TenderTee: Secure Tendermint

Lionel Beltrando
LIP6, Sorbonne University
Finaxys
Paris, France
lbeltrando@finaxys.com

Maria Potop-Butucaru
LIP6,Sorbonne University,
Paris, France
maria.potop-butucaru@lip6.fr

Jose Alfaro
Finaxys, Paris, France
jalfaro@finaxys.com

***Abstract*—Blockchain and distributed ledger technologies have emerged as one of the most revolutionary distributed systems, with the goal of eliminating centralised intermediaries and installing distributed trusted services. They facilitate trustworthy trades and exchanges over the Internet, power cryptocurrencies, ensure transparency for documents, and much more. Committee based-blockchains are considered today as a viable alternative to the original proof-of-work paradigm, since they offer strong consistency and are energy efficient. One of the most popular committee based-blockchain is Tendermint used as core by several popular blockchains such Tezos, Binance Smart Chain or Cosmos. Interestingly, Tendermint as many other committee based-blockchains is designed to tolerate one third of Byzantine nodes. In this paper we propose TenderTee, an enhanced version of Tendermint, able to tolerate one half of Byzantine nodes. The resilience improvement is due to the use of a trusted abstraction, a light version of attested append-only memory, which makes the protocol immune to equivocation (i.e behavior of a faulty node when it sends different faulty messages to different nodes). Furthermore, we prove the correctness of TenderTee for both one-shot and repeated consensus specifications.**

***Index Terms*—Blockchain, Tendermint, Trusted Abstraction**

I. Introduction

A blockchain is a distributed ledger that mimics the functioning of a classical traditional ledger (i.e., transparency and falsification-proof of documentation) in an untrusted environment where the computation is distributed. In blockchain systems, nodes (a.k.a miners) maintain a replica of a continuously-growing list of ordered blocks that include one or more transactions that have been verified by the members of the system. Blocks are linked using cryptography and the order and the content of newly-added blocks is the outcome of a distributed agreement algorithm among the nodes.

First examples of blockchains (Bitcoin [24] and Ethereum [9]) use the proof-of-work paradigm. That is, nodes have to solve a cryptographical puzzle in order to be allowed to produce a new block. The difficulty of this puzzle is high enough such as with high probability only one block is generated at a specific time. Once produced, the new block is diffused in the network and each correct node adds the newly produced block to its local ledger. Proof-of-Work blockchains have two main drawbacks. Firstly, they present a huge electrical consumption. Secondly, they potentially allow the creation of forks which can be a major issue for using blockchain in industrial applications requiring strong consistency.

These problems motivated the emergence of new blockchains (e.g. Solidus [2], Byzcoin [20], PeerCensus [16], Hyperledger [5], RedBelly [15], Tendermint [8],[3],[4], Hot-Stuff [1], Tenderbake [6] etc) using the consensus paradigm, a necessary building block in order to ensure blocks linearizability.

Consensus introduced by Shostak, Pease and Lamport [22], is one of the fundamental problems in the area of fault-tolerant distributed computing. In the consensus problem, n nodes attempt to reach agreement on a value, despite the malicious behavior of up to t of them. One of the measures of the quality of a consensus protocol is its *resiliency*: the fraction of faulty parties the protocol can tolerate. Since the proof of the resilience bound of one third for the Byzantine consensus [22] in environments with no authentication, proved later even for models with local authentication [7] research struggled to increase the consensus resilience.

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, Correia *et al.* introduces TTCB wormhole in [13], [14] 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. Although this method allows to increase fault tolerance, the trusted part remains too wide and makes practical implementation too difficult to set up.

The practicality of the implementations motivated Chun *et al.* to propose Attested Append-Only Memory (A2M) [11], a trusted system that removes to 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. Furthermore, they propose PBFT-EA a modified PBFT [10] that uses A2M for each message exchanged, the message is append to a log and the attestation produced is sent with the message. The use of this abstraction increases the resilience to half. Compared to TTCB that requires a secure and synchronous communication channel, A2M requires no stronger assumptions on network than PBFT.

An alternative to this is the use of a monotonic counter implemented in a tamperproof module. Levin *et al.* propose TrInc ([23]), 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 further used by Veronese *et al.* in [25]. 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 tamperproof module.

Another line of research combines speculative methods and trusted environments (e.g. CheapBFT [19] and ReBFT [17]). In normal execution case (when there are no Byzantine nodes), $f+1$ nodes are enough to guarantee the agreement. In case of detected or suspected Byzantine nodes the protocol switches to a PBFT inspired protocol with trusted hardware and activates f extra passive replicas. Although interesting, the integration of these methods in blockchains environments may generate transient forks.

Interestingly, in the context of blockchains, the use of trusted environments in order to increase the resilience is very recent. The first use of it was proposed in [26]. The authors enhance Hot-Stuff blockchain in order to tolerate a minority of corruptions.

1) Our contribution: Continuing the line of research proposed in [26] we enhance Tendermint [21], [3],[4] with a light version of the trusted abstraction attested append-only memory introduced in [11]. The use of this abstraction makes our protocol, TenderTee, immune to equivocation (i.e behavior of a faulty node when it sends different faulty messages to different nodes). TenderTee enjoys one half Byzantine resilience for both one-shot and repeated consensus. It should be noted that our work is the first to study repeated consensus in trusted environments. Beside the theoretical advancement our work has a strong practical impact since the most important drawback of the industrialization of the current versions of Tendermint suffer from their resilience limitations.

II. System Model

We consider a blockchain system with an infinite set of nodes which can be

- *Obedient (correct)*: Nodes that *always* follow the protocol.
- *Byzantine*: Nodes that can *deviate arbitrarily* from the protocol.

In order to guarantee the linearizability of the blockchain nodes repeatedly execute one-shot consensus instances. We consider that in each consensus instance participate a finite subset of nodes (called committee members) of size $n = 2f+1$ out of which at most f are Byzantine. The way the committees are chosen is currently a hot topic in blockchain area but is out of scope of the current work.

In the following, we assume the presence of a reliable byzantine broadcast. Nodes communicate by exchanging messages through an eventually synchronous network. *Eventually Synchronous* means that after a finite unknown time τ there is an a priori unknown bound δ on the message transfer delay. We do not consider asynchronous communication systems since it is impossible to solve consensus in asynchronous systems when there is at least one failure [18].

We assume that a lightweight Attested Append-Only Memory (A2M) equips each node with a set of trusted ordered append-only logs. Each log has an identifier q and offers methods to append values and access it. There is no method to replace old value, once a value is added to log it can't be replaced. We only present the methods used is the current paper, for a complete specification of A2M, see [11]. In A2M original specification, A2M stores only a suffix of the log, starting from "low" position L to last "high" position H, $H >= L$. For simplicity, in the sequel we omit L and consider infinite logs.

Our lightweight A2M abstraction offers the following methods to write and read values in log q:

- $append(q, x)$ takes a value x, appends it to the log q and returns an attestation att_x. Increments the sequence number by 1, fills last log entry with x and computes the cumulative digest d_H. This method does not erase old value, if the log is unable to allocate storage to new entry, the method fails.
- $lookup(q, n, z)$ takes log identifier q, a sequence number n and a nonce z and returns a $LOOKUP$ attestation.
- $end(q, z)$ is similar to $lookup$ but returns the last entry of the given log. We do not use z parameter (z will be set to n).
- $advance(q, n, d, x)$ is similar to $append$ but allows to skip ahead by multiple sequence numbers. It takes a sequence number $n > H$, and write a new entry in position n if $n > H$

Although the use of A2M abstraction makes protocols immune to equivocation, Clement *et al.* show in [12] that non-equivocation is not enough to provide a one half resilience and that is mandatory to add *transferable authentication* of messages. (*i.e* digital signatures). Therefore, we further assume that digital signatures are unforgeable, hash function is safe. Moreover, we consider that each node has a private key and a public key. We also assume a Public Key Infrastructure (PKI) and that each node is identified by it's A2M public key.

III. Consensus and Repeated Consensus problems

The repeated production of blocks in committee based blockchains can be viewed as a repeated consensus problem. At each height of the blockchain exactly one block is decided and added via a one-shot consensus specified below. The traditional specification has been modified using the validity borrowed from [15] in order to meet the requirements in blockchain systems. That is, a new block is added to the blockchain only if it does not contain transactions that conflict with existing transactions in the blockchain.

Definition 1 (One-Shot Consensus)**.** We say that an algorithm implements One-Shot Consensus if and only if it satisfies the following properties:

- *Termination.* Every correct process eventually decides some value.
- *Integrity.* No correct process decides twice.
- *Agreement.* If there is a correct process that decides a value B, then eventually all the correct processes decide B.
- *Validity[15].* A decided value is valid, it satisfies the predefined predicate denoted $\mathsf{isValid}()$.

The one-shot consensus is needed to agree on the next block that will be appended to the current blockchain. However, a blockchain is a list of blocks cryptographically linked and in order to ensure that nodes agree on the same view of the blockchain an additional abstraction is needed. To this end, we will use the generalisation of the *repeated consensus* resilient to Byzantine failures introduced in [3]. Each obedient (correct) node outputs an infinite sequence of decisions, each decision corresponding to a block in the blockchain analogy. We call the sequence of decisions the *output* of the node.

Definition 2 (Repeated Consensus)**.** An algorithm implements a Byzantine repeated consensus if and only if it satisfies the following properties:

- *brc-Termination.* Every obedient node has an infinite output;
- *brc-Agreement.* If the i^{th} value of the output of an obedient node is B, and the i^{th} value of the output of another obedient node is B', then $B = B'$;
- *brc-Validity.* Each value in the output of any obedient node is valid; it satisfies a predefined predicate.

If an algorithm implements the repeated consensus then each obedient node will have an infinite sequence of decisions (blocks); any two obedient nodes will have the exact same sequence (the same blockchain), and all blocks in the sequence will be valid with respect to the application dependant predicate.

In the following we propose and prove correct one-shot and repeated consensus TenderTee protocols that improve the resilience of Tendermint protocols proposed in [3] by using the lightweight A2M abstraction.

IV. TenderTee description

1) General idea: We present algorithms 1, 2, 3 and 4 that solve one-shot consensus in an eventually synchronous model in presence of Byzantine faulty nodes. We integrate A2M to Tendermint algorithms proposed in [4] in order to increase the protocol resilience.

Each block is characterised by its height h which is the distance in terms of blocks from the genesis block (height 0). For each new height, algorithms 3 and 4 proceeds in $epochs$, and each epoch e consists in three rounds : PRE-PROPOSE, PROPOSE and VOTE. During the PRE-PROPOSE round, the proposer pre-proposes a value v to all validators. During the PROPOSE round, if a validator accepts v then it proposes such value. If a validator receives $enough$ proposals for the same value v then it votes for v during the VOTE round. Finally, if a validator receives $enough$ votes for v, it decides on v. In this case, $enough$ means at least $f+1$ occurences of the same value from $f+1$ different validators and from each validator only the first value delivered for each round is considered. If the proposer is correct then it pre-proposes the same value to all the $f+1$ correct validators. All the $f+1$ correct validators propose such value, it follows that all the $f+1$ correct validators vote for such value and decide for it. If the proposer is Byzantine, once it pre-proposes a value v, it can't propose a different value v' for the same round (see Lemma 4). Each slot in A2M's log is associated with a protocol step. So every process can write only one value for each protocol step (for a given epoch) because each message exchange is associated with an attestation produced by A2M (otherwise the message will be rejected).

Each message in the protocol is sent through A2M-Broadcast (Algorithm 2, see figure 1). A node adds its message in his A2M's log that corresponds to the protocol round (PRE-PROPOSE, PROPOSE or VOTE), and broadcasts it with the attestation produced when call A2M's log interface (with an $advance$ call - see Lines 16, 17 and 18 of Figure 3). Each message received is verified through Algorithm 1 that verifies that the attestation is valid (see Figure 2). To simplify algorithms presentation, we omit attestation verification in Algorithms 3 and 4 (as it is done by Algorithm 1 for each message received).

2) Detailed description: Algorithms 3 and 4 proceed in 3 rounds for a given epoch e at height h. Each protocol message (pre-propose, propose or vote) is sending through A2M-Broadcast (Algorithm 2) which appends the message value to the A2M's log that correspond to the protocol phase and sends the message with the corresponding A2M's attestation produce by the $advance$ call.

We define a $SizeEpoch$ parameter, for each height h, epoch e_i is set to 0. For each (unsuccessful) try of propose a value, epoch e_i is incremented up to a maximum value equal to $2^{SizeEpoch} - 1$. We suppose that $SizeEpoch$ is large enough and that e_i never reaches its maximum.

Logs position: For each new height h, each log position is set to $height \times SizeEpoch$ via and $advance$ call.

- Round PRE-PROPOSE : If the validator p_i is the proposer of the epoch, it pre-proposes its proposal value, otherwise, it waits for the proposal from the proposer. The proposal value of the proposer is its $validValue_i$ if $validValue_i$. If a validator p_j delivers the pre-proposal from the proposer of the epoch, p_j checks the validity of the pre-proposal and that A2M's attestation is valid and if both conditions are verified, he accepts it with respect to the values in $validValue_i$, $lockedValue_i$, $validEpoch_j$ and $lockedEpoch_i$. If the pre-proposal is accepted and valid, p_j sets its proposal $proposal_j$ to the pre-proposal, otherwise it sets it to nil.

Algorithm 1 Secure messages management for node i

```
1:  upon reception of $\langle \text{TYPE}, h, e, \text{message}, att_{message,j} \rangle$ from node $j$ do
2:    if $\nexists c : (\langle \text{TYPE}, h, e, c \rangle, j) \in messagesSet$
      $\wedge verify(att_{message,j}, message, pubKey_j)$ then
3:        /* We suppose the existence of a function verify that takes an attestation, a
          message and a pubKey and return true is attestation is valid for message and
          is produce by j's A2M */
4:      $\text{messagesSet} \leftarrow \text{messagesSet} \cup (\langle \text{TYPE}, h, e, \text{message} \rangle, j)$
```

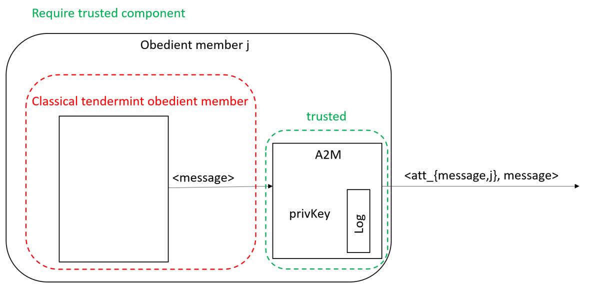
Require trusted component
Obedient member j
Classical tendermint obedient member
trusted
A2M
<message>
privKey
Log
<att_{message,j}, message>

Fig. 1. Send message

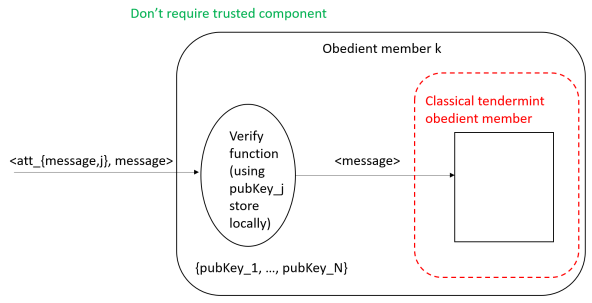
Don't require trusted component
Obedient member k
Classical tendermint obedient member
<att_{message,j}, message>
Verify function (using pubKey_j store locally)
<message>
{pubKey_1, ..., pubKey_N}

Fig. 2. Receive message

Algorithm 3 (part 1) Eventual Synchronous TenderTee for height h executed by i

```
1:   /* There is 3 A2M's log available for i : $logPrepropose_i$, $logPropose_i$ and
     $logVote_i$ */
2:  Initialisation:
3:    $e_i := 0$                                  /* Current epoch number */
4:    $decision_i := nil$      /* Store the decision of the committee member $i$ */
5:    $lockedValue_i := nil$; $validValue_i := nil$
6:    $lockedEpoch_i := -1$; $validEpoch_i := -1$
7:    $proposal_i := \text{getValue}()$      /* Store the value the committee member will
      (pre-)propose */
8:    $v_i := nil$            /* Local variable stocking the pre-proposal if delivered */
9:    $validEpoch_j := nil$      /* Local variable stocking the proposer's validEpoch */
10:   $vote_i := nil$          /* Store the value the committee member will vote for */
11:   $\texttt{timeoutPrePropose} := \Delta_{\text{Pre-propose}}$; $\texttt{timeoutPropose} := \Delta_{\text{Propose}}$;
      $\texttt{timeoutVote} := \Delta_{\text{Vote}}$

12: Round PRE-PROPOSE :
13:   Send phase:
14:     if $decision_i \neq nil$ then
15:       $\forall v, j : (\langle \text{VOTE}, h, e_i, v \rangle, j) \in$
          $\text{messagesSet}_i$, broadcast $\langle \text{VOTE}, h, e_i, v \rangle$
16:       return
17:     if $\text{proposer}(h, e_i) = i$ then
18:       A2M-Broadcast($PRE-PROPOSE, h, e_i, logPrepropose, proposal_i, validEpoch_i$)
          /* Call to trusted hardware, z is a nonce, end return a Lookup attestation :
          $\langle LOOKUP, logPrepropose_i, n, z, proposal_i, w, n', d \rangle$ */
19:   Delivery phase:
20:     set $timerPrePropose$ to $\texttt{timeoutPrePropose}$
21:     while ($timerPrePropose$ not expired)
        $\wedge \neg(\exists v_j, e_j : \text{sentByProposer}(h, e_i, v_j, e_j))$ do
22:       if $\exists v_j, e_j : \text{sentByProposer}(h, e_i, v_j, e_j))$ then
23:         $v_i \leftarrow v_j$                     /* $v_j$ is the value sent by the proposer */
24:         $validEpoch_j \leftarrow e_j$      /* $e_j$ is the $validEpoch$ sent by the proposer */
25:     if $\neg(\exists v, epochProp : \text{sentByProposer}(h, e_i, v, epochProp))$ then
26:       $\texttt{timeoutPrePropose} \leftarrow \texttt{timeoutPrePropose} + 1$
27:   Compute phase:
28:     if f+1$(\langle \text{PROPOSE}, h, validEpoch_j, v_i \rangle)$
        $\wedge\ validEpoch_j \geq lockedEpoch_i$
        $\wedge\ validEpoch_j < e_i$
        $\wedge \texttt{isValid}(v_i)$
       then
29:       $proposal_i \leftarrow v_i$
30:     else
31:       if $\neg\texttt{isValid}(v_i)$
          $\vee(lockedEpoch_i > validEpoch_j \wedge lockedValue_i \neq v_i)$ then
32:         $proposal_i \leftarrow nil$       /* Note that $\texttt{isValid}(nil)$ is set to $\texttt{false}$ */
33:       if $\texttt{isValid}(v_i) \wedge (lockedEpoch_i = -1 \vee lockedValue_i = v_i)$ then
34:         $proposal_i \leftarrow v_i$
```

- Round PROPOSE : During the PROPOSE round, each validator broadcasts (through A2M-Broadcast primitive) its proposal, and collects the proposals sent by other validators (only the one that have a valid A2M log's attestation). After the Delivery phase, validator p_i has a set of proposals, and checks if v, pre-proposed by the proposer, was proposed by at least $f+1$ different validators, if it is the case, and the value is valid, then p_i sets $vote_i$, $validValue_i$ and $lockedValue_i$ to v and updates $lockedEpoch_i$ to the current epoch e_i, otherwise it sets $vote_i$ to nil.
- Round VOTE : In the round VOTE, a correct validator p_i votes $vote_i$ and broadcasts (using A2M-Broadcast primitive) all the proposals it delivered during the current epoch. Then p_i collects all the message that were broadcast. First p_i checks if it has delivered at least $f+1$ of proposal for a value v' pre-proposed by the proposer of the epoch, in that case, it sets $validValue_i$, to that value then it checks if a value v' pre-proposed by the proposer of the current epoch is valid and has at least $f+1$ votes, if it is the case, then p_i decides v' and goes to next height; otherwise it increases the epoch number and updates the value of $proposal_i$, with respect to $validValue_i$.

Algorithm 2 A2M-Broadcast

```
1:  A2M-Broadcast(round,height, epoch,log,message, validEpoch) :

2:  broadcast
    $\langle round, height, epoch, message, validEpoch, advance(log, height \times SizeEpoch + epoch, digest, message) \rangle$
```

V. Correctness of TenderTee in Eventual Synchronous Setting

In this section, we prove the correctness of TenderTee (Algorithms 3 & 4) in an eventually synchronous system. We suppose that $n = 2f+1$ and that each protocol's message sent by a node i is sent with an A2M attestation $< att_{m,i}, m >$.

Lemma 1 (Validity)**.** In an eventually synchronous system, TenderTee verifies the following property: A decided value satisfies the predefined predicate denoted as $\texttt{isValid}()$.

Proof. The proof follows by construction. When an obedient

Algorithm 4 (part 2) Eventual Synchronous TenderTee for height h executed by i

```
1:  Round PROPOSE:
2:    Send phase:
3:      if $proposal_i \neq nil$ then
4:        A2M-Broadcast($PROPOSE, h, e_i, logPropose, proposal_i,$
          $validEpoch_i$)
5:      else
6:        A2M-Broadcast($PROPOSE, h, e_i, logPropose, HeartBeat,$
          $validEpoch_i$)
7:    Delivery phase:
8:      set timerPropose to timeoutPropose
9:      while (timerPropose not expires)
        $\wedge \neg$f+1($\langle$(HeartBeat, PROPOSE)|PROPOSE, $h, e_i\rangle$) do{}   /* Note that
        the HeartBeat messages should be from different committee members */
10:       if $\neg$f+1($\langle$(HeartBeat, PROPOSE)|PROPOSE, $h, e_i\rangle$) then
11:         timeoutPropose $\leftarrow$ timeoutPropose $+ 1$
12:     Compute phase:
13:       if $\exists v'$ : f+1($\langle$PROPOSE, $h, e_i, v'\rangle$)
          $\wedge$isValid($v'$)
          $\wedge$sentByProposer($h, e_i, v'$) then
14:         $lockedValue_i \leftarrow v'$
15:         $lockedEpoch_i \leftarrow e_i$
16:         $validValue_i \leftarrow v'$
17:         $validEpoch_i \leftarrow e_i$
18:         $vote_i \leftarrow v'$
19:       else
20:         $vote_i \leftarrow nil$

21: Round VOTE:
22:   Send phase:
23:     if $vote_i \neq nil$ then
24:       A2M-Broadcast($VOTE, h, e_i, logVote, vote_i, validEpoch_i$)
25:     else
26:
        A2M-Broadcast($VOTE, h, e_i, logVote, HeartBeat, validEpoch_i$)

27:   Delivery phase:
28:     set timerVote to timeoutVote
29:     while (timerVote not expires)
        $\wedge \neg$f+1($\langle$(HeartBeat, VOTE)|VOTE, $h, e_i\rangle$) do{}
30:     if $\neg$f+1($\langle$(HeartBeat, VOTE)|VOTE, $h, e_i\rangle$) then
31:       timeoutVote $\leftarrow$ timeoutVote $+ 1$
32:   Compute phase:
33:     if $\exists v''$ : f+1($\langle$PROPOSE, $h, e_i, v''\rangle$) $\wedge$ isValid($v''$) $\wedge$
        sentByProposer($h, e_i, v''$) then
34:       $validValue_i \leftarrow v''$
35:       $validEpoch_i \leftarrow e_i$
36:     if $\exists v_d, e_d$ : f+1($\langle$VOTE, $h, e_d, v_d\rangle$)
        $\wedge$isValid($v_d$)
        $\wedge decision_i = nil$ then
37:       $decision_i \leftarrow v_d$
38:     else
39:       $e_i \leftarrow e_i + 1$
40:       $v_i \leftarrow nil$
41:       if $validValue_i \neq nil$ then
42:         $proposal_i \leftarrow validValue_i$
43:       else
44:         $proposal_i \leftarrow$ getValue()
```

committee member decides a value (Line 37 of Algorithm 4), it checks before if that value is valid (Line 36 of Algorithm 4). Therefore, an obedient committee member only decides a valid value. □

Lemma 2 (Integrity)**.** In an eventually synchronous system, TenderTee verifies the following property: No obedient committee member decides twice.

Proof. The proof follows by construction. Before deciding (Lines 36 - 37), an obedient committee member i checks if there is not already a value decided ($decision_i = nil$) for the current height ($i.e.$ line 36). If there is already a value decided ($decision_i \neq nil$), there is no decision (Lines 38 - 44). No obedient committee member decides twice. Moreover, note that an obedient committee member exits the algorithm, the epoch after it has decide (Line 14 of Algorithm 3). □

Lemma 3. In an eventually synchronous system, TenderTee verifies the following property: An obedient committee member proposes and votes only once per epoch.

Proof. We prove this lemma by construction. In Algorithm 4, an obedient committee member proposes (Line 4) and votes only once during the corresponding round (Line 24 or Line 26). At the end of the VOTE round, a committee member changes epoch (Line 39). Therefore, it cannot propose nor vote for that epoch any more. □

Lemma 4. Processes propose and vote at most once per epoch.

Proof. A process cannot produce two different logs for the same step as sequence number is monotonic and each height in logPrepropose (respectively logPropose or logVote) is associated with a round and an epoch. □

Lemma 5. In an eventually synchronous system, TenderTee verifies the following property: At most one value can be proposed by at least $f + 1$ committee members per epoch, and at most one value can be voted at least $f + 1$ times per epoch.

Proof. We prove this lemma by contradiction. Let v, v' such that $v \neq v'$. Since there are $2f + 1$ committee members in the system, if v or v' gets at least $f + 1$ proposals (resp. votes), it means that at least 1 committee members propose (resp. vote) for both v and v' which contradicts Lemma 4 □

Lemma 6. Let v be a value, e an epoch, and the set $L^{v,e} = \{i$: i obedient $\wedge$ $lockedValue_i = v \wedge lockedEpoch_i = e$ at the end of epoch $e\}$. In an eventually synchronous system, TenderTee verifies the following property: If $|L^{v,e}| \geq 1$ then no obedient committee member i will have $lockedValue_i \neq v \wedge lockedEpoch_i \geq e$, at the end of each epoch $e' > e$; moreover, a committee member in $L^{v,e}$ only proposes v or nil for each epoch $e' > e$.

Proof. Let v be a value, e an epoch, and $L^{v,e} = \{i$: i obedient $\wedge$ $lockedValue_i = v \wedge lockedEpoch_i = e$ at the end of epoch $e\}$, we assume that $|L^{v,e}| \geq 1$. We prove the theorem by induction:

- *Initialisation:* At the end of epoch e, by assumption, we have that $|L^{v,e}| \geq 1$. There is an obedient committee member in that set, say i ($i \in L^{v,e}$). It means that i updates $lockedValue_i$ to v during epoch e, therefore i delivered $f+1$ proposals for the value v (Lines 13 - 15 of Algorithm 4). By Lemma 5, at most one value can have at least $f+1$ proposals during epoch e, and since v has at least $f + 1$ proposals, no obedient committee member j can update $lockedValue_j$ to a value $v' \neq v$ during epoch e. At the end of e, $lockedValue_j \neq v \vee lockedEpoch_j < e$.
- Induction: Let $a \geq 1$, we assume that $\forall i \in L^{v,e}$, $lockedValue_i = v$ at the end of each epoch between e

and $e + a$, we also assume that if a value was proposed at least $f + 1$ times during these epochs it was either v or *nil*. We prove that at the end of epoch $e + a + 1$, no obedient committee member j will have $\mathit{lockedValue}_j = v' \wedge \mathit{lockedEpoch}_j = e + a + 1$ with $v' \neq v$.

Let $i \in L^{v,e}$ such that i delivers a pre-proposal for v, then i will set $\mathit{proposal}_i$ to v; it will propose v since $\mathit{lockedValue}_i = v$ (Lines 28 - 34 of Algorithm 3 & Line 4 of Algorithm 4), in any other case, if i does not deliver a pre-proposal, or delivers a pre-proposal for a value $v' \neq v$, it will set $\mathit{proposal}_i$ to *nil* and will propose *nil* (Lines 28 - 34 of Algorithm 3 & Line 4 of Algorithm 4), since $\texttt{isValid}(\mathit{nil}) = \texttt{false}$ and by assumption, there is no $e' \in \{e, \ldots, e + a\}$ where there were at least $f + 1$ proposals for a value $v' \neq v$, and $\mathit{lockedEpoch}_i \geq e$. All committee members in $L^{v,e}$ will then propose v or *nil* during epoch $e + a + 1$. By Lemma 3, obedient committee members only propose once per epoch, at least $f + 1$ committee members (the ones in $L^{v,e}$) propose v or *nil*; since messages cannot be forged, the only values that can get at least $f + 1$ proposals for the epoch $e + a + 1$ are v and *nil*. If an obedient committee member j delivers at least $f+1$ proposals for v, it sets $\mathit{lockedValue}_j$ to v and $\mathit{lockedEpoch}_j$ to $e + a + 1$ (Lines 13 - 15 of Algorithm 4); otherwise, it does not change $\mathit{lockedValue}_j$ nor $\mathit{lockedEpoch}_j$ (Line 20 of Algorithm 4). At the end of epoch $e+a+1$, there is no obedient committee member j such that $\mathit{lockedValue}_j \neq v \wedge \mathit{lockedEpoch}_j = e+a+1$. Moreover, committee members in $L^{v,e}$, only propose v or *nil* during epoch $e + a + 1$.

We proved that if $|L^{v,e}| \geq 1$, no obedient committee member i will have $\mathit{lockedValue}_i \neq v \wedge \mathit{lockedEpoch}_i \geq e$; moreover a committee member in $L^{v,e}$ only proposes v or *nil* for each epoch $e' > e$.

□

Lemma 7 (Agreement)**.** In an eventually synchronous system, TenderTee verifies the following property: If there is an obedient committee member that decides a value v, then eventually all the obedient committee members decide v.

Proof. Let i be an obedient committee member. Without loss of generality, assume that i is the first obedient committee member that decides, and assume that it decides value v during epoch e. At time t where i decided, no other node has decided, even those having a bigger epoch number. To decide, i delivered at least $f+1$ votes for v for epoch e. Since there are less than f Byzantine committee members, and by Lemma 3 obedient committee members can only vote once per epoch, so at least 1 obedient committee members voted for v during epoch e, so we have $|L^{v,e}| = |\{j : j \text{ obedient} \wedge \mathit{lockedValue}_j = v \wedge \mathit{lockedEpoch}_j = e \text{ at the end of epoch } e\}| \geq 1$. By Lemma 6 a committee member in $L^{v,e}$ only proposes v or *nil* during each epoch after e, and no obedient committee member j will have $\mathit{lockedValue}_i \neq v \wedge \mathit{lockedEpoch}_i \geq e$. Thanks to the broadcast guarantees (brb-Termination-1), all obedient committee members will eventually deliver the $f+1$ votes for v from epoch e; since when an obedient committee member decides, it sends back all votes it delivered than makes it decided (Line 14 of Algorithm 3).

If an obedient committee member j does not decide before delivering these votes, when eventually it delivers them, it will decide v (Lines 36 - 37 of Algorithm 4). Otherwise, it means that j decides before delivering the votes from epoch e.

By contradiction, we assume that j decides a value $v' \neq v$ during an epoch $e' > e$, so j delivered at least $f + 1$ votes for v' during epoch e' (Lines 36 - 37 of Algorithm 4). Since an obedient committee member only votes once by Lemma 3, there are less than f Byzantine committee members and the messages are unforgeable, at least 1 obedient committee members voted for v' during epoch e'.

An obedient committee member votes a non-*nil* value if that value was proposed at least $f + 1$ times during the current epoch (Lines 13 - 26 of Algorithm 4). By Lemma 4 processes proposes at most once, there are less than f Byzantine committee members and the messages are unforgeable, so at least 1 obedient committee members proposed v' during e'. Since $e' > e$ and $|L^{v,e}| \geq 1$, by Lemma 6 there are at least 1 obedient committee members that proposed v or *nil* during epoch e'. Even if all the f committee members remaining proposes v', there cannot be $f+1$ proposals for v', which is a contradiction. So j cannot decide $v' \neq v$ after epoch e and we assume that e is the first epoch where an obedient committee member decides. □

Lemma 8. In an eventually synchronous system, if there is an epoch after which when an obedient committee member broadcasts a message during a round, it is delivered by all obedient committee members during the same round, TenderTee verifies the following property: If an obedient committee member i updates $\mathit{lockedValue}_i$ to a value v during epoch e, then at the end of the epoch e, all obedient committee members have $\mathit{validValue} = v$ and $\mathit{validEpoch} = e$.

Proof. We prove this lemma by construction.

Let e be the epoch after which when an obedient committee member broadcasts a message during a round r, it is delivered by all obedient committee members during the same round r. Let i be an obedient committee member, we assume that at the end of epoch $e' \geq e$, i has $\mathit{lockedValue}_i = v$ and $\mathit{lockedEpoch}_i = e'$, it means that i delivered at least $f + 1$ proposals for v during epoch e' (Lines 13 - 15 of Algorithm 4). Thanks to the reliable broadcast guarantees, and since all messages are propagated,all obedient committee members will deliver these proposals for v, in the worst-case in the VOTE round. Let j be an obedient committee member since j will deliver at least $f + 1$ proposals for v and epoch e' during the VOTE round, it will set $\mathit{validValue}_j = v$ and $\mathit{validEpoch}_j = e'$ (Lines 33 - 35 of Algorithm 4). □

Lemma 9 (Termination)**.** In an eventually synchronous system, TenderTee verifies the following property: Every obedient committee member eventually decides some value.

Proof. By construction, if an obedient committee member does not deliver more than $f + 1$ messages (or 1 from the proposer in the PRE-PROPOSE round) from different committee members during the corresponding round, it increases the duration of its round, so eventually during the synchronous period of the system all the obedient committee members will deliver the pre-proposals, proposals and votes from obedient committee members respectively during the PRE-PROPOSE, PROPOSE and the VOTE round; and messages delivered by an obedient committee member will be delivered by the others at most in the following round. Let e be the first epoch after that time.

If an obedient committee member decides before e, by Lemma 7 all obedient committee members eventually decide, which ends the proof. Otherwise, at the beginning of epoch e, no obedient committee member decides yet. Let i be the proposer of epoch e. First, we assume that i is obedient and pre-propose v; v is valid since $\mathsf{getValue}()$ always return a valid value (Line 7 of Algorithm 3 & Line 44 of Algorithm 4), and $\mathit{validValue}_i$ is always valid (Lines 13 & 33 of Algorithm 4). We have 2 cases:

- Case 1: At the beginning of epoch e, $|\{j : j \text{ obedient} \wedge (\mathit{lockedEpoch}_j \leq \mathit{validEpoch}_i \vee \mathit{lockedValue}_j = v)\}| \geq f + 1$.

  Let j be an obedient committee member where the condition $\mathit{lockedEpoch}_j \leq \mathit{validEpoch}_i \vee \mathit{lockedValue}_j = v$ holds. After the delivery of the pre-proposal v from i, j will update $\mathit{proposal}_j$ to v (Lines 28 - 34 of Algorithm 3). During the PROPOSE round, j proposes v (Line 4 of Algorithm 4), since there are at least $f + 1$ similar obedient committee members (included j), they will all propose v, and all obedient committee members will deliver at least $f+1$ proposals for v (Line 8 of Algorithm 4).

  Obedient committee members will set their variable *vote* to v (Lines 13 - 4 of Algorithm 4), then will vote v, and they will deliver all the votes (at least $f + 1$) from this epoch (Lines 24, 26 & 27 of Algorithm 4). Since we assume that no obedient committee members decided yet, and since they each deliver at least $f + 1$ votes for v, they will decide v (Lines 36 - 37 of Algorithm 4).
- Case 2: At the beginning of epoch e, $|\{j : j \text{ obedient} \wedge (\mathit{lockedEpoch}_j \leq \mathit{validEpoch}_i \vee \mathit{lockedValue}_j = v)\}| < f + 1$.

  Let j be an obedient committee member where the condition $\mathit{lockedEpoch}_j > \mathit{validEpoch}_i \wedge \mathit{lockedValue}_j \neq v$ holds. When i will make the pre-proposal, j will set $\mathit{proposal}_j$ to *nil* (Line 32 of Algorithm 3) and will propose *nil* (Line 4 of Algorithm 4).

  By counting only the proposed values of the obedient committee members, no value will have at least $f + 1$ proposals for v. There are two cases:

  - No obedient committee member delivers at least $f+1$ proposals for v during the PROPOSE round, so they will all set their variable *vote* to *nil*, then they will vote *nil* and go to the next epoch without changing their state (Lines 20 & 24 - 27 & 38 - 44 of Algorithm 4).
  - If some obedient committee members delivers at least $f+1$ proposals for v during the PROPOSE round, *i.e.*, some Byzantine committee members send proposals for v to those committee members.

    As in the previous case, they will vote for v, and since there are $f + 1$ of them, all obedient committee members will decide v. Otherwise, there are less than $f+1$ obedient committee members that deliver at least $f+1$ proposals for v. Only them will vote for v (Line 24 or 26 of Algorithm 4). Without Byzantine committee members, there will be less than $f + 1$ votes for v, no obedient committee member will decide (Lines 36 - 37 of Algorithm 4) and they will go to the next epoch; if Byzantine committee members send votes for v to an obedient committee member such that it delivers at least $f + 1$ votes for v during VOTE round, then the obedient committee member will decide (Lines 36 - 37 of Algorithm 4), and by Lemma 7 all obedient committee members will eventually decide.

    Let k be one of the obedient committee members that delivers at least $f + 1$ proposals for v during the PROPOSE round, it means that $\mathit{lockedValue}_k = v$ and $\mathit{lockedEpoch}_k = e$. It follows by Lemma 8 that at the end of epoch e, all obedient committee members will have $\mathit{validValue} = v$ and $\mathit{validEpoch} = e$.

  If there is no decision, either no obedient committee member changes its state, or all obedient committee members change their state and have the same *validValue* and *validEpoch*; therefore, eventually a proposer of an epoch will satisfy Case 1, and that ends the proof.

If i, the proposer of epoch e, is Byzantine and more than $f+1$ obedient committee members delivered the same message during PRE-PROPOSE round, and the pre-proposal is valid, the situation is like i was obedient. Otherwise, there are not enough obedient committee members that delivered the pre-proposal, or if the pre-proposal is not valid, then there will be less than $f+1$ obedient committee members that will propose that value, which is similar to the case 2.

Since proposers are selected in a round-robin fashion, an obedient committee member will eventually be the proposer, and obedient committee members will decide. □

Theorem 10. In an eventually synchronous system, TenderTee implements the one-shot consensus specification.

Proof. The proof follows directly from Lemmas 1, 2, 7 and 9. By Lemma 1, we show that TenderTee satisfies Validity, by Lemma 7, we show that TenderTee satisfies Agreement, and by Lemma 9, we show that TenderTee satisfies Termination. □

VI. TENDERTEE REPEATED CONSENSUS ALGORITHM

We now present the repeated consensus module of TenderTee.

```
Function TenderTee-Repeated-Consensus($\Pi$);
%Repeated consensus for the set $\Pi$ of nodes%

Init:
 (1) $h \leftarrow 1$ %Height%; $B \leftarrow \perp$; $C \leftarrow \perp$ %Set of committee members%;
 (2) $commitsReceived_i^h \leftarrow \emptyset$; $toReward_i^h \leftarrow \emptyset$; $TimeOutCommit \leftarrow \Delta_{\text{Commit}}$;
--------------------------------------------------------------------------------
while (true) do
 (3) $B \leftarrow \perp$;
 (4) $C \leftarrow$ committeeMembers($h$); %Application and blockchain dependant%
 (5) if ($i \in V$) then
 (6)     $B \leftarrow$ TenderTee-consensus($h, V, toReward_i^{h-1}$);
 (7)   % Consensus A2M function for the height $h$%
 (8)     trigger A2M-broadcast $\langle \texttt{COMMIT}, (B, h)_i, att_i \rangle$;
 (9)   else
 (10)    wait until ($\exists B'$ : |MoreThanHalf($B'$, $commitsReceived_i^h$)|); %Wait f+1 (verified)
commits for the same block%
 (11)    $B \leftarrow B'$;
 (12) endif
 (13) set $timerCommit$ to $TimeOutCommit$;
 (14) wait until($timerCommit$ expired);
 (15) trigger decide($B$);
 (16)  advance(logPrepropose, h*SizeEpoch - 1, digest, 0);
 (17)  advance(logPropose, h*SizeEpoch - 1, digest, 0);
 (18)  advance(logVote, h*SizeEpoch - 1, digest, 0);
 (19)   %Update log $h$%
 (20)  $h \leftarrow h + 1$;
endwhile
--------------------------------------------------------------------------------
upon event delivery $\langle \texttt{COMMIT}, (B', h')_j, att_j \rangle$:
 (21) if ($(B', h')_j \notin commitsReceived_i^{h'}$) $\wedge$ ($j \in$ committeeMembers($h'$))
$\wedge verify$(COMMIT, (B',h')$_j$, $att_j$, $pk_j$)) then
 (22)    $commitsReceived_i^{h'} \leftarrow commitsReceived_i^{h'} \cup (B', h')_j$;
 (23)    $toReward_i^{h'} \leftarrow toReward_i^{h'} \cup j$;
 (24)    trigger A2M-broadcast $\langle \texttt{COMMIT}, (B', h')_j, att_j \rangle$;
 (25) endif
```

Fig. 3. TenderTee Repeated Consensus algorithm for Obedient node i.

a) Detailed description of the algorithm: We describe in Figure 3 the Tendermint algorithm for the repeated consensus (Definition 2). Each message is sent through A2M-Broadcast (Algorithm 2) and each message received is verified with Algorithm 1 (verify A2M's attestation validity). For a node i, the algorithm proceeds as follows:

- i computes the set of committee members for the current height;
- If i is a committee member, then it calls the TenderTee-consensus function (the one-shot TenderTee consensus) solving the consensus using A2M for the current height, then broadcasts the decision (using A2M-broadcast function), and sets B to that decision;
- Otherwise, if i is not a committee member, it waits for at least $n/2$, or equivalently $f+1$, commits from the same block and sets B to that block;
- In any case, it sets the timer to *TimeOutCommit* to collect more commits and lets it expire. Then i decides B and goes to the next height.

Whenever i delivers a commit, it broadcasts it using A2M-broadcast function (Lines 21 - 25 of Figure 3).

b) Data structures: The integer h represents the current height of the node. C is the current set of committee members. B is the variable that will be set to the block to be appended. $commitsReceived_i^h$ is the set containing all the commits i delivered for the height h. $toReward_i^h$ is the set containing the committee members from which i delivered commits for the height h. *TimeOutCommit* represents the time a committee member has for collecting commits after an instance of consensus. *TimeOutCommit* is set to the default value Δ_{Commit}.

c) Functions: Let $Height = \mathbb{N}^*$ be the set of all heights, let *Commits* be a set of all possible commits, and let $\mathbb{B}$ be the set containing all possible blocks.

We also recall that Π_ρ is the set of nodes in the system run.

- committeeMembers : $\Pi \times Height \rightarrow 2^{\Pi_\rho}$ is an application dependent and deterministic selection function which gives the set of committee members for a given height with respect to the blockchain history. We have $\forall h \in Height$, $|\text{committeeMembers}(h)| = n$.
- TenderTee-consensus : $Height \times 2^{\Pi_\rho} \times 2^{Commits} \rightarrow \mathbb{B}$ is a consensus algorithm. It outputs for the node the decision of the TenderTee-consensus (Definition 1) among the committee members.
- MoreThanHalf : $\mathbb{B} \times 2^{Commits} \rightarrow \{\texttt{false}, \texttt{true}\}$ is a predicate which checks if there are commits from at least $f+1$ different committee members for a given block in the given set.
- `isValid` : $\mathbb{B} \rightarrow \{\texttt{false}, \texttt{true}\}$ is an application-dependent predicate that is satisfied if the given block is valid. If there is a block B such that `isValid`$(B) =$ `true`, we say that B is valid. We note that for any non-block, we set `isValid` to false, (*i.e.*, `isValid`$(\perp) =$ `false`), the validity of the block depends on the

blockchain and the application, and `isValid` is known by all nodes.

VII. Correctness of TenderTee Repeated Consensus

In this section, we prove the correctness of the repeated consensus module of TenderTee, when assuming that the consensus algorithm used (Line 6 of Figure 3) is correct (Definition 1). In the proofs below, the lines mentioned refer to the lines in the algorithm presented in Figure 3.

Lemma 11 (brc-Termination)**.** In an eventually synchronous system, and assuming that TenderTee one-shot consensus is correct, the TenderTee Repeated Consensus algorithm verifies the following property: Every obedient node has an infinite output.

Proof. By contradiction, let i be an obedient node and we assume that i has a finite output. Two scenarios are possible, either i cannot go to a new height, or from a certain height h it outputs only $\perp$.

- If i cannot progress, one of the following cases is satisfied:
  - The function TenderTee-consensus does not terminate (Line 6), which is a contradiction since it violates the Termination property of the TenderTee-consensus (Termination in Definition 1).
  - i waits an infinite time for receiving enough commits (Line 10), which cannot be the case thanks to the A2M-broadcast guarantees and the eventual synchronous assumption, all the obedient committee members terminate the function TenderTee-consensus and broadcast their commit.
- If i does not decide at each height (Line 15), it means that from a given height, i only outputs $\perp$. Let height h be the first such height, and let $h' > h$; (i) either i is a committee member for h' and the function TenderTee-consensus returns $\perp$ (Lines 5 & 6), or (ii) i is not a committee member for h' but delivered at least $f+1$ commits for $\perp$ (Lines 10 & 24).
  - (i) If TenderTee-consensus returns the value $\perp$, it means by the Validity property (Definition 1) of the consensus that $\texttt{isValid}(\perp) = \texttt{true}$, which is a contradiction with the definition of the function `isValid`.
  - (ii) Only committee members commit, and each of them broadcasts its commit (Lines 5 - 8), and $f < n/2$. Since non-committee members collect at least $n/2$ commits, it means that i delivered a commit from at least one obedient committee member. By the Validity property of the TenderTee-consensus (Definition 1), obedient nodes only decide/commit on valid value, and $\perp$ is not valid, which is a contradiction.

Therefore, if i is an obedient node, then it has an infinite output. □

Lemma 12 (brc-Agreement)**.** In an eventually synchronous system, and assuming that the TenderTee-consensus function is correct (i.e. it verifies the one-shot consensus specification), TenderTee Repeated Consensus Algorithm verifies the following property: If the h^{th} value of the output of an obedient node is B, then B is the h^{th} value of the output of any other obedient node.

Proof. We prove this lemma by construction. Let i and j be two obedient nodes, and let h be a height. Two cases are possible:

- i and j are committee members for the height h, so both calls the function TenderTee-consensus (Lines 5 & 6). By Agreement property of the TenderTee-consensus (Lemma 7), i and j decide the same value and then output that same value (Line 15).
- At least one of i and j is not a committee member for the height h. Without loss of generality, we assume that i is not a committee member for the height h. Since all the obedient committee members commit the same value, let say B, thanks to the Agreement property of the TenderTee-consensus (Definition 1), and since they broadcast their commit (Line 8), eventually there will be more than $f+1$ commits for B. So no other value $B' \neq B$ can be present at least $f+1$ times in the set $\mathit{commitReceived}_i^h$. Therefore, i outputs the same value B as obedient committee members (Line 10). If j is a committee member, that ends the proof. If j is not a committee member, then by the same argument, j outputs the same value B. Hence, both i and j output the same value B for height h.

□

Lemma 13 (brc-Validity)**.** In an eventually synchronous system, and assuming that TenderTee-consensus function is correct (i.e. it verifies the one-shot consensus specification), TenderTee Repeated Consensus Algorithm verifies the following property: Each value in the output of any obedient node is valid; it satisfies the predefined predicate denoted `isValid`.

Proof. We prove this lemma by construction. Let i be an obedient node, and assume that the h^{th} value of the output of i is B. If i decides a value (Line 15), then that value has been set during the execution and for that height (Line 3).

- If i is a committee member for the height h, then B is the value returned by the function TenderTee-consensus. By the Validity property of the TenderTee-consensus (Definition 1), we have $\texttt{isValid}(B) = \texttt{true}$.
- Let h be a height, and B be the h^{th} value of the output of a node j. If j is not a committee member for the height h, it means that it delivered more than $f+1$ signed commits from the committee members for the value B (Lines 5 - 8 and 21 - 25), hence at least one obedient committee member committed B, and by the Validity property of the TenderTee-consensus (Definition 1), we have $\texttt{isValid}(B) = \texttt{true}$.

Each value in the output of an obedient node satisfies the predicate `isValid`. □

Theorem 14. In an eventually synchronous system, and assuming that the TenderTee-consensus function is correct (i.e. it verifies the one-shot consensus specification), the TenderTee Repeated Consensus algorithm implements the Repeated Consensus.

Proof. Assuming that the TenderTee-consensus function verifies the one-shot consensus specification, the proof follows directly from Lemmas 11, 12 and 13. By Lemma 11, we show that the TenderTee Repeated Consensus algorithm satisfies brc-Termination, by Lemma 12, we show that the TenderTee Repeated Consensus algorithm satisfies brc-Agreement, and by Lemma 13, we show that the TenderTee Repeated Consensus algorithm satisfies brc-Validity. □

VIII. Conclusion

This work presents TenderTee, a enhanced version of Tendermint blockchain. TenderTee uses a lightweight A2M trusted abstraction in order to increase the Byzantine resilience of the original one-shot and repeated consensus Tendermint protocols from one third to one half. By reducing the number of needed nodes this protocol is appealing for industrialisation since the number of nodes to be maintained in order to guarantee agreement is drastically reduced.

Although, TenderTee is not the first in the blockchain context to use trusted abstractions in order to improve the blockchain resilience from one third to one half, our contribution has the merit to address one-shot and repeated consensus modules, both necessary in guaranteeing the linearizability of any committee based blockchain.

An important further research direction is to automate our approach in order to enhance any committee based blockchain with one half resilience. Finally, the automatic design of correct by construction of a committee based blockchain having optimal resilience and optimal time complexity is the final target of this line of research.

References

[1] Abraham, I., Gueta, G., Malkhi, D.: Hot-stuff the linear, optimal-resilience, one-message BFT devil. CoRR **abs/1803.05069** (2018), `http://arxiv.org/abs/1803.05069`

[2] Abraham, I., Malkhi, D., Nayak, K., Ren, L., Spiegelman, A.: Solidus: An incentive-compatible cryptocurrency based on permissionless byzantine consensus. CoRR, abs/1612.02916 (2016)

[3] Amoussou-Guenou, Y., Pozzo, A.D., Potop-Butucaru, M., Tucci Piergiovanni, S.: Correctness of tendermint-core blockchains. In: Cao, J., Ellen, F., Rodrigues, L., Ferreira, B. (eds.) 22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China. LIPIcs, vol. 125, pp. 16:1–16:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2018)

[4] Amoussou-Guenou, Y., Pozzo, A.D., Potop-Butucaru, M., Tucci Piergiovanni, S.: Dissecting tendermint. In: Atig, M.F., Schwarzmann, A.A. (eds.) Networked Systems - 7th International Conference, NETYS 2019, Marrakech, Morocco, June 19-21, 2019, Revised Selected Papers. Lecture Notes in Computer Science, vol. 11704, pp. 166–182. Springer (2019). https://doi.org/10.1007/978-3-030-31277-0_11, `https://doi.org/10.1007/978-3-030-31277-0_11`

[5] Androulaki, E., Barger, A., Bortnikov, V., Cachin, C., Christidis, K., Caro, A.D., Enyeart, D., Ferris, C., Laventman, G., Manevich, Y., Muralidharan, S., Murthy, C., Nguyen, B., Sethi, M., Singh, G., Smith, K., Sorniotti, A., Stathakopoulou, C., Vukolic, M., Cocco, S.W., Yellick, J.: Hyperledger fabric: a distributed operating system for permissioned blockchains. In: Proceedings of the Thirteenth EuroSys Conference, EuroSys 2018, Porto, Portugal, April 23-26, 2018. pp. 30:1–30:15 (2018)

[6] Astefanoaei, L., Chambart, P., Pozzo, A.D., Rieutord, T., Tucci Piergiovanni, S., Zalinescu, E.: Tenderbake - A solution to dynamic repeated consensus for blockchains. In: Gramoli, V., Sadoghi, M. (eds.) 4th International Symposium on Foundations and Applications of Blockchain 2021, FAB 2021, May 7, 2021, University of California, Davis, California, USA (Virtual Conference). OASIcs, vol. 92, pp. 1:1–1:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2021)

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

[8] Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on bft consensus. arXiv preprint arXiv:1807.04938 (2018)

[9] Buterin, V.: Ethereum whitepaper, `https://ethereum.org`

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

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

[12] Clement, A., Junqueira, F., Kate, A., Rodrigues, R.: On the (limited) power of non-equivocation. In: Kowalski, D., Panconesi, A. (eds.) ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012. pp. 301–308. ACM (2012). https://doi.org/10.1145/2332432.2332490, `https://doi.org/10.1145/2332432.2332490`

[13] Correia, M., Neves, N.F., Veríssimo, P.: 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. pp. 174–183. IEEE Computer Society (2004). https://doi.org/10.1109/RELDIS.2004.1353018, `https://doi.org/10.1109/RELDIS.2004.1353018`

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

[15] Crain, T., Natoli, C., Gramoli, V.: Red belly: A secure, fair and scalable open blockchain. In: 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021. pp. 466–483. IEEE (2021)

[16] Decker, C., Seidel, J., Wattenhofer, R.: Bitcoin Meets Strong Consistency. In: Proceedings of the 17th International Conference on Distributed Computing and Networking Conference (ICDCN) (2016)

[17] Distler, T., Cachin, C., Kapitza, R.: Resource-efficient byzantine fault tolerance. IEEE Trans. Computers **65**(9), 2807–2819 (2016). https://doi.org/10.1109/TC.2015.2495213, `https://doi.org/10.1109/TC.2015.2495213`

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

[19] Kapitza, R., Behl, J., Cachin, C., Distler, T., Kuhnle, S., Mohammadi, S.V., Schröder-Preikschat, W., Stengel, K.: Cheapbft: resource-efficient byzantine fault tolerance. In: Felber, P., Bellosa, F., Bos, H. (eds.) European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference 2012, EuroSys '12, Bern, Switzerland, April 10-13, 2012. pp. 295–

308. ACM (2012). https://doi.org/10.1145/2168836.2168866, https://doi.org/10.1145/2168836.2168866

[20] Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In: Proceedings of the 25th USENIX Security Symposium (2016)

[21] Kwon, J.: Tendermint: Consensus without mining. https://tendermint.com/static/docs/tendermint.pdf (2014), [Online; accessed 2020 July 10]

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

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

[24] Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System p. 9 (2008), https://bitcoin.org/bitcoin.pdf

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

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