Asynchronous Byzantine Consensus with Trusted Monotonic Counters

Yackolley Amoussou-Guenou[1], Maurice Herlihy[2], and Maria Potop-Butucaru[3]

[1] Université Paris-Panthéon-Assas, CRED, Paris, France
[2] Brown University Computer Science Dept, Providence RI 02912, USA
[3] Sorbonne Université, CNRS, LIP6, Paris, France

Abstract. The paper promotes a new design paradigm for Byzantine tolerant distributed algorithms using trusted abstractions (oracles) specified in a functional manner. The contribution of the paper is conceptual. The objective here is to design distributed fundamental algorithms such as reliable broadcast and asynchronous byzantine consensus using trusted execution environments and to help designers to compare various solutions on a common ground. In this framework we revisit the Bracha's seminal work on Asynchronous Byzantine Consensus. Our solution uses trusted monotonic counters abstraction and tolerates t Byzantine processes in a system with n processes, $n \geq 2t+1$. The keystone of our construction is a novel and elegant Byzantine Reliable Broadcast algorithm resilient to $t < n$ Byzantine processes that uses an unique trusted monotonic counter (at the initiator).

Keywords: Asynchronous, Byzantine agreement, Trusted abstractions

1 Introduction

Byzantine Agreement problem introduced in the seminal paper [29] has been studied for decades in various models ranging from synchronous (e.g. [28]), to asynchronous (e.g. [7, 35]).

The blockchains era revived the interest for the problem. In Algorand [12] for example the authors propose solutions for synchronous and partially synchronous systems constructing their solution on top of Verifiable Random Function abstraction [33]. In [1] the authors define and address the complexity issues. More recently, in [18] the authors use recent advances in cryptography in order to reduce the word complexity. In [36] authors analyze the complexity of randomized algorithms for Binary consensus under different common coins. However, none of these works focus on breaking the $3f + 1$ bound for the asynchronous settings.

One of the main problems to be solved in order to break the $3f + 1$ bound in Byzantine prone environments is the ability of Byzantine nodes to equivocate (i.e. a faulty node may send different messages to different nodes). *Trusted execution components* (e.g. A2M [14], TrInc [31], USIG [38]) are reputed to be powerful tools for avoiding equivocation.

Trusted components have been heavily used to increase the resilience of PBFT like protocols [10] and therefore they found applications in many recent blockchain algorithms. Among the first to attack the execution of PBFT in trusted execution environments are Correia *et al.* [19, 20]. They introduced TTCB wormhole, 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 PBFT can support a fraction of half Byzantine nodes. In other words, they circumvent FLP [24] impossibility by relying on a synchronous and secure distributed subsystem. Although this method allows to increase fault tolerance, its practical implementation is too difficult to set up.

In the quest of practicality, Chun *et al.* [14] introduced Attested Append-Only Memory (A2M), a trusted system that targets to remove from the faulty nodes the ability to equivocate. An A2M is a set of trusted ordered append-only logs that provide an attestation for each entry. Furthermore, they propose PBFT-EA, a modified PBFT [9] that uses A2M for each message exchanged; the message is appended to a log and the attestation produced is sent along 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. However, A2M needs large secure storage (for the log).

An alternative to this is the use of a monotonic counters implemented in a tamperproof module. Levin *et al.* propose TrInc ([31]), 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.

Later, Veronese *et al.* in [38] propose a specific monotonic trusted counter, 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 service offers two functions: one that returns a certificate, and one that validates certificates. These certificates are based on a secure counter: the counter value is never duplicated, and successive counter values are successive integers. This service has to be implemented in a tamper-proof module. Furthermore, they propose two algorithms (MinBFT and its speculative version MinZyzzyva) that implement following the same pattern as PBFT *state machine replication* which consists of replicating a service in a group of servers with strong consistency guaranties. Each server maintains a set of state variables, which are modified by a set of operations. The operations are deterministic and atomic. The initial state of the servers is the same. The properties that the two proposed algorithms satisfy are: safety– all correct servers execute the same requests in the same order; liveness– all correct clients' requests are eventually executed. MinBFT and its speculative version MinZyzzyva implement state machine replication using USIG in systems with a minority of Byzantines.

Another line of research combines speculative methods and trusted environments (e.g. CheapBFT [27] and ReBFT [23]). In a 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.

Interestingly, in the context of blockchains, the use of trusted environments in order to increase the resilience is very recent (e.g. *VABA* [40], the asynchronous version of *HotStuff*, *Damysus* [22], and TenderTee [3]). The first use of it was proposed in [40]. The authors enhance HotStuff blockchain in order to tolerate a minority of corruptions in a PBFT style protocol. Their algorithm builds on top of an underlying expander graphs and use threshold signatures. The small trusted component used has A2M flavour. Another recent paper introduces *Damysus* ([22]), a PBFT protocol that uses trusted environments to improve Hotstuff resilience. In this paper, the authors introduce two trusted services *Checker* and *Accumulator* that respectively increase resilience and reduce latency. The correctness of the proposed protocol has been proven for the partially synchronous environments. In [3] the authors propose a methodology to automatically plug A2M in the Tendermint protocol to increase its resilience. Furthermore, they prove that the same methodology can be applied to the repeated consensus abstraction with the same results in terms of resilience. This work still needs a *partially synchronous* execution environment.

In this paper we are interested in designing trusted distributed algorithms (e.g. reliable broadcast, consensus) having optimal resilience to Byzantine faults in asynchronous settings without the use of threshold signatures or assumptions on the underlying topology as in [40]. We focus here in solving Byzantine Consensus problem [7] in *asynchronous* settings. Probabilistic consensus enhanced with trusted environments in order to avoid equivocation seems to be a good compromise in this direction. One of the key building blocks of our Probabilistic Byzantine Consensus is a deterministic Byzantine Reliable broadcast algorithm tolerant to $t < n$ Byzantine processes. *Byzantine reliable broadcast* is a fundamental problem in fault-tolerant distributed systems. It consists of ensuring that a correct initiator process broadcasts its value to all correct processes, even in the presence of malicious Byzantine processes. For decades, Byzantine Reliable Broadcast has been at the core of various consensus protocols, and more recently, at the core of certain blockchains.

Algorithms solving the Byzantine Reliable Broadcast problem have been proposed in various environments: with static or dynamic Byzantine nodes, or in conjunction with transient faults. Byzantine Reliable Broadcast solutions (e.g. [5,6,25,32,37]) achieve resilience of at least $n \geq 3t+1$ processes, where t is the maximum number of Byzantine processes. In this paper we continue the line of work opened by [21] which proposes a reliable broadcast algorithm tolerant with $n \geq 2f+1$ processes where f is the number of Byzantine faults. Contrarily to us, they use failure detectors. In [39] authors proposed an algorithm similar to ours but which only tolerates $f < n/3$ Byzantine processes. This bound has been ameliorated to $f < n/2$ in [2]. Our work improves further the bound to $f < n$.

Tackling another problem, [30] proposes an algorithm for the atomic broadcast problem which uses a trusted execution environment with a monotonic counter similarly to us. However, differently from us the focus of the authors is not on the minimal trust assumptions is expected from the trusted execution environment. Moreover, their solution builds on the failure detector based reliable broadcast of [21].

Another line of research related to the use of trusted execution environments for improving the resilience of distributed algorithms is the line initiated by Clement *et al.* [17]. Although the trusted execution environments make protocols immune to equivocation (where the initiator sends different messages to different processes), Clement *et al.* [17] show that non-equivocation is not enough to provide $n \geq 2f+1$ resilience, nor to support the equivalent of digital signatures. The authors prove that it is possible to use non-equivocation to transform any protocol that works under the crash fault model into a protocol that tolerates Byzantine faults by adding the ability to guarantee the transferable authentication of network messages (e.g., using digital signatures). In [4] the authors revisited and extend the work of Clement *et al.* [17] providing a transformer with a polynomial communication overhead instead of exponential and covering also randomized distributed algorithms.

Our approach is more conceptual in the sens that we would like to design Byzantine tolerant distributed algorithms on top of trusted oracles that allow to abstract low level (trusted hardware) assumptions and provide a separation between the conceptual and the technical part. Notice that, the trusted oracles we define may have different implementations. Specifying the functional property that is needed at the application level we need a generic solution that can be implemented using different technologies or physical architectures. Oracles in distributed systems have been used for decades starting with the seminal work on failure detectors [11] that paved the way of designing protocols with formal proofs of correctness and the investigation of precise lower bounds and impossibility results.

Our contribution. Our work extends the line of research related to using trusted execution environments in order to increase the resilience of distributed algorithms in Byzantine prone environments. The novelty of our approach is in identifying the minimal assumptions on trusted abstraction needed to solve fundamental building blocks in distributed computing (e.g. consensus and reliable broadcast). In this paper we revisit the original simple and elegant solutions proposed by Bracha [7] in an environment where processes are equipped with a trusted monotonic counter abstraction that provides a non-falsifiable, verifiable, unique, monotonic, and sequential counter. The use of this abstraction in a clever way allows us to first implement a Reliable Broadcast resilient to $t < n$ Byzantine processes in asynchronous communication environments. Moreover, on top of this optimized Reliable Broadcast primitive we construct a Probabilistic Byzantine Consensus resilient to t Byzantine processes in systems of size $n \geq 2t+1$. Differently from the transformer-based approach where the transformations are obtained with an important communication overhead (exponential in the case of [17] and polynomial in the case of [4]), our design uses only a constant overhead with the respect of the Bracha's original solutions while improving both the resilience and the number of communication rounds. Our work opens a new direction of research similar to oracle-based distributed computing [11]. The trusted environment is encapsulated in a trusted abstraction (oracle) providing a set of guaranties. The methodology used in this paper for Byzantine Reliable Broadcast and Probabilistic Byzantine Consensus can be easily extended to other Byzantine tolerant distributed algorithms.

2 System Model and Problems definition

We consider a set of n asynchronous sequential processes, of which up to t can be *Byzantine*, meaning they can deviate from the given protocol. The rest are *correct* processes.

Processes communicate by exchanging messages through an asynchronous network. We make the usual assumptions that there is a public key infrastructure (PKI) where public keys are distributed, each process has a (universally known) public key a matching private key, and each message is signed by its creator. Messages are not lost or spuriously generated. Each process can send messages directly to any other process, and each process can identify the sender of every message it receives.

We assume that the communication is asynchronous, and that processes have access to a communication primitive which ensures that any message m sent by a correct process is received by every correct process in a finite (but unknown) time.

Following Bracha, [7], we define *Byzantine Reliable Broadcast* and *Probabilistic Byzantine Consensus* as follows:

Definition 1 (Byzantine Reliable Broadcast). *We say that an algorithm implements* Byzantine reliable broadcast *if:*

- brb-CorrectInit: *If the initiator is correct, all correct processes deliver the initiator's value.*
- brb-ByzantineInit: *If the initiator is Byzantine, then either no correct process delivers any value, or all correct processes deliver the same value.*

Definition 2 (Probabilistic Byzantine Consensus). *A protocol implements* probabilistic Byzantine consensus *if:*

- *Agreement: all correct processes decide on the same value.*
- *Validity: if all processes start with the same value v, then all correct processes decide on v.*
- *Termination (probabilistic). The probability that a correct process is undecided after r rounds approaches zero as r approaches infinity.*

3 Trusted Monotonic Counter Object

The Trusted Monotonic Counter Oracle abstraction TMC-Object defined below is the core of our novel Byzantine Reliable Broadcast protocol that supports t Byzantine failures among n processes, where $n > t$, a great improvement on the classical $n \geq 3t + 1$ algorithms.

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 a *unique identifier*. The certificate certifies that the returned unique identifier was created by the tamper-proof TMC-Object object for the message m. The unique identifier is essentially a reading of the monotonic counter trustedCounter, which is incremented whenever **get_certificate(m)** is called. The TMC-Object object guarantees the following properties:

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

Note that the sequentiality property implies *Strict Monotonicity:* TMC-Object will always assign an identifier that is strictly greater than the previous one.

To send a message u certified by TMC-Object, a process p first invokes the TMC-Object, which creates a certificate $\mathcal{C}_{(p,u)}$ corresponding to the value of the trustedCounter c_p, then the process sends the tuple $(u, \mathcal{C}_{(p,u)}, c_p)$, which can be verified by any other process receiving the message. Each invocation to 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.

4 Asynchronous Byzantine Reliable Broadcast

We describe Algorithm 1, a Byzantine reliable broadcast algorithm which uses a unique TMC-Object(with the counter initialized at 0), where the initiator of the broadcast uses the TMC-Object-Send operation to send the value to be broadcast. Our Byzantine Reliable Broadcast is resilient to any number, $t < n$, of Byzantine processes.

The protocol works in two sequential asynchronous steps. In the initial step (Step 0) of the protocol, when a process p wants to broadcast a value v, it TMC-Object-Sends an initial message for v ($< initial, v >$) 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.

In Step 1, when receiving a valid initial message from the proposer, say with value v, a process sends back the initiator message (with the certificate and value of the counter). 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 1 are considered valid, since a message with a higher counter value means that the initiator already sent another value. More generally, the value of the counter should be 1 more than the previously known value of the counter.

Algorithm 1: Byzantine Reliable Broadcast with an unique TMC-Object

```
[1] TMC-Object_Broadcast v  Step 0   if p is the initiator then   TMC-Object-Send
    < initial, v, id_initiator > to all Equivalent to Send (m, C_(initiator,m), c_initiator) to all, where
    m =< initial, u, id_initiator >
Step 1  Upon reception of (< initial, u, id_initiator >, C, 1) message The message of the initiator should
    have value 1 as the trusted counter of the initiator and related to the certificate
    Send (< initial, u, id_initiator >, C, 1) to all The process sends back the initiator's message with the
    associated certificate and trusted counter to all processes
    Deliver u
```

Theorem 1. *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 with a unique TMC-Object and in $O(n^2)$ messages.*

Proof. First, notice that when a correct process p delivers a message u, it means that p received a valid initiator message u. Eventually every other correct process q will receive that same initiator message thanks to the network guarantees. Therefore, if there is no equivocation, they must deliver that same value.

If the initiator is correct, then all other correct processes eventually receive and deliver the initiator's message.

It remains to show that when two different correct processes deliver a message, they deliver the same message.

Recall that when a correct process receives a (TMC-Object) message, it checks the value of the trusted counter associated to the message. Moreover, it checks whether the message and the value of the counter are coherent with the corresponding certificate.

By way of contradiction, assume there exist two correct processes p and q which deliver distinct messages $u \neq v$. Without loss of generality, assume that p delivers message u and q delivers message v. To do so, p must have received $(< initial, u, id\_initiator >, \mathcal{C}_{initiator}, 1)$ and q must have received $(< initial, v, id\_initiator >, \mathcal{C}'_{initiator}, 1)$, which is impossible because of the *Uniqueness* property of the TMC-Object. Therefore, p and q must have delivered the same message, which concludes the proof.

In Algorithm 1, the number of messages sent is exactly $n + n^2$, which is a $O(n^2)$. In more details, the initiator sends n messages (one to each other process), and each process when receiving the initiator message sends it back which is $n \times n$.

5 Asynchronous Byzantine Consensus

In this section we propose a Byzantine Probabilistic Consensus algorithm (Algorithm 2) resilient to $t < n/2$ Byzantine processes. The keystone of the solution is the Byzantine Reliable Broadcast resilient to any number, $t < n$, of Byzantine processes (Algorithm 1).

Remark 1. Notice that when there are two different initiators, there are two instances of the broadcast.

Algorithm 2: Probabilistic Byzantine Consensus

[1] Probabilistic_Consensusv_i **Step** $2k$ `TMC-Object_Broadcast`(v_i) here, we say that the process proposes v_i `wait until` $n - t$ messages from Step $2k$ or $\exists k' \leq k, v : (\#(\mathtt{d}, v) received from Step 2k' + 1) > n/2$: `if` $\exists v : (\# v received from Step 2k > n/2)$ `then` $v_i = (\mathtt{d}, v)$ v is tagged by symbol `d`, *i.e.,* the process is ready to decide v during this round

`if` $\exists k' \leq k, v : (\#(\mathtt{d}, v) received from Step 2k' + 1) > n/2$ `then` **Decide** v and `Terminate`

Step $2k+1$ `TMC-Object_Broadcast`(v_i) `wait until` $n - t$ messages from Step $2k+1$ or $\exists k' \leq k, v : (\#(\mathtt{d}, v) received from Step 2k' + 1) > n/2$: `if` $\exists k' \leq k, v : (\#(\mathtt{d}, v) received from Step 2k' + 1) > n/2$ `then` **Decide** v and `Terminate`

`if` $\exists v : (\#(\mathtt{d}, v) received from Step 2k + 1) \geq 1)$ `then` $v_i = v$

`otherwise` $v_i =$ `flip`()The `flip` function is made thanks to VRF to ensure non-manipulability

$k = k + 1$ `go to` Round k More specifically, `go to` **Step** $2k$

We present a probabilistic Byzantine consensus protocol in Algorithm 2 inspired from the Bracha's probabilistic consensus [7]. The protocol works in asynchronous sequential rounds where each round, $k \geq 0$ is split in two sequential steps, $2k$ and $2k + 1$.

In the each even step $2k$, of round $k \geq 0$, after proposing its local value, each process waits to collect $n - t$ messages from this step. If the process delivers the same message, say v, from strictly more than $n/2$ times from different processes, then the process locally tags v, by setting its local value to $(\mathtt{d}, v)$. $\mathtt{d}$ is the tag marker. It means the process is ready to decide v and is letting the other processes know about it.

In the each odd step $2k+1$ of round $k \geq 0$, after broadcasting its local value, each process waits to collect $n-t$ messages from the current step. If among the delivered messages, the process delivers at least one tagged value from the current step, say $(\mathtt{d}, v)$, the process sets its local value to the value v which will be propose during the next step. Otherwise if the process delivers no tagged value from the current step, it randomly selects a value which will be its proposal for the next step.

In each step, if a process delivers the tagged message $(\mathtt{d}, v)$ from strictly more than $n/2$ different processes from the same round, the process decides value v and terminates.

To prove the correctness of Algorithm 2, we emphasize the main guarantees we rely on. These guarantees need not be taken as axiomatic, because they could be implemented as described below. Moreover, we also make explicit for which property each guarantee is necessary.

Hypothesis 1 *Byzantine processes cannot lie about the output of their randomness.*

Hypothesis 1 is achievable by the use of verifiable random functions (VRF [13, 34]). We rely on this hypothesis to prove the (probabilistic) Termination property.

Hypothesis 2 *All messages sent are causally valid.*

We rely on Hypothesis 2 to provide the Agreement property. Hypothesis 2 can be implemented by requiring each process to accompany each message with the prior messages that caused the process to compute that message. In the case of Algorithm 2, this hypothesis simply means that a message sent at a step $s > 0$ should be accompanied by the messages the process received at the previous step $s-1$. Any message that is not causally valid is ignored.

Lemma 1. *Suppose there are n processes, of which at most t are Byzantine. If $n \geq 2t+1$, for any $k \geq 0$, if all correct processes at step $2k$ proposes value v, then at the end of step $2k$ all correct processes will have as value either $(\mathtt{d}, v)$ or v.*

Proof. Assume there exists a correct process q that has a value u or $(\mathtt{d}, u)$ with $u \neq v$ at the end of step $2k$.

Case 1: q received more than $n/2$ messages with value u, which is impossible since all correct processes started with the same value v and the number of Byzantine processes is limited to $t < n/2$.

Case 2: q changed its value to some value u. This is impossible since q received $n-t$ messages. Among the $n-t$ messages there at most t messages with an erroneous value u forged by fewer than $n/2$ Byzantine processes, and at least one message from correct processes with value v. In this case, q keeps its initial value v.

Lemma 2. *Let n be the number of processes, and t be an upper bound of the Byzantine processes. If $n \geq 2t+1$, for any $k \geq 0$, if all processes at step $2k$ propose value v, then at the end of the step $2k$ all correct processes will have as value $(\mathtt{d}, v)$.*

Proof. By Lemma 1 if all correct processes start with value v then at the end of step $2k$ all correct processes will have as value either $(\mathtt{d}, v)$ or v.

Assume that there exists a correct process q which ends step $2k$ with v instead of $(\mathtt{d}, v)$. This means that q did not receive more than $n/2$ messages with the same value v. This is impossible since q waits for $n-t$ messages. Over the $n-t$ messages either all of them come from correct processes (hence same value v) or some of them (up to t) come from Byzantine processes. However, Byzantine processes start with v (by assumption) and cannot change their initial value without proof of modification.

Overall, if all processes start with value v then at the end of step $2k$ all correct processes will have value $(\mathtt{d}, v)$.

Lemma 3. *For any $k \geq 0$, if all processes propose value v at step $2k$, then at step $2k+1$, if a Byzantine process broadcasts a value then its only causally valid message is $(\mathtt{d}, v)$.*

Proof. By Lemma 2, a correct process at the end of step $2k$ will have as value $(\mathtt{d}, v)$. Hence in the next step it will broadcast $(\mathtt{d}, v)$. If at step $2k+1$ a Byzantine process broadcasts a message, then that message should be $(\mathtt{d}, v)$.

Lemma 4 (Validity). *For any $k \geq 0$, if all processes propose value v at step $2k$, then all correct processes decide on v.*

Proof. If all processes start with the same value v then at the end of step 0 (which is a $2k$ step with $k = 0$) all correct processes end with the value $(\mathtt{d}, v)$ (see Lemma 2). In step 1 (which is $2k+1$) each correct process sends $(\mathtt{d}, v)$ and waits for $n - t$ messages from step $2k+1$. Based on Hypothesis #2 and Lemma 3, Byzantine processes either send $(\mathtt{d}, v)$ or stay silent. Therefore, all correct processes will gather at least $n - t$ messages with v, hence all correct processes decide v.

Lemma 5 (Agreement). *Let n be the number of processes, and t be an upper bound of the Byzantine processes. If $n \geq 2t + 1$, if two correct processes decide, they decide the same value. More generally, if one correct process decides, all correct processes decide the same value.*

Proof. Let p and q be two correct processes that decided. Let p decide at some round k and q at some round $k' > k$. It follows that p received more than $n/2$ messages $(\mathtt{d}, v)$ in round k. Therefore, all correct processes receive at least one $(\mathtt{d}, v)$ message in round k, in particular, q received at least one $(\mathtt{d}, v)$ in round k. Since all processes send the causal proof of their messages (to satisfy Hypothesis #2), it follows that all processes, including q, will start round $k+1$ with the same value v. Following Lemma 4 all correct processes, including q, decide v in round $k+1$.

Lemma 6 (Termination). *The probability that a correct process is undecided after r rounds approaches zero as r approaches infinity.*

Proof. As shown in Lemma 4, if all processes propose the same value v, then all correct processes eventually decide value v.

If one correct process decides, then thanks to Lemma 5, all correct processes eventually decide. Therefore, let us assume that no correct process decides yet. There are two cases. Either (i) no process enters line 3 and all processes randomly flip their value, or (ii) at least one correct process received one $(\mathtt{d}, v)$ message from its current step and so does not flip its value.

- First, consider the case where no process ever receives a $(\mathtt{d}, v)$ message corresponding to its current step and round. Therefore, they will always flip their coins at each round. Notice, however, that thanks to Lemma 4, when all processes will have the same proposal, then all correct processes decide that value, which guarantees Termination. The probability of such an event happening for each round k is low, *i.e.*, p^n, where p the probability of having any value. Since the Byzantine processes cannot control (nor lie about, by Hypothesis #1) their random input value, the probability of having a value for any Byzantine process is the same as for any correct process. However, the probability this event never occurs in an infinite execution is the limit of $(1 - p^n)^k$ when k goes to infinity, which is equal to 0, since $p \in (0, 1)$, and $n > 0$. Hence, if processes always flip their value, they will terminate with probability 1.

– There remains to discuss what happens if at least one process does not flip their value.
If a process does not flip its value, it means that such process received a $(\mathtt{d}, v)$ message from its current step (say, $2k+1$). It means no other value ($v' \neq v$) could be sent as $(\mathtt{d}, v')$, in fact, by Hypothesis 2, since Byzantine processes must causally justify their votes, they cannot send a $(\mathtt{d}, v')$ with $v' \neq v$. Therefore, all the processes that will receive such message and will not flip will have v as value for the next round. Notice that if one correct process decides, then by Lemma 5, all correct processes eventually decide. Suppose there is no decision yet. Either all correct processes do not flip and have the same value v for the next round (and terminate at that round by Lemma 4), or some do flip.
Therefore, if some processes flip repeatedly, and the others do not flip but have the same value; then as in the case above, with probability 1 over the infinite execution, there will be a situation where the processes that flipped will end up with the same value as those that do not flip, and hence, by Lemma 4 they will terminate. Notice that the probability of having all processes having the same value must be higher in this case (than in the above case) since some are already proposing the same value.

In both cases, therefore, with probability 1, all correct processes eventually terminate.

Theorem 2. *Let n be the number of processes, and t be an upper bound of the Byzantine processes. If $n \geq 2t+1$, Algorithm 2 implements Probabilistic Byzantine Consensus with $O(n^3)$ messages.*

6 Conclusions and Discussions

In this paper we propose a novel solution for Probabilistic Byzantine Consensus in an environment where processes are equipped with a trusted monotonic counter abstraction that provides a non-falsifiable, verifiable, unique, monotonic, and sequential counter. Our solution tolerates t Byzantine processes with $n \geq 2t+1$ in asynchronous settings. The keystone of our construction is an elegant deterministic Byzantine Reliable Broadcast algorithm that uses a single trusted monotonic counter in a clever way and implements a Reliable Broadcast resilient to $t < n$ Byzantine processes. Our work continues the line of research opened by [19, 20] and continued by [26, 38] that promotes the benefits of using trusted hardware in improving the resilience of distributed algorithms. The novelty of our study resides in investigating the minimal trusted abstractions or minimal trusted properties needed to solve two fundamental problems in distributed computing in Byzantine prone environments. Our work has similar flavor to the line of work of oracle-based distributed computing [11] and opens several research directions including the modelisation of distributed algorithms executed in trusted environments or their composition [8, 15, 16]. We believe that our oracle-based approach may be easily refined in order to address other distributed problems.

References

1. Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Peter Robinson and Faith Ellen, editors, *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, PODC 2019, Toronto, ON, Canada, July 29 - August 2, 2019*, pages 337–346. ACM, 2019. `doi:10.1145/3293611.3331612`.
2. Yackolley Amoussou-Guenou, Lionel Beltrando, Maurice Herlihy, and Maria Potop-Butucaru. Byzantine reliable broadcast with one trusted monotonic counter. *IACR Cryptol. ePrint Arch.*, page 774, 2024. URL: `https://eprint.iacr.org/2024/774`.
3. Lionel Beltrando, Maria Potop-Butucaru, and José Alfaro. Tendertee: Increasing the resilience of tendermint by using trusted environments. In *24th International Conference on Distributed Computing and Networking, ICDCN 2023, Kharagpur, India, January 4-7, 2023*, pages 90–99. ACM, 2023. `doi:10.1145/3571306.3571394`.

4. Naama Ben-David, Benjamin Y. Chan, and Elaine Shi. Revisiting the power of non-equivocation in distributed protocols. In Alessia Milani and Philipp Woelfel, editors, *PODC '22: ACM Symposium on Principles of Distributed Computing, Salerno, Italy, July 25 - 29, 2022*, pages 450–459. ACM, 2022.
5. Silvia Bonomi, Jérémie Decouchant, Giovanni Farina, Vincent Rahli, and Sébastien Tixeuil. Practical byzantine reliable broadcast on partially connected networks. In *2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS)*, pages 506–516. IEEE, 2021.
6. Silvia Bonomi, Giovanni Farina, and Sébastien Tixeuil. Reliable broadcast despite mobile byzantine faults. In Alysson Bessani, Xavier Défago, Junya Nakamura, Koichi Wada, and Yukiko Yamauchi, editors, *27th International Conference on Principles of Distributed Systems, OPODIS 2023, December 6-8, 2023, Tokyo, Japan*, volume 286 of *LIPIcs*, pages 18:1–18:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: `https://doi.org/10.4230/LIPIcs.OPODIS.2023.18`, `doi:10.4230/LIPICS.OPODIS.2023.18`.
7. Gabriel Bracha. Asynchronous byzantine agreement protocols. *Inf. Comput.*, 75(2):130–143, 1987. `doi:10.1016/0890-5401(87)90054-X`.
8. Ran Canetti, Ling Cheung, Dilsun Kirli Kaynar, Nancy A. Lynch, and Olivier Pereira. Compositional security for task-pioas. In *20th IEEE Computer Security Foundations Symposium, CSF 2007, 6-8 July 2007, Venice, Italy*, pages 125–139. IEEE Computer Society, 2007.
9. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third USENIX Symposium on Operating Systems Design and Implementation (OSDI), New Orleans, Louisiana, USA, February 22-25, 1999*, pages 173–186. USENIX Association, 1999. URL: `https://dl.acm.org/citation.cfm?id=296824`.
10. Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.
11. Tushar Deepak Chandra and Sam Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.
12. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. URL: `https://doi.org/10.1016/j.tcs.2019.02.001`, `doi:10.1016/J.TCS.2019.02.001`.
13. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019. URL: `https://doi.org/10.1016/j.tcs.2019.02.001`, `doi:10.1016/J.TCS.2019.02.001`.
14. Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007*, pages 189–204. ACM, 2007. `doi:10.1145/1294261.1294280`.
15. Pierre Civit and Maria Potop-Butucaru. Brief announcement: Composable dynamic secure emulation. In Kunal Agrawal and I-Ting Angelina Lee, editors, *SPAA '22: 34th ACM Symposium on Parallelism in Algorithms and Architectures, Philadelphia, PA, USA, July 11 - 14, 2022*, pages 103–105. ACM, 2022.
16. Pierre Civit and Maria Potop-Butucaru. Dynamic probabilistic input output automata. In Christian Scheideler, editor, *36th International Symposium on Distributed Computing, DISC 2022, October 25-27, 2022, Augusta, Georgia, USA*, volume 246 of *LIPIcs*, pages 15:1–15:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022.
17. Allen Clement, Flavio Junqueira, Aniket Kate, and Rodrigo Rodrigues. On the (limited) power of non-equivocation. In *ACM Symposium on Principles of Distributed Computing, PODC '12, Funchal, Madeira, Portugal, July 16-18, 2012*, pages 301–308. ACM, 2012. `doi:10.1145/2332432.2332490`.
18. Shir Cohen and Idit Keidar. Brief announcement: Subquadratic multivalued asynchronous byzantine agreement WHP. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 39:1–39:6. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
19. Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *23rd International Symposium on Reliable Distributed Systems (SRDS 2004), 18-20 October 2004, Florianpolis, Brazil*, pages 174–183. IEEE Computer Society, 2004. `doi:10.1109/RELDIS.2004.1353018`.
20. Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. BFT-TO: intrusion tolerance with less replicas. *Comput. J.*, 56(6):693–715, 2013. `doi:10.1093/comjnl/bxs148`.
21. Miguel Correia, Giuliana Santos Veronese, and Lau Cheuk Lung. Asynchronous byzantine consensus with 2f+1 processes. In Sung Y. Shin, Sascha Ossowski, Michael Schumacher, Mathew J. Palakal, and Chih-Cheng Hung, editors, *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), Sierre, Switzerland, March 22-26, 2010*, pages 475–480. ACM, 2010. `doi:10.1145/1774088.1774187`.
22. Jérémie Decouchant, David Kozhaya, Vincent Rahli, and Jiangshan Yu. DAMYSUS: streamlined BFT consensus leveraging trusted components. In *EuroSys '22: Seventeenth European Conference on Computer Systems, Rennes, France, April 5 - 8, 2022*, pages 1–16. ACM, 2022. `doi:10.1145/3492321.3519568`.
23. Tobias Distler, Christian Cachin, and Rüdiger Kapitza. Resource-efficient byzantine fault tolerance. *IEEE Trans. Computers*, 65(9):2807–2819, 2016. `doi:10.1109/TC.2015.2495213`.

24. Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. In *Proceedings of the Second ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 21-23, 1983, Colony Square Hotel, Atlanta, Georgia, USA*, pages 1–7. ACM, 1983. doi:10.1145/588058.588060.
25. Rachid Guerraoui, Jovan Komatovic, Petr Kuznetsov, Yvonne-Anne Pignolet, Dragos-Adrian Seredinschi, and Andrei Tonkikh. Dynamic byzantine reliable broadcast [technical report]. *arXiv preprint arXiv:2001.06271*, 2020.
26. Suyash Gupta, Sajjad Rahnama, Shubham Pandey, Natacha Crooks, and Mohammad Sadoghi. Dissecting BFT consensus: In trusted components we trust! In Giuseppe Antonio Di Luna, Leonardo Querzoni, Alexandra Fedorova, and Dushyanth Narayanan, editors, *Proceedings of the Eighteenth European Conference on Computer Systems, EuroSys 2023, Rome, Italy, May 8-12, 2023*, pages 521–539. ACM, 2023.
27. Rüdiger Kapitza, Johannes Behl, Christian Cachin, Tobias Distler, Simon Kuhnle, Seyed Vahid Mohammadi, Wolfgang Schröder-Preikschat, and Klaus Stengel. Cheapbft: resource-efficient byzantine fault tolerance. In *European Conference on Computer Systems, Proceedings of the Seventh EuroSys Conference 2012, EuroSys '12, Bern, Switzerland, April 10-13, 2012*, pages 295–308. ACM, 2012. doi:10.1145/2168836.2168866.
28. Valerie King and Jared Saia. Breaking the $O(n^2)$ bit barrier: Scalable byzantine agreement with an adaptive adversary. *J. ACM*, 58(4):18:1–18:24, 2011.
29. Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982. doi:10.1145/357172.357176.
30. Marc Leinweber and Hannes Hartenstein. Brief announcement: Let it TEE: asynchronous byzantine atomic broadcast with n $\geq$ 2f+1. In Rotem Oshman, editor, *37th International Symposium on Distributed Computing, DISC 2023, October 10-12, 2023, L'Aquila, Italy*, volume 281 of *LIPIcs*, pages 43:1–43:7. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.DISC.2023.43.
31. Dave Levin, John R. Douceur, Jacob R. Lorch, and Thomas Moscibroda. Trinc: Small trusted hardware for large distributed systems. In *Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2009, April 22-24, 2009, Boston, MA, USA*, pages 1–14. USENIX Association, 2009. URL: http://www.usenix.org/events/nsdi09/tech/full_papers/levin/levin.pdf.
32. Alexandre Maurer and Sébastien Tixeuil. Self-stabilizing byzantine broadcast. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems*, pages 152–160. IEEE, 2014.
33. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science, FOCS '99, 17-18 October, 1999, New York, NY, USA*, pages 120–130. IEEE Computer Society, 1999. doi:10.1109/SFFCS.1999.814584.
34. Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, FOCS '99, page 120, USA, 1999. IEEE Computer Society. doi:10.1109/SFFCS.1999.814584.
35. Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with t < n/3, o(n2) messages, and O(1) expected time. *J. ACM*, 62(4):31:1–31:21, 2015. doi:10.1145/2785953.
36. Achour Mostefaoui, Matthieu Perrin, and Julien Weibel. Randomized consensus: Common coins are not the holy grail! Technical report, LS2N-Nantes Université, 2024.
37. Michel Raynal. *Fault-tolerant message-passing distributed systems: an algorithmic approach.* springer, 2018.
38. Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, Lau Cheuk Lung, and Paulo Veríssimo. Efficient byzantine fault-tolerance. *IEEE Trans. Computers*, 62(1):16–30, 2013. doi:10.1109/TC.2011.221.
39. Roger Wattenhofer. Distributed systems. Lecture notes, https://disco.ethz.ch/courses/hs21/distsys/lnotes/DistSys_Script.pdf, 2022.
40. Sravya Yandamuri, Ittai Abraham, Kartik Nayak, and Michael K. Reiter. Communication-efficient BFT protocols using small trusted hardware to tolerate minority corruption. *IACR Cryptol. ePrint Arch.*, page 184, 2021. URL: https://eprint.iacr.org/2021/184.