set is not changed. It equals the total number of malicious agents. In Garay's model a process has the ability to detect its own 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 that are based on a corrupted state. Under this assumption, Garay [19] proposes an algorithm that solves Mobile Byzantine Agreement provided that $n > 6t$. In the same model, Banu *et al.* [20] propose a Mobile Byzantine Agreement algorithm for $n > 4t$. Sasaki *et al.* [21] investigate the problem in a different model where processes do not have this ability to detect when malicious agents move. In the case of cured processes, messages are computed in the previous round (i.e., when the process was still faulty) and the cured process cannot control the buffer where these messages are stored, even though the process is no longer faulty. It follows that a cured process may behave as a malicious one for one additional round. They propose tight bounds for Mobile Byzantine Agreement in arbitrary networks if $n > 6t$ and the degree of the network is $d > 4t$. This work extends the tight bounds ($n > 3t$ and $d > 2t$) for Byzantine Agreement of Dolev [22] in arbitrary networks with static faults. In [23] the authors introduced a new model referred in the following Bonnet et al model in which similarly to Sasaki's model [21], a cured process may send corrupted messages, but only computed based on the corrupted state left by a malicious agent. In particular, a malicious agent cannot corrupt the code or the identity of the process it occupies, and a cured process always executes a correct code which ensures, for instance, that it will send the same message to all of its neighbors. In Bonnet model malicious nodes have the same power as in Garay's model, but the cured processes may send messages with corrupted content as in Sasaki's model. In [23] 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 the Sasaki's model [21].

Since the definition of the Byzantine Agreement problem in both static and mobile Byzantine settings, the research targeted to push the limits of the Byzantine resilience. In order to further increase the resilience of systems to Byzantine behaviors, one recent direction is the use of a trusted environments that provide 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.* introduces TTCB wormhole in [24, 25] , a distributed component with local parts (local TTCBs) in nodes and its own bounded secure communication channel (i.e. a channel that cannot be affected by malicious faults where all operations have a bounded delay). By using this wormhole, the authors proved that BFT can

support a fraction of half Byzantine nodes. In other words, they circumvent FLP [26] impossibility by relying on a synchronous and secure distributed subsystem. Although this method allows to increase fault tolerance, the trusted part remains too wide and makes practical implementation too difficult to set up. Chun *et al.* proposed Attested Append-Only Memory (A2M) [27], 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). An A2M equips a node with a set of trusted ordered append-only logs that provide an attestation for each entry. They introduce PBFT-EA, a modified PBFT [28] 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 to half. An alternative to this is the use of a monotonic counter implemented in a tamperproof module. Levin *et al.* propose TrInc [29], 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 [30]. 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. The counter verifies *uniqueness* (never assign the same identifier to two different messages), *monotonicity* (never assign an identifier that is lower than a previous identifier) and *sequentiality* (never assign an identifier that is not the successor of the last assigned identifier). This service has to be implemented in a tamper-proof module.

Furthermore, in the context of blockchains in [31] the authors enhance Hot-Stuff blockchain in order to tolerate a minority of corruptions. *Damysus* ([32]) proposes a BFT protocol that uses trusted environments to improve Hotstuff resilience. In [33] the authors proposed to plug a specific authenticated broadcast service (called A2M-broadcast) into Tendermint protocol in order to tolerate a minority of faulty processes.

In this paper we follow the line of research recently opened by [34] where the authors introduce the *Trusted Monotonic Counter* abstraction, defined in terms of liveness and safety. This simple trusted counter plugged into Bracha's algorithms increase their resilience to $f+1$ in the case of reliable broadcast and $2f+1$ in the case of probabilistic consensus [35]. Differently, from [34] and [35] we address Mobile Byzantine settings.

Our contribution. In this paper we address the Mobile Byzantine Agreement problem in Garay's model [23], in Bonnet's model [23] and in Buhrman's model [18], enhanced with the Trusted Monotonic Counters abstraction [34]. In all of these three models we prove their tight bound for the Byzantine Agreement problem. We prove in Section 3 that the problem has no solution for Garay's model if the size of the network is $n \leq 3t$ (where t is an upper bound on the number of faulty agents), for Bonnet's model if $n \leq 4t$, and for Buhrman's model if $n \leq 2t$. Then we propose algorithms for all three models that match these bounds in Section 5, Section 6 and Section 7.

2. Model and Definition

2.1. System model

Processes. We consider a synchronous message-passing system consisting of n processes p_0, p_1, ..., 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 message.

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 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 three variants of Byzantine mobility:

- *Garay's model*[19]: 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.

- *Bonnet's model*[23]: Processes do not know if they are correct or cured when the malicious agent moved. In the case of cured processes, the computation may be performed using a corrupted state.

- *Buhrman's model*[18]: Malicious agents move together with the message (with the *send* or *broadcast* operation), *i.e.*, they move between the send step and the receive round of the same round.

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 $\mathcal{W}$.

2.2. Mobile Byzantine Agreement problem

Each initially-correct process p_i has an initial value w_i. All processes must *decide*[2] 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:

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

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

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

We assume that there is at least one process remaining uncorrupted for $\Omega(n)$ rounds of communication.

[2]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.

2.3. Trusted Monotonic Counter Object

We enhance the system with The Trusted Monotonic Counter Oracle abstraction TMC-Object defined below. This abstraction allow 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 [20]. In Buhrman's model, we reduce the bound to $n \geq 2t+1$, improving the $n \geq 3t + 1$ bound [18].

We adopt the specification of the Trusted Monotonic Counter presented in [36]. 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 the tamper-proof TMC-Object object for the message m. That is, it the value 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 first invokes the **get_certificate()** operation of TMC-Object, which creates a certificate $\mathcal{C}_{(p,u)}$ corresponding to the value of the trustedCounter c_p,[3] then the process sends the tuple $(u, \mathcal{C}_{(p,u)}, c_p)$, which can be verified by any other process receiving the message. The verification is done using the **check_certificate()** operation of the TMC-Object, which confirms whether the given certificate was produced for the given message and has given the value of the counter. 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.

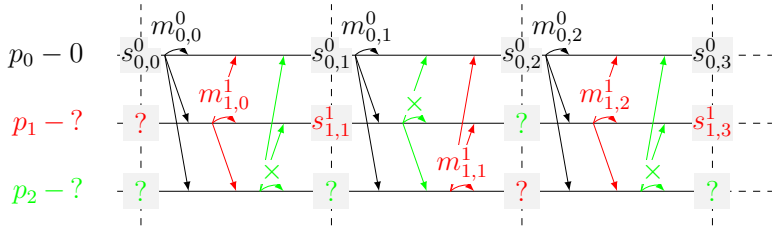
[3] When the value of a counter can be verified against a given certificate, we say that the certificate corresponds to that value of the counter, similarly to digital signatures.

3. Lower Bounds for Mobile Byzantine Agreement with TMC

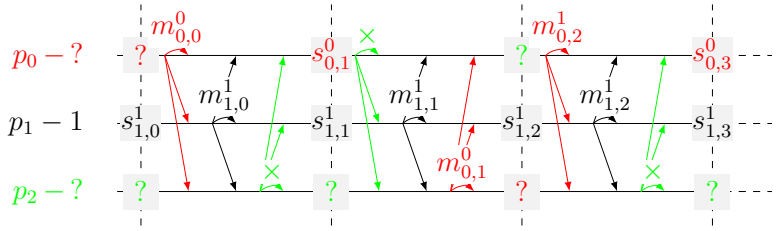
In this section we prove that in environments where nodes are equipped with equivocation detection mechanisms $3t + 1$ nodes are necessary to solve Mobile Byzantine Agreement problem in Garay's model, $4t + 1$ nodes are necessary in Bonnet's model, and $2t + 1$ nodes are necessary in Buhrman's model.

3.1. Garay's model

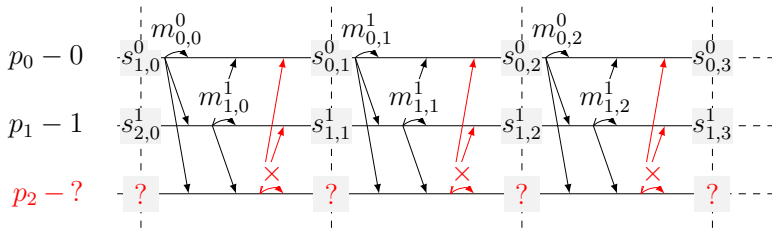
Theorem 1. *There is no deterministic algorithm that solves Mobile Byzantine Agreement in a synchronous 3-process system with equivocation prevention, in the presence of (1) a single mobile Byzantine agent following Garay's model and (2) a single process with an initially-corrupted state (even with a permanently correct process).*



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



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



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

Figure 1: A contradiction in a 3-process system for Garay's model. Legend: arrows with "×" means "sends nothing / stay silent".

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

We consider three executions of this algorithm: E^0, E^1 and E^{01}. The three executions are represented on Figure 1. (Legend: 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 line separate successive rounds.)

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

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 stay silent to all processes.

A simple 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 are always correct in these two executions).* Formally:

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

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. □

When $n \leq 3t$, the proof of Theorem 1 can be generalized by replacing any process appearing in the proof by a group of processes of size at most t.

Corollary 1. *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 if $n \leq 3t$ and (2) t processes with an initially-corrupted state (even with a permanently correct process).*

3.2. Bonnet's model

Theorem 2. *There is no deterministic algorithm that solves Mobile Byzantine Agreement in a synchronous 4-process system with equivocation prevention, in the presence of (1) a single mobile Byzantine agent following Bonnet's model and (2) a single process with an initially-corrupted state (even with a permanently correct process).*

Proof. The proof is similar to Theorem 1. Given a system consisting of 4 processes $\{p_0, \ldots, p_3\}$, let us suppose that there exists an algorithm that can solve the BA problem in the presence of a single malicious mobile agent. Consider three executions of this algorithm: E^0, E^1 and E^{01}, represented on Figure 2. Additional legend: Blue indicates actions taken by the cured processes.

In E^0, the malicious agent alternates between processes p_0 and p_1, and processes p_2 and p_3 are initially correct and propose 0 in E^0. In E^1, the malicious agent alternates between processes p_2 and p_3, and processes p_0 and p_1 are initially correct and propose 1 in E^1.

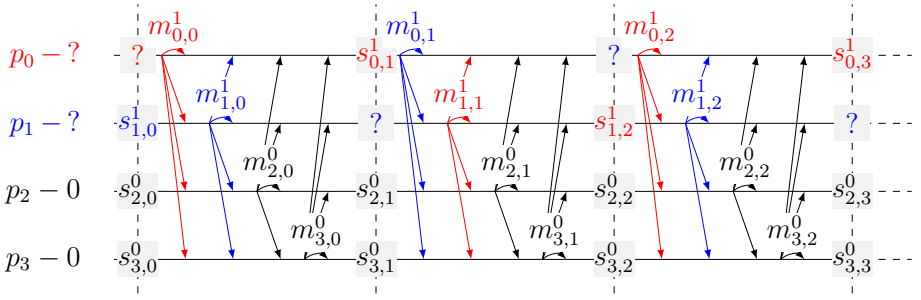
We now define the behavior of the malicious agent. In E^0, for the faulty process p_i (either p_0 or p_1) at round r of execution E^0, we choose that p_i sends the message $m^1_{i,r}$ (*i.e.*, the message it would have sent at the same round in E^1) and we choose that p_i updates its local state to $s^1_{i,r+1}$ at the end of the round (*i.e.*, the same state it would have computed in E^1). Similarly we choose that the faulty process p_i (either p_2 or p_3) at round r of execution E^1 sends the message $m^0_{i,r}$ and updates its state to $s^0_{i,r+1}$.

In execution E^{01}, the malicious agent always occupies process p_3. The three other processes are initially (and forever) correct. As in E^0, processes p_2 propose 0. As in E^1, processes p_0 and p_1 propose 1. In this execution, the faulty process p_3 send always 0 to all processes. At any round, p_3 sends the message $m^0_{3,r}$ to all processes.

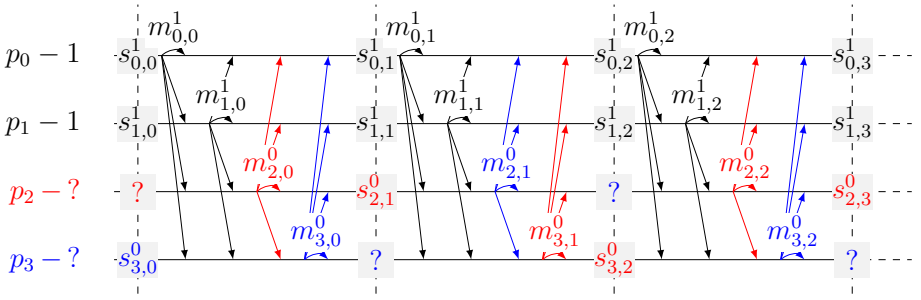
A simple induction on the round number r shows that: *E^0 and E^{01} are indistinguishable for p_2 (which is always correct in these two executions), and similarly E^1 and E^{01} are indistinguishable for p_0 and p_1 (which are always correct in these two executions).* Formally: We have the predicate $\mathcal{P}(r)$ for $r \geq 0$:

$$\mathcal{P}(r) = \begin{cases} p_0 \text{ starts round } r \text{ in } E^1 \text{ and } E^{01} \text{ with the same local state} \\ p_1 \text{ starts round } r \text{ in } E^1 \text{ and } E^{01} \text{ with the same local state} \\ p_2 \text{ starts round } r \text{ in } E^0 \text{ and } E^{01} \text{ with the same local state} \end{cases}$$

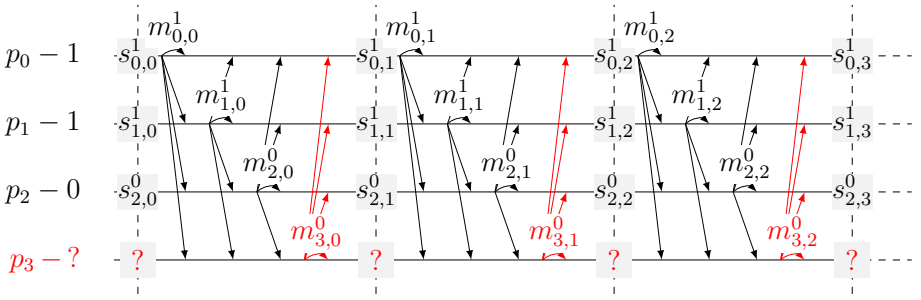
The BA properties imply that, eventually, all non-faulty processes decide 0 in E^0 and decide 1 in E^1. Since p_0 and p_1 eventually decide 1 in E^1, they



(a) E^0: initially-correct processes p_2 and p_3 propose value 0.



(b) E^1: initially-correct processes p_0 and p_1 propose value 1.

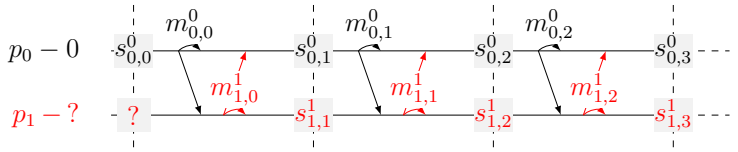


(c) E^{01}: initially-correct processes p_0 and p_1 propose value 1 while initially-correct processes p_2 proposes value 0. Process p_3 is faulty.

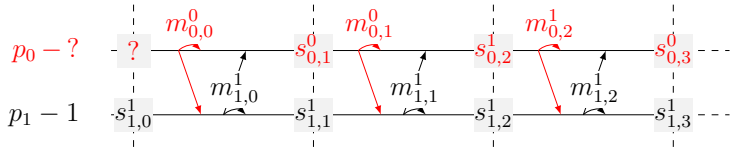
Figure 2: Three executions leading to a contradiction of the existence of a BA protocol in a 4-process system with one mobile malicious agent following Bonnet's model.

also decide 1 in E^{01}. Similarly, since p_2 eventually decides 0 in E^0, it also decides 0 in E^{01}. This violates the BA-Agreement property in E^{01}, therefore contradiction. □

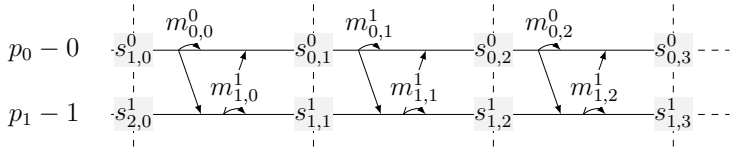
Corollary 2. *There is no deterministic algorithm that solves the Mobile Byzantine Agreement problem in a synchronous n-process system with equivocation prevention, in the presence of (1) t mobile Byzantine agents following Bonnet's model if $n \leq 4t$ and (2) t processes with an initially-corrupted state (even with a permanently correct process).*



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



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



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

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

It is simple to show that $2t + 1$ nodes are necessary to solve even the static Byzantine Agreement, let alone any mobile model.

Consider 3 executions represented by Figure 3, similarly we have: *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.

4. Byzantine Reliable Broadcast with Trusted Monotonic Counter

Our protocols Mobile Byzantine Agreement protocols use the Byzantine Reliable Broadcast [36] protocol resilient to t Byzantine processes when $n > t$.

Algorithm 1 (from [36])Byzantine Reliable Broadcast invoked by process p_i

procedure TMC-OBJECT-BROADCAST(p_i,v_i,*instance*)
  TMC-Object-Send $\langle initial, v_i, p_i \rangle$ to all

  `Upon reception` of $(\langle initial, u, id_initiator \rangle, \mathcal{C}, instance + 1)$ message
    Send $(\langle initial, u, id_initiator \rangle, \mathcal{C}, instance + 1)$ to all
    Deliver u

When a process p_i, at counter value of *instance*, wants to broadcast a value v_i, it TMC-Object-Sends an initial message for v_i ($\langle initial, v \rangle$) to all other processes. The process initiating the broadcast is called the *initiator*. The initiator sends the message as well as the associated certificate and counter, which can be checked by all other processes. When receiving a valid initial message from the proposer, say with value u, a process sends the initiator message (with the certificate and value of the counter) to all the processes. In such a way, it ensures that all other processes will eventually receive the initiator's certified message. First notice that only messages with counter value of $instance + 1$ are considered valid, since a message with a higher counter value means that the initiator already sent another value.

Theorem 3 (proved in [36])**.** *Let n be the number of processes, and t be an upper bound of the Byzantine processes. If $n > t$, Algorithm 1 implements Byzantine Reliable Broadcast in $O(n^2)$ messages and using a unique invocation of the* ***get_certificate()*** *operation of the TMC-Object.*

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

Given a system with t malicious mobile agents following Garay's model, we introduce an algorithm 2 that 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.

5.1. Description of the algorithm

Our algorithm follows the structure of the algorithm in [23], but integrates the use of TMC-Object and adapts threshold modifications accordingly for Garay's model. We assume that the round number counter is stored in a tamper-resistant register.

The algorithm consists of four types of rounds. During the first n phases of 3 rounds each, 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 have the same value v and decide it (line 24). From the $3n + 1$ round onward, it continuously executes the maintaining round.

- **Proposing round:** 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 $n - 2t$) of the next (collecting) round start with at most one non-bottom value v.
- **Collecting round:** processes exchange the values proposed in the previous round, store them in array *Rec*. For all non-faulty processes, according to the TMC-Object, only the valid value is stored.

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:       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 − 2t) and (#v(Prop) + #⊥(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 1 to n 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 − 2t times
```

- **Deciding round:** processes try to agree on the same value v using the rotating coordinator paradigm. 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 are faulty.
- **Maintaining round:** this part is simple and repeats forever from round $3n$. The goal is to allow cured processes to recover the decided value from correct ones.

Equivocation Prevention ensured by the use of TMC. TMC-Object ensures that even a faulty process cannot send conflicting messages to non-faulty processes without being detected. Each faulty process is restricted to committing a single value per round.

5.2. Proof of correctness

Lemma 1. *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. By contradiction, assume 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). With the use of TMC-Object, by Theorem 3, in fact the array $Prop$ are the same for all non-faulty processes. 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 2. *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 1, 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 3. *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:*

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

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

Proof. 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 4. *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 2 and 3 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 3 from the first round. This concludes the proof. □

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

Similarly, for Bonnet's model, we introduce an algorithm 3 that uses TMC-Object and solves Mobile Byzantine Agreement under the following two conditions: (1) there are at least $4t + 1$ processes in total, and (2) some process remains uncorrupted for the first $3n$ rounds.

Algorithm 3 MBA-TMC-MB (code for p_i, Bonnet's model)

```
1: procedure MBA-TMC-MB(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 − 2t 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 unique 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 unique valid Rec_j received from p_j do Echo[j] ← Rec_j
15:      Cand[0..n − 1] ← [⊥, ..., ⊥]
16:      for each k from 1 to n do
17:          if ∃w ≠ ⊥ : #w(Echo[·, k]) > 2t then Cand[k] ← w
18:      if ∃w ≠ ⊥ : #w(Cand) > 2t then
19:          v_i ← w
20:      else
21:          c ← s
22:          if ∃w ≠ ⊥ : #w(Echo[c][·]) > 2t 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 − 2t times
```

Lemma 4. *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:*

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

Proof. By contradiction, assume 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). With the use of TMC-Object, by Theorem 3, in fact the array $Prop$ are the same for all non-faulty processes since there is no equivocation. According to line 7 we have $\#v_1(Prop) \geq n - 2t$ and $\#v_2(Prop) \geq n-2t$. So $\#v_1(Prop)+\#v_2(Prop) \geq (n-2t)\times 2 = n+(n-4t) \geq n+1$, which is impossible since the size of $Prop$ is n. Therefore contradiction. □

Lemma 5. *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 4, all the $n-t$ 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, let w denote this value.

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) (line 7). Only cured and faulty processes may send other values. Thus all non-faulty processes receive at most $2t$ values different from w and $\perp$. Therefore, the arrays Rec of all non-faulty processes contain at most $2t$ elements different from w and $\perp$ and these elements are all located at the same indexes. Formally:

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

In round $3c+2$, all correct processes (non-faulty in round $3c+1$) send an array Rec (line 11) satisfying the previous conditions, and then all non-faulty processes update $Echo$ such that:

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

As there is at most $2t$ faulty or cured 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. There are two cases to consider:

- 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.

- At least one non-faulty process updates its variable v_i at line 19. Denote p_m such a process. As stated above, p_m and all other processes which updates their v_i from line 19 necessarily update them to the value w. 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 $2t$ times the value w. Denote $\mathcal{W}$ the set of indexes k such that $Cand[k] = w$. We have $|\mathcal{W}| \geq 2t$. As p_m has executed line 17, for all k in $\mathcal{W}$, the column $Echo[\cdot][k]$ contains more than $2t$ times the value w. Since there are at most $2t$ non-correct (faulty or cured) 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 $Rec_j[k] = w$ at line 12. 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 11, 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. So p_c sends at line 12 in round $3c+2$ an array Rec_c that contains at least $|\mathcal{W}| \geq 2t$ 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 6. *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:*

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

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

Proof. We consider the three rounds of phase s:

Round $3s$. All $n-2t$ correct processes start the round with the same value w. They all send this w at line 4. Then all non-faulty processes receive w at least $n-2t$ times, then all of them update their v_i to w.

Round $3s+1$. All correct processes in this round 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 correct processes 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 $3s+1$:

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

Therefore for all non-faulty processes, the array $Cand$ contains at least $|\mathcal{C}o_{3s+1}| \geq n-2t > 2t$ 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 5. *Let n be the number of processes, and t be an upper bound of the Byzantine processes. If (1) $n \geq 4t+1$ 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.*

Proof. Lemma 5 and 6 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 6 from the first round. This concludes the proof. □

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

Similarly, for Buhrman's model, we introduce an algorithm 4 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.

The overall proof structure is similar to that of the previous section, but we recall that the key difference between Buhrman's model and Garay's model is that, by definition, Buhrman's model allows a malicious agent to take control only from *receive* step to *send* step of next round. As a result, the state of *send* step in a given round depends on the state of previous round, rather than that of the current round.

Lemma 7. *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:*

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

Algorithm 4 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 1 to n 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
```

Proof. By contradiction, assume 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). With the use of TMC-Object, by Theorem 3, in fact the array $Prop$ are the same for all non-faulty processes since there is no equivocation. According to line 7 we have $\#v_1(Prop) \geq n - t$ 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 contradiction. □

Lemma 8. *There exists a phase such that all non-faulty processes end with the same non-bottom value. Formally:*

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

Proof. Similarly to Lemma 2, we consider the cth phase were p_c remains non-

faulty for at least $3n$ rounds.

In round $3c$, by Lemma 7, 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 2. □

Lemma 9. *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. 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. □

Similarly to Theorem 4 we conclude:

Theorem 6. *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 4 solves the Mobile Byzantine Agreement problem in a synchronous n-process system.*

8. Conclusion and Future work

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 in three different mobile Byzantine models: Garay, Bonnet and Buhrman. In the case of Garay's model our deterministic algorithm tolerates t Byzantine processes with $n \geq 3t+1$ while in the original model the bound was $n \geq 4t + 1$. Our second algorithm for Bonnet's model tolerates t Byzantine processes with $n \geq 4t + 1$ while in the original model the bound was $n \geq 5t + 1$. Our third 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 three bounds are tight.

Our work opens several research directions. An interesting line of research would be to extend the current work to multi-hop networks and to other problems such as the implementation of mobile approximate agreement or registers.

Finally, we plan to design a generic transformer from mobile byzantine model to mobile byzantine with trusted components model as suggested in [33] for the static case.

References

[1] L. Lamport, R. Shostak, M. Pease, The byzantine generals problem, ACM Transactions on Programming Languages and Systems 4 (3) (1982) 382–401.

[2] M. Pease, R. Shostak, L. Lamport, Reaching agreement in the presence of faults, Journal of the ACM 27 (2) (1980) 228–234.

[3] M. Raynal, Fault-tolerant Agreement in Synchronous Message-passing Systems, Synthesis Lectures on Distributed Computing Theory, Morgan & Claypool Publishers, 2010.

[4] M. Correia, G. S. Veronese, L. C. Lung, Asynchronous byzantine consensus with $2f + 1$ processes, in: Proceedings of the 25th ACM Symposium on Applied Computing (SAC'10), 2010, pp. 475–480.

[5] J.-P. Martin, L. Alvisi, Fast byzantine consensus, IEEE Transactions on Dependable and Secure Computing 3 (3) (2006) 202–215.

[6] D. Dolev, M. J. Fischer, T. R. Fowler, N. A. Lynch, H. R. Strong, An efficient algorithm for byzantine agreement without authentication, Information and Control 52 (3) (1982) 257–274.

[7] M. Okun, A. Barak, Efficient algorithms for anonymous byzantine agreement, Theory of Computing Systems 42 (2) (2008) 222–238.

[8] G. Bracha, An o(log n) expected rounds randomized byzantine generals protocol, Journal of the ACM 34 (4) (1987) 910–920.

[9] P. Feldman, S. Micali, An optimal probabilistic protocol for synchronous byzantine agreement, SIAM Journal on Computing 26 (4) (1997) 873–933.

[10] L. Lamport, R. E. Shostak, M. C. Pease, The byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401. doi:10.1145/357172.357176.
URL http://doi.acm.org/10.1145/357172.357176

[11] M. Borcherding, Levels of authentication in distributed agreement, in: Distributed Algorithms,10th International Workshop, WDAG'96 Bologna, Italy, October 9-11, 1996, 1996, pp. 40–55.

[12] N. Santoro, P. Widmayer, Time is not a healer, in: Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science (STACS'89), 1989, pp. 304–313.

[13] N. Santoro, P. Widmayer, Majority and unanimity in synchronous networks with ubiquitous dynamic faults, in: Proceedings of the 12th International Conference on Structural Information and Communication Complexity (SIROCCO'05), 2005, pp. 262–276.

[14] U. Schmid, B. Weiss, I. Keidar, Impossibility results and lower bounds for consensus under link failures, SIAM Journal on Computing 38 (5) (2009) 1912–1951.

[15] M. J. Fischer, N. A. Lynch, M. S. Paterson, Impossibility of distributed consensus with one faulty process, Journal of the ACM 32 (2) (1985) 374–382.

[16] R. Reischuk, A new solution for the byzantine generals problem, Information and Control 64 (1-3) (1985) 23–42.

[17] R. Ostrovsky, M. Yung, How to withstand mobile virus attacks (extended abstract), in: Proceedings of the 10th Annual ACM Symposium on Principles of Distributed Computing (PODC'91), 1991, pp. 51–59.

[18] H. Buhrman, J. A. Garay, J.-H. Hoepman, Optimal resiliency against mobile faults, in: Proceedings of the 25th International Symposium on Fault-Tolerant Computing (FTCS'95), 1995, pp. 83–88.

[19] J. A. Garay, Reaching (and maintaining) agreement in the presence of mobile faults, in: Proceedings of the 8th International Workshop on Distributed Algorithms, Vol. 857, 1994. doi:10.1007/BFb0020438.

[20] N. Banu, S. Souissi, T. Izumi, K. Wada, An improved byzantine agreement algorithm for synchronous systems with mobile faults, International Journal of Computer Applications 43 (22) (2012) 1–7.

[21] T. Sasaki, Y. Yamauchi, S. Kijima, M. Yamashita, Mobile byzantine agreement on arbitrary network, in: Proceedings of the 17th International Conference on Principles of Distributed Systems (OPODIS'13), 2013, pp. 236–250.

[22] D. Dolev, The byzantine generals strike again, Journal of Algorithms 3 (1) (1982) 14–30.

[23] F. Bonnet, X. Défago, T. D. Nguyen, M. Potop-Butucaru, Tight bound on mobile byzantine agreement, Theor. Comput. Sci. 609 (2016) 361–373. doi:10.1016/J.TCS.2015.10.019.
URL https://doi.org/10.1016/j.tcs.2015.10.019

[24] M. Correia, N. F. Neves, P. 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, IEEE Computer Society, 2004, pp. 174–183. doi:10.1109/RELDIS.2004.1353018.
URL https://doi.org/10.1109/RELDIS.2004.1353018

[25] M. Correia, N. F. Neves, P. Veríssimo, BFT-TO: intrusion tolerance with less replicas, Comput. J. 56 (6) (2013) 693–715. doi:10.1093/comjnl/bxs148.
URL https://doi.org/10.1093/comjnl/bxs148

[26] M. J. Fischer, N. A. Lynch, M. Paterson, Impossibility of distributed consensus with one faulty process, in: R. Fagin, P. A. Bernstein (Eds.), Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA, ACM, 1983, pp. 1–7. doi:10.1145/588058.588060.
URL https://doi.org/10.1145/588058.588060

[27] B. Chun, P. Maniatis, S. Shenker, J. Kubiatowicz, Attested append-only memory: making adversaries stick to their word, in: T. C. Bressoud, M. F. Kaashoek (Eds.), Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007, ACM, 2007, pp. 189–204. doi:10.1145/1294261.1294280.
URL https://doi.org/10.1145/1294261.1294280

[28] M. Castro, B. Liskov, Practical byzantine fault tolerance, in: M. I. Seltzer, P. J. Leach (Eds.), Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999, USENIX Association, 1999, pp. 173–186.
URL https://dl.acm.org/citation.cfm?id=296824

[29] D. Levin, J. R. Douceur, J. R. Lorch, T. Moscibroda, Trinc: Small trusted hardware for large distributed systems, in: J. Rexford, E. G. Sirer (Eds.), Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA, USENIX Association, 2009, pp. 1–14.
URL http://www.usenix.org/events/nsdi09/tech/full_papers/levin/levin.pdf

[30] G. S. Veronese, M. Correia, A. N. Bessani, L. C. Lung, P. Veríssimo, Efficient byzantine fault-tolerance, IEEE Trans. Computers 62 (1) (2013) 16–30. doi:10.1109/TC.2011.221.
URL https://doi.org/10.1109/TC.2011.221

[31] S. Yandamuri, I. Abraham, K. Nayak, M. K. Reiter, Communication-efficient BFT protocols using small trusted hardware to tolerate minority corruption, IACR Cryptol. ePrint Arch. (2021) 184.
URL https://eprint.iacr.org/2021/184

[32] J. Decouchant, D. Kozhaya, V. Rahli, J. Yu, DAMYSUS: streamlined BFT consensus leveraging trusted components, in: Y. Bromberg, A. Kermarrec, C. Kozyrakis (Eds.), EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022, ACM, 2022, pp. 1–16. doi:10.1145/3492321.3519568.
URL https://doi.org/10.1145/3492321.3519568

[33] L. Beltrando, M. Potop-Butucaru, J. 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, ACM, 2023, pp. 90–99.

[34] Y. Amoussou-Guenou, L. Beltrando, M. Herlihy, M. Potop-Butucaru, Byzantine reliable broadcast with one trusted monotonic counter, in: T. Masuzawa, Y. Katayama, H. Kakugawa, J. Nakamura, Y. Kim (Eds.), Stabilization, Safety, and Security of Distributed Systems - 26th International Symposium, SSS 2024, Nagoya, Japan, October 20-22, 2024, Proceedings, Vol. 14931 of Lecture Notes in Computer Science, Springer, 2024, pp. 360–374.

[35] Y. Amoussou-Guenou, M. Herlihy, M. Potop-Butucaru, Asynchronous byzantine consensus with trusted monotonic counters, IACR Cryptol. ePrint Arch. (2024) 1937.
URL https://eprint.iacr.org/2024/1937

[36] Y. Amoussou-Guenou, M. Herlihy, M. P. Butucaru, Asynchronous byzantine consensus with trusted monotonic counters, Cryptology ePrint Archive (2024).