As easy as ABC: Optimal (A)ccountable (B)yzantine (C)onsensus is easy!

Pierre Civit
Sorbonne University, CNRS, LIP6
pierre.civit@lip6.fr

Seth Gilbert
NUS Singapore
seth.gilbert@comp.nus.edu.sg

Vincent Gramoli
University of Sydney, EPFL
vincent.gramoli@sydney.edu.au

Rachid Guerraoui
École Polytechnique Fédérale de Lausanne (EPFL)
rachid.guerraoui@epfl.ch

Jovan Komatovic
École Polytechnique Fédérale de Lausanne (EPFL)
jovan.komatovic@epfl.ch

***Abstract*—It is known that the agreement property of the Byzantine consensus problem among n processes can be violated in a non-synchronous system if the number of faulty processes exceeds $t_0 = \lceil n/3 \rceil - 1$ [10], [19]. In this paper, we investigate the *accountable* Byzantine consensus problem in non-synchronous systems: the problem of solving Byzantine consensus whenever possible (e.g., when the number of faulty processes does not exceed t_0) and allowing correct processes to obtain proof of culpability of (at least) $t_0 + 1$ faulty processes whenever correct processes disagree. We present four complementary contributions:**

1) **We introduce $\mathcal{ABC}$: a simple yet efficient transformation of any Byzantine consensus protocol to an accountable one. $\mathcal{ABC}$ introduces an overhead of only two all-to-all communication rounds and $O(n^2)$ additional bits in executions with up to t_0 faults (i.e., in the common case).**
2) **We define the accountability complexity, a complexity metric representing the number of accountability-specific messages that correct processes must send. Furthermore, we prove a tight lower bound. In particular, we show that any accountable Byzantine consensus protocol incurs cubic accountability complexity. Moreover, we illustrate that the bound is tight by applying the $\mathcal{ABC}$ transformation to any Byzantine consensus protocol.**
3) **We demonstrate that, when applied to an optimal Byzantine consensus protocol, $\mathcal{ABC}$ constructs an accountable Byzantine consensus protocol that is (1) optimal with respect to the communication complexity in solving consensus whenever consensus is solvable, and (2) optimal with respect to the accountability complexity in obtaining accountability whenever disagreement occurs.**
4) **We generalize $\mathcal{ABC}$ to other distributed computing problems besides the classic consensus problem. We characterize a class of agreement tasks, including reliable and consistent broadcast [5], that $\mathcal{ABC}$ renders accountable.**

I. Introduction

Ensuring both safety ("nothing bad ever happens") and liveness ("something good eventually happens") of a wide variety of distributed Byzantine problems is impossible if the number of Byzantine processes exceeds a certain predefined threshold [19]. This limitation motivated researchers to investigate *accountable* variants of these problems [9], [23]. The accountable variant of a problem $\mathcal{P}$ consists in (1) solving problem $\mathcal{P}$ under the appropriate assumptions (e.g., whenever the number of Byzantine processes does not exceed the threshold), and (2) allowing all correct participants to detect some fraction of culprits if the safety of problem $\mathcal{P}$ is violated. Accountability in distributed systems is important since it discourages bad behaviors. If some malicious behavior is guaranteed to result in apprehension and punishment, malicious processes are much less likely to carry out an attack in the first place, thus strengthening the security.

This paper focuses on obtaining accountability in Byzantine consensus protocols that operate in non-synchronous systems. The Byzantine consensus problem [19] is defined among n processes while tolerating up to $t_0 = \lceil n/3 \rceil - 1$ Byzantine (malicious) processes. A process initially *proposes* a value and eventually *decides* a value such that the following properties hold:

- (Liveness) *Termination:* All correct processes eventually decide.
- (Safety) *Agreement:* All correct processes decide the same value.
- (Safety) *Validity:* If all correct processes propose the same value, only that value can be decided by a correct process.

The conjunction of the aforementioned properties can only be ensured if the number of faulty processes does not exceed t_0 [19]. If faulty processes indeed overpopulate the system, any of these properties might be violated. This work focuses on cases when a violation of the agreement property occurs. Specifically, we take a closer look at the *accountable* Byzantine consensus problem. A process initially proposes and later decides a value (as in the Byzantine consensus problem) and *detects* some faulty processes. Formally, the accountable Byzantine problem is solved if and only if the following properties are ensured:

- *Termination:* If the number of faulty processes does not exceed t_0, then all correct processes eventually decide.
- *Agreement:* If the number of faulty processes does not exceed t_0, then all correct processes decide the same value.
- *Validity:* If the number of faulty processes does not exceed t_0 and all correct processes propose the same value, only that value can be decided by a correct

process.

- *Accountability:* If two correct processes decide different values, then every correct process eventually detects at least $t_0 + 1$ faulty processes and obtains proof of culpability of all detected processes.

Importantly, *only* disagreement triggers accountability: if correct processes do not decide, accountability is not ensured.

A. Contributions

The contributions of the paper are fourfold:

1) We present a generic and simple transformation - $\mathcal{ABC}$ – that enables any Byzantine consensus protocol to obtain accountability. Additionally, our transformation is efficient: it introduces an overhead of only two all-to-all communication rounds and $O(n^2)$ exchanged bits of information in all executions with up to t_0 faulty processes (i.e., in the common case).
$\mathcal{ABC}$ owns its simplicity and efficiency to the observation that the composition presented in Algorithm 1 solves the Byzantine consensus problem. Indeed, if the number of faults does not exceed t_0, all correct processes eventually decide the same value from Byzantine consensus (line 3). Therefore, all correct processes eventually receive $n-t_0$ matching CONFIRM messages (line 5) and, thus, decide (line 6). The critical mechanism, which is illustrated in Algorithm 1, is that faulty processes *must* send conflicting CONFIRM messages in order to cause disagreement. Hence, whenever correct processes disagree, an exchange of received CONFIRM messages is sufficient to obtain accountability.

Algorithm 1 Intuition Behind $\mathcal{ABC}$ Transformation

```
1: function $propose(v)$ do
2:     ▷ $bc$ is any Byzantine consensus protocol
3:     $v' \leftarrow bc.propose(v)$
4:     broadcast $[\text{CONFIRM}, v']$
5:     wait for $[\text{CONFIRM}, v']$ from $n - t_0$ processes
6:     return $v'$
```

2) We define the *accountability complexity*, a novel complexity metric representing the number of accountability-specific messages that correct processes must send. Furthermore, we prove a tight lower bound on the accountability complexity. In particular, we show that any accountable Byzantine consensus protocol incurs cubic accountability complexity. Moreover, we illustrate that the bound is tight by applying our $\mathcal{ABC}$ transformation to any Byzantine consensus protocol.
3) We demonstrate that when applied to an optimal (with respect to the communication complexity) Byzantine consensus protocol, $\mathcal{ABC}$ produces an accountable Byzantine consensus protocol that is (1) optimal with respect to the communication complexity in solving consensus whenever consensus is solvable, and (2) optimal with respect to the accountability complexity in obtaining accountability whenever disagreement occurs.
4) We show that $\mathcal{ABC}$ is not limited to Byzantine consensus. Specifically, we define a class of *easily accountable agreement tasks* and we demonstrate that generalized $\mathcal{ABC}$ transformation provides accountability for such tasks. Important distributed problems, like Byzantine reliable [6] and Byzantine consistent [6] broadcast, fall into the class of easily accountable agreement tasks.

B. Related Work

The work on accountability in distributed systems was pioneered in [17]. The authors presented PeerReview, a generic accountability layer for distributed systems. Importantly, PeerReview does not allow correct processes to irrefutably detect faulty processes in non-synchronous environments, i.e., faulty processes might be *suspected forever*, but never irrefutably detected. Therefore, PeerReview does *not* suffice for accountability in non-synchronous Byzantine consensus protocols. Some of these authors initiated the formal study of Byzantine failures in the context of accountability [18].

Recently, with the expansion of blockchain systems, the interest in accountable distributed protocols resurfaced once again. Polygraph [9], the first accountable Byzantine consensus protocol, was introduced by Civit *et al.* The Polygraph protocol is based on the DBFT consensus protocol [11] used in blockchains [12], tolerates up to n faulty processes in achieving accountability and has the communication complexity of $O(n^4)$ in the common case, where n denotes the total number of processes. It is worth observing that Polygraph worsens the communication complexity of the (binary-value version) DBFT base protocol by an $O(n)$ multiplicative factor. Casper [4] is another system designed around the goal of obtaining accountability in blockchains, while Trap [22] combines accountability with game theory to increase the Byzantine fault tolerance of blockchains.

Most recently, authors of [23] investigated the possibility of obtaining accountability in protocols based on PBFT [7] in scenarios in which the system is not severely corrupted. Specifically, they present variants of PBFT [7] and HotStuff [25] that achieve accountability; however, they allow for accountability only if up to $2n/3$ processes are faulty, which implies that their "accountability threshold" is lower than the one of Polygraph. The commonality between the discussed prior work is employing sophisticated mechanisms for obtaining accountability. Indeed, the prior work achieves accountability with the help of non-trivial modifications applied to the base consensus protocol.

In contrast, we take a fundamentally different approach that allows us to treat the base consensus protocol as a "black box", thus obtaining simpler and more efficient accountable Byzantine consensus protocols. Table I compares accountable Byzantine consensus protocols obtained by $\mathcal{ABC}$ with the existing alternatives.[1]

Roadmap: We present the system model in §II. We devote §III to our $\mathcal{ABC}$ transformation. Specifically, we first intro-

[1]We purposefully omit the accountability complexity metric from Table I since we are the first to design accountable protocols around this metric.

Base Consensus Protocol	Communication Complexity of the Base Consensus Protocol	Communication Complexity of the Accountable Variant in the Common Case	Accountability Threshold	Paper
PBFT [7]	$O(n^4)$	$\Omega(n^4)$	$2n/3$	Sheng *et al.* [23]
HotStuff [25]	$O(n^3)$	$\Omega(n^3)$	$2n/3$	Sheng *et al.* [23]
Binary DBFT [11]	$O(n^3)$	$O(n^4)$	n	Civit *et al.* [9]
Multivalue DBFT [11]	$O(n^4)$	$O(n^4)$	n	Civit *et al.* [9]
Any	X	X	n	**this paper**

TABLE I: Overview of the main properties of existing accountable Byzantine consensus protocols. The authors of [23], when presenting accountable variants of PBFT and HotStuff, do not give details of incorporated view synchronization mechanisms; hence, communication complexities of these accountable protocols given in the table do not take into account the complexity of synchronization mechanisms.

duce the novel accountable confirmer problem (§III-A), the crucial building block of $\mathcal{ABC}$. Then, we present $\mathcal{ABC}$ and prove its correctness (§III-B). In §III-C, we demonstrate that $\mathcal{ABC}$ suffices for obtaining optimal accountability complexity in accountable Byzantine consensus protocols. In §IV, we discuss the applicability of $\mathcal{ABC}$ to various variants of Byzantine consensus protocols and propose an alternative definition of the accountability complexity metric. We define easily accountable agreement tasks and prove the applicability of generalized $\mathcal{ABC}$ to such tasks in §V. Finally, we conclude the paper in §VI.

II. Model

We consider a system with a set $\{P_1, ..., P_n\}$ of n processes that communicate by exchanging messages through a point-to-point network. The system is *non-synchronous*: there is no bound that always holds on message delays and relative speed of processes. Non-synchronous systems include (1) asynchronous systems, where the bound does not exist, and (2) partially synchronous systems [15], where the bound holds only eventually. All the results given in the present paper assume a non-synchronous system.

Each process is assigned its *local protocol*. A local protocol of a process defines steps to be taken during a run of the system. The collection of all local protocols assigned to processes is a *distributed protocol* (or simply a *protocol*).

A subset of all processes might be *faulty*: these processes may arbitrarily deviate from their local protocol, i.e., we consider the Byzantine failure model. If a process is not faulty, the process is *correct*. We assume that every message sent by a correct process to a correct process is eventually received, i.e., we assume that communication is *reliable*. Moreover, we assume that the order of message receptions is controlled by a computationally bounded adversary.

An *execution* is a single run of the system, i.e., it is a sequence of sending and receiving events, as well as the internal events of processes. Since disagreement can occur only if two (or more) correct processes exist, we do not consider executions with less than two correct processes. The actual number of faulty processes in an execution is denoted by t. Finally, the power set of a set X is denoted by $\mathbb{P}(X)$.

a) Cryptographic Primitives: We assume an idealized *public-key infrastructure* (PKI): each process is associated with its public/private key pair that is used to sign messages and verify signatures of other processes. A message m signed with the PKI private key of a process P_i is denoted by m_{σ_i}.

Additionally, we assume a (k, n)-threshold signature scheme [20], where $k = n - t_0$. In this scheme, each process holds a distinct private key, and there exists a single public key. Each process P_i can use its private key to produce a partial signature of a message m by invoking $\mathit{ShareSign}_i(m)$. Moreover, a partial signature $\mathit{tsignature}$ of a message m produced by process P_i could be verified by $\mathit{ShareVerify}_i(m, \mathit{tsignature})$. Finally, set $S = \{\mathit{tsignature}_i\}$ of partial signatures, where $|S| = k$ and, for each $\mathit{tsignature}_i \in S$, $\mathit{tsignature}_i = \mathit{ShareSign}_i(m)$, could be combined into a *single* digital signature by invoking $\mathit{Combine}(S)$; a combined digital signature $\mathit{tcombined}$ of message m could be verified by $\mathit{Verify}(m, \mathit{tcombined})$. In the paper, we assume that the cost of obtaining the threshold signature scheme [1] is amortized and, thus, negligible.

Crucially, we assume that the PKI private key of a correct process is *never* revealed (irrespectively of the number of faulty processes in the system). Therefore, if a message m is signed with the PKI private key of a process P_i and P_i is correct, then the message m was certainly sent by P_i. Conversely, if the number of faulty processes exceeds t_0, the threshold private key of a process *can* be revealed, and a partial signature of a correct process might be forged.

b) Proof of Culpability: A set Σ of messages properly signed with the PKI private key of a process P_i is *proof of culpability* of P_i if and only if there does not exist an execution of the system where (1) P_i sends all the messages from the Σ set, and (2) P_i is correct. Since the PKI private key of a correct process is never revealed (as opposed to the threshold private key of a correct process that might be revealed if the number of faults exceeds t_0), proof of culpability of a correct process can *never* be obtained.

c) Communication Complexity: In this work, as in many in distributed computing [2], [24], we care about the *communication complexity*. To this end, we define *authenticators* and *words*. An authenticator is either a partial signature or a signature. A word contains a constant number of authenticators and values, and each message contains at least one word. Finally, the communication complexity of a protocol is the maximum number of words sent in messages by correct processes across all possible executions.

d) Accountability Complexity: The *accountability complexity* is a novel complexity metric designed for measuring the accountability-specific performance of protocols. We define the accountability complexity since the communication complexity is not a suitable metric for measuring the performance of accountable Byzantine consensus protocols in the degraded case (i.e., when the number of faults exceeds t_0). Indeed, Polygraph and accountable variants of PBFT and Hotstuff, presented in [23], suffer from infinite communication complexity in the degraded case: Byzantine processes force correct processes to constantly execute "one more round", thus constructing an infinite execution where correct processes never decide.

Let us formally define the accountability complexity. Let abc be a protocol solving accountable Byzantine consensus. A message m is an *accountability-specific* message if there exists an execution in which a correct process P_i obtains proof of culpability Σ of a process P_j, where $m \in \Sigma$. Intuitively, a message is accountability-specific if it can be used as (a part of) proof of culpability of its sender.

Next, we define the accountability complexity of an execution e. Let a correct process P_i forward an accountability-specific message m, where P_i is not the original sender of m, in e. Then, this action counts as one unit towards the accountability complexity of e. Importantly, if P_i forwards $X > 1$ accountability-specific messages "together" (i.e., in the same message), the action counts as X units towards the complexity of the execution. The accountability complexity of abc is the maximum accountability complexity across all possible executions, i.e., across all executions with at least two correct processes. In a nutshell, the accountability complexity represents the number of messages correct processes exchange "thinking" that exactly those messages might be crucial for proving culpability of faulty processes.

III. $\mathcal{ABC}$ Transformation

This section presents $\mathcal{ABC}$, our transformation that enables any Byzantine consensus protocol to obtain accountability. We first introduce the *accountable confirmer* problem and give its implementation (§III-A). Then, we construct our $\mathcal{ABC}$ transformation around accountable confirmer (§III-B). Finally, §III-C proves that $\mathcal{ABC}$ suffices for achieving optimal accountability complexity.

A. Accountable Confirmer

The accountable confirmer problem is a distributed problem defined among n processes. The problem is associated with parameter $t_0 = \lceil n/3 \rceil - 1$ emphasizing that some properties are ensured only if the number of faulty processes does not exceed t_0.[2] The accountable confirmer exposes the following interface: (1) request $submit(v)$ - a process *submits* value v; invoked at most once; (2) indication $confirm(v')$ - a process *confirms* value v'; triggered at most once; and (3) indication $detect(F, proof)$ - a process *detects* processes from the set F such that $|F| \geq t_0 + 1$ and $proof$ represents proof of culpability of all processes that belong to F; triggered at most once. The following properties are ensured:

- *Terminating Convergence:* If the number of faulty processes does not exceed t_0 and all correct processes submit the same value, then that value is eventually confirmed by all correct processes.
- *Agreement:* If the number of faulty processes does not exceed t_0, then no two correct processes confirm different values.
- *Validity:* The value confirmed by a correct process was submitted by a correct process.
- *Accountability:* If two correct processes confirm different values, then every correct process eventually detects at least $t_0 + 1$ faulty processes and obtains proof of culpability of all detected processes.

Terminating convergence ensures that, if (1) the number of faults does not exceed t_0, and (2) all correct processes submit the same value, then all correct processes eventually confirm that value.[3] Agreement stipulates that no two correct processes confirm different values if the system is not corrupted (even if submitted values of correct processes differ). Validity ensures that any confirmed value is submitted by a correct process. Finally, accountability ensures detection of $t_0 + 1$ faulty processes by every correct process whenever correct processes confirm different values.

a) Intuition: We now give intuition behind our solution of the accountable confirmer problem. Once a correct process submits its value, it broadcasts a signed message containing the submitted value. Then, the process waits for $n - t_0$ messages containing the same value. Once this happens, the process (1) confirms the value, and (2) broadcasts the received $n - t_0$ messages to all processes in the system.

This simple algorithm ensures terminating convergence since, when there are up to t_0 faults and all correct processes submit the same value, all correct processes eventually receive $n - t_0$ messages containing the submitted value; thus, all correct processes confirm the value. As for the accountability property, if two correct processes disagree, every correct process eventually receives two conflicting sets of $n - t_0$ messages. Every process whose messages belong to both sets is faulty as no correct process submits multiple values.

b) Implementation: The implementation (Algorithm 2) of the accountable confirmer problem builds upon the presented intuition. It takes advantage of threshold signatures (see §II) in order to achieve quadratic communication complexity in the common case (i.e., in executions with up to t_0 faults).

Each process initially broadcasts the value it submitted in a SUBMIT message (line 18): the SUBMIT message contains the value and a partial signature of the value. Moreover, the entire message is signed with the PKI private key of the sender. Once a process receives such a SUBMIT message, the process (1) checks whether the message is properly

[2]Recall that t_0 is the number of faults tolerated in Byzantine consensus.

[3]Note that it is *not* guaranteed that any correct process confirms a value if correct processes submit different values (even if the number of faulty processes does not exceed t_0).

signed (line 6), (2) verifies the partial signature (line 20), and (3) checks whether the received value is equal to its submitted value (line 20). If all of these checks pass, the process stores the received partial signature (line 22) and the entire message (line 23). Once a process stores partial signatures from (at least) $n-t_0$ processes (line 25), the process confirms its submitted value (line 27) and informs other processes about its confirmation by combining the received partial signatures into a *light certificate* (line 28). The role of threshold signatures in our implementation is to allow every light certificate to contain a *single* signature (rather than $n - t_0 = O(n)$ signatures), thus obtaining quadratic overall communication complexity if $t \leq t_0$.

Once a process receives two conflicting light certificates (line 33), the process concludes that correct processes might have confirmed different values. If the process has already confirmed its value, the process broadcasts the set of (at least) $n-t_0$ properly signed $[\textsc{submit}, v, *]$ messages (line 34), where v is the value confirmed (and submitted) by the process; such a set of messages is a *full certificate* for value v. Finally, once a process receives two conflicting full certificates (line 39), the process obtains proof of culpability of (at least) $t_0 + 1$ faulty processes (line 42), which ensures accountability. Indeed, each full certificate contains $n-t_0$ properly signed messages: every process whose messages belong to the conflicting full certificates is faulty and these messages represent proof of its misbehavior. Recall that no faulty process *ever* obtains the PKI private key of a correct process.

Accountable Confirmer - Definitions for Algorithm 2

1) A combined digital signature $tsig$ is a *valid light certificate for value* v if and only if $Verify(v, tsig) = \top$.

2) A set $\mathcal{S}$ of properly signed $[\textsc{submit}, v, *]_{\sigma_*}$ messages is a *valid full certificate for value* v if and only if:
 a) $|\mathcal{S}| \geq n - t_0$
 b) Each message m is sent (i.e., signed) by a distinct process.

3) Let $tsig_v$ be a valid light certificate for value v and let $tsig_{v'}$ be a valid light certificate for value v'. $tsig_v$ *conflicts* with $tsig_{v'}$ if and only if $v \neq v'$.

4) Let $\mathcal{S}_v$ be a valid full certificate for value v and let $\mathcal{S}_{v'}$ be a valid full certificate for value v'. $\mathcal{S}_v$ *conflicts* with $\mathcal{S}_{v'}$ if and only if $v \neq v'$.

5) Let (m_1, m_2) be a pair of messages properly signed by the same process P_i. (m_1, m_2) is *proof of culpability of* P_i if and only if:
 a) $m_1 = [\textsc{submit}, v, share_1]_{\sigma_i}$
 b) $m_2 = [\textsc{submit}, v', share_2]_{\sigma_i}$
 c) $v \neq v'$.

Theorem 1. *Algorithm 2 solves the accountable confirmer problem with:*

- *$O(n^2)$ communication complexity in the common case, and*
- *at most $O(n^3)$ SUBMIT messages forwarded by correct processes.*

Proof. We start by proving the terminating convergence property. Indeed, if $t \leq t_0$ and all correct processes submit the same value v, then the rule at line 25 eventually triggers at every correct process. Since every correct process confirms only the value it has submitted (line 27), the property is satisfied by Algorithm 2.

We prove agreement by contradiction. Let a correct process P_i confirm a value v, let another correct process P_j confirm a value $v' \neq v$ and let $t \leq t_0$. Hence, P_i (resp., P_j) has received $n - t_0$ SUBMIT messages for value v (resp., v'). Given that $t_0 < n/3$, the number of processes that have sent the SUBMIT messages for both values must be greater than t_0. Therefore, there are more than t_0 faulty processes, which contradicts the fact that $t \leq t_0$. Thus, the agreement property is ensured.

Validity follows from the fact that each correct process confirms only the value it has submitted (line 27).

We now prove accountability. Let a correct process P_i confirm a value v and let another correct process P_j confirm a value $v' \neq v$. The rule at line 33 is eventually triggered at each correct process that confirms a value. Once the rule is triggered at P_i and P_j, these processes broadcast their full certificates to all processes (line 34). Eventually, the rule at line 39 is triggered at each correct process, which ensures accountability. Indeed, every process whose SUBMIT messages belong to both conflicting full certificates is detected; moreover, such a process is indeed faulty since no correct process submits different values, implying that no correct process ever sends different SUBMIT messages.

Finally, we prove the claimed complexity:

- If $t \leq t_0$, the communication complexity of the algorithm is quadratic because (1) light certificates are sent only once and they contain a single signature, and (2) no correct process broadcasts a full certificate.
- Each correct process forwards $O(n)$ SUBMIT messages in each FULL-CERTIFICATE message it sends (line 34). Therefore, each correct process forwards $O(n^2)$ SUBMIT messages, which implies that (at most) $O(n^3)$ SUBMIT messages are forwarded by all correct processes.

The theorem holds. □

B. $\mathcal{ABC}$: Byzantine Consensus + Accountable Confirmer = Accountable Byzantine Consensus

We now present our $\mathcal{ABC}$ transformation (Algorithm 3), the main contribution of our work. $\mathcal{ABC}$ is built on the observation that any Byzantine consensus protocol paired with accountable confirmer solves the accountable Byzantine consensus problem. Specifically, we prove that Algorithm 3 solves the accountable Byzantine consensus problem, which implies that $\mathcal{ABC}$ indeed enables Byzantine consensus protocols to obtain accountability.

Theorem 2. *Let bc be a Byzantine consensus protocol with the communication complexity X_{bc}. Let abc be a protocol obtained by applying $\mathcal{ABC}$ (Algorithm 3) to bc. Then, abc solves the accountable Byzantine consensus problem with (1) $max(X_{bc}, O(n^2))$ communication complexity in the common case, and (2) $O(n^3)$ accountability complexity.*

Algorithm 2 Accountable Confirmer - Code for Process P_i

```
1: Implements:
2:     Accountable Confirmer, instance $ac$
3: Uses:
4:     Best-Effort Broadcast [5], instance $beb$    ▷ Simple broadcast without any guarantees if the sender is faulty.
5: Rules:
6:     1) Any SUBMIT message that is not properly signed is discarded.
7:     2) Rules at lines 25, 33 and 39 are activated at most once.
8: upon event $\langle ac, Init \rangle$ do
9:     $value_i \leftarrow \bot$
10:    $confirmed_i \leftarrow false$
11:    $from_i \leftarrow \emptyset$
12:    $lightCertificate_i \leftarrow \emptyset$
13:    $fullCertificate_i \leftarrow \emptyset$
14:    $obtainedLightCertificates_i \leftarrow \emptyset$
15:    $obtainedFullCertificates_i \leftarrow \emptyset$
16: upon event $\langle ac, Submit \mid v \rangle$ do    ▷ $P_i$ submits a value
17:    $value_i \leftarrow v$
18:    trigger $\langle beb, Broadcast \mid [\text{SUBMIT}, v, ShareSign_i(v)]_{\sigma_i} \rangle$
19: upon event $\langle beb, Deliver \mid P_j, [\text{SUBMIT}, value, share]_{\sigma_j} \rangle$ do
20:    if $ShareVerify_j(value, share) = \top$ and $value = value_i$ and $P_j \notin from_i$ then
21:        $from_i \leftarrow from_i \cup \{P_j\}$
22:        $lightCertificate_i \leftarrow lightCertificate_i \cup \{share\}$
23:        $fullCertificate_i \leftarrow fullCertificate_i \cup \{[\text{SUBMIT}, value, share]_{\sigma_j}\}$
24:    end if
25: upon $|from_i| \geq n - t_0$ do
26:    $confirmed_i \leftarrow true$
27:    trigger $\langle ac, Confirm \mid value_i \rangle$    ▷ $P_i$ confirms a value
28:    trigger $\langle beb, Broadcast \mid [\text{LIGHT-CERTIFICATE}, value_i, Combine(lightCertificate_i)] \rangle$    ▷ Combine any $n - t_0$ partial signatures
29: upon event $\langle beb, Deliver \mid P_j, [\text{LIGHT-CERTIFICATE}, value_j, lightCertificate_j] \rangle$ do
30:    if $lightCertificate_j$ is a valid light certificate for $value_j$ then
31:        $obtainedLightCertificates_i \leftarrow obtainedLightCertificates_i \cup \{[\text{LIGHT-CERTIFICATE}, value_j, lightCertificate_j]\}$
32:    end if
33: upon $certificate_1, certificate_2 \in obtainedLightCertificates_i$ where $certificate_1$ conflicts with $certificate_2$
    and $confirmed_i = true$ do
34:    trigger $\langle beb, Broadcast \mid [\text{FULL-CERTIFICATE}, value_i, fullCertificate_i] \rangle$
35: upon event $\langle beb, Deliver \mid P_j, [\text{FULL-CERTIFICATE}, value_j, fullCertificate_j] \rangle$ do
36:    if $fullCertificate_j$ is a valid full certificate for $value_j$ then
37:        $obtainedFullCertificates_i \leftarrow obtainedFullCertificates_i \cup \{fullCertificate_j\}$
38:    end if
39: upon $certificate_1, certificate_2 \in obtainedFullCertificates_i$ where $certificate_1$ conflicts with $certificate_2$ do
40:    $proof \leftarrow$ extract proof of culpability of (at least) $t_0 + 1$ processes from $certificate_1$ and $certificate_2$
41:    $F \leftarrow$ the set of processes detected via $proof$
42:    trigger $\langle ac, Detect \mid F, proof \rangle$    ▷ $P_i$ detects faulty processes
```

Proof. Consider an execution where $t \leq t_0$. All correct processes eventually decide the same value v from Byzantine consensus at line 9 (by termination and agreement of Byzantine consensus). Moreover, if all correct processes have proposed the same value (line 7), then the proposed value is indeed v (ensured by validity of Byzantine consensus). Terminating convergence of accountable confirmer ensures that all correct processes eventually confirm v (line 11) and decide from accountable Byzantine consensus (line 12). Hence, Algorithm 3 satisfies termination, agreement and validity. Since the communication complexity of accountable confirmer in the common case is $O(n^2)$ (by Theorem 1), the communication complexity of abc in the common case is $max(X_{bc}, O(n^2))$.

If correct processes disagree (i.e., decide different values at line 12), then these processes have confirmed different values from accountable confirmer (line 11). Thus, accountability is ensured by Algorithm 3 since accountability is ensured by accountable confirmer, i.e., every correct process eventually detects faulty processes from accountable confirmer (line 13). Finally, all accountability-specific messages that are forwarded are the SUBMIT messages of accountable confirmer (see Algorithm 2). Given that (at most) $O(n^3)$ SUBMIT messages are forwarded by correct processes (by Theorem 1), the accountability complexity of Algorithm 3 is $O(n^3)$. □

Finally, we explicitly note that $\mathcal{ABC}$ does not *worsen* the communication complexity of any Byzantine consensus protocol. It is well-known that any protocol that solves the Byzantine consensus problem incurs quadratic communication complexity due to the lower bound set by Dolev *et*

Algorithm 3 $\mathcal{ABC}$ Transformation - Code For Process P_i

```
Implements:
    Accountable Byzantine Consensus, instance $abc$
Uses:
    ▷ Byzantine consensus protocol to be transformed
    Byzantine Consensus, instance $bc$
    Accountable Confirmer implemented by Algorithm 2,
    instance $ac$
upon event $\langle abc, Propose \mid proposal \rangle$ do                ▷ Proposal
    trigger $\langle bc, Propose \mid proposal \rangle$
upon event $\langle bc, Decide \mid decision \rangle$ do
    trigger $\langle ac, Submit \mid decision \rangle$
upon event $\langle ac, Confirm \mid confirmation \rangle$ do
    trigger $\langle abc, Decide \mid confirmation \rangle$               ▷ Decision
upon event $\langle ac, Detect \mid F, proof \rangle$ do
    trigger $\langle abc, Detect \mid F, proof \rangle$                   ▷ Detection
```

al. [14]. Therefore, every Byzantine consensus protocol *retains* its complexity after our transformation (by Theorem 2).

Corollary 1. Let bc be a Byzantine consensus protocol with the communication complexity X_{bc}. Let abc be a protocol obtained by applying $\mathcal{ABC}$ to bc. Then, abc solves the Byzantine consensus problem with the communication complexity X_{bc}.

C. $\mathcal{ABC}$ Suffices For Optimal Accountability

This subsection proves that any distributed protocol that solves the accountable Byzantine consensus problem incurs cubic accountability cost.

Let abc be a distributed protocol that solves the accountable Byzantine consensus problem among n processes. If up to $t_0 = \lceil n/3 \rceil - 1$ processes are faulty, abc ensures termination, agreement and validity; if disagreement occurs, each correct process eventually detects at least $t_0 + 1$ faulty processes (and obtains proof of culpability of all detected processes). Without loss of generality, let $n = 3t_0 + 1$.

The proof of the lower bound relies on the classical "partitioning" argument introduced in [15]. We start by separating processes that execute abc into three disjoint groups: (1) group A, where $|A| = t_0$, (2) group B, where $|B| = t_0 + 1$, and (3) group C, where $|C| = t_0$. Given that abc solves the accountable Byzantine consensus problem, the following two executions exist:

1) e_1: All processes from the group C are faulty and silent throughout the entire execution. Moreover, all processes from the $A \cup B$ set propose a value v. Since $|C| = t_0$, abc ensures that all processes from the $A \cup B$ set eventually decide the same value v (because of the validity property) by some global time t_1.
2) e_2: All processes from the group A are faulty and silent throughout the entire execution. Moreover, all processes from the $B \cup C$ set propose a value $v' \neq v$. Since $|A| = t_0$, abc ensures that all processes from the $B \cup C$ set eventually decide the value $v' \neq v$ (because of the validity property) by some global time t_2.

We can devise another execution e, where:

- Processes from the group A and processes from the group C are correct, whereas processes from the group B are faulty. Moreover, all processes from the group A propose v, and all processes from the group C propose $v' \neq v$.
- Processes from the group B behave towards processes from the group A as in execution e_1 and processes from the group B behave towards processes from the group C as in e_2. Moreover, if an event ϵ has occurred at global time t_ϵ in e_1 or e_2, then ϵ occurs at the same time t_ϵ in execution e.
- All messages between processes from groups A and C are delayed until time $max(t_1, t_2)$.

Execution e is indistinguishable from execution e_1 to processes from the group A, which implies that all processes from A decide value v by time t_1. Similarly, all processes from the group C decide value $v' \neq v$ by time t_2.

Finally, we denote by *partitioningExecution* the prefix of execution e up to time $T = max(t_1, t_2)$. Observe that the following holds for *partitioningExecution*:

- All processes from the group A decide v in *partitioningExecution*.
- All processes from the group C decide $v' \neq v$ in *partitioningExecution*.
- No message is exchanged between any two processes $(a \in A, c \in C)$.

We are now ready to prove the cubic lower bound on the accountability complexity.

Theorem 3. *The accountability complexity of abc is* $\Omega(n^3)$.

Proof. We build the proof upon *partitioningExecution*. Namely, *partitioningExecution* is convenient for proving the cubic lower bound since correct processes (i.e., processes from the $A \cup C$ set) cannot obtain proof of culpability of *any* process in *partitioningExecution*. Indeed, faulty processes (i.e., processes from the group B) appear correct in *partitioningExecution* to all processes from the $A \cup C$ set.

Our goal is to create an execution α as a continuation of *partitioningExecution* such that each correct process $c \in C$ forwards a quadratic number of accountability-specific messages in α. We fix a process $a \in A$. In α, only processes $\{a\} \cup C$ are correct; all other processes are faulty. Moreover, processes from the group B are silent after *partitioningExecution* (i.e., after time T) and messages between processes from the $\{a\} \cup C$ set are delayed.

Consider any process $c \in C$. There exists some time $T_1 > T$ by which c has forwarded $t_0 + 1 = O(n)$ accountability-specific messages to a process $a_1 \in A \setminus \{a\}$. Indeed, c cannot distinguish the current execution from the one in which (1) a_1 and c are the only correct processes, (2) they have disagreed, and (3) a_1 does not receive any message from any process s, where $s \neq c$ and $s \neq a_1$, in the continuation of *partitioningExecution* (i.e., after time T): as a_1 does not receive messages from processes other than a_1 and c after time T, a_1 is required to satisfy accountability, and a_1 cannot construct proof of culpability of any process given the messages received in *partitioningExecution* (i.e., by time T), process c must "help". However, process c cannot distinguish the execution until time T_1 from the one in which (1) a_1 is faulty, (2) only processes $a_2 \in A \setminus \{a, a_1\}$ and c are correct and they have disagreed, and (3) no process other than c "helps" a_2 to satisfy accountability. Thus, there exists some time $T_2 > T_1$ by which c has forwarded $O(n)$ accountability-specific messages to a_2 as well (besides forwarding $O(n)$ accountability-specific messages to a_1). As we are able to "replicate" the aforementioned logic for all processes from the $A \setminus \{a\}$ set, process c forwards $|A \setminus \{a\}|(t_0 + 1) = (t_0 - 1)(t_0 + 1) = O(n^2)$ accountability-specific messages in α.

Given that (1) all processes from the $A \setminus \{a\}$ set can behave towards each process $c \in C$ in the way explained above, and (2) messages among processes from the $\{a\} \cup C$ set are delayed for "sufficiently long", every process from the C set forwards $O(n^2)$ accountability-specific messages in α. Since $|C| = t_0 = O(n)$, the accountability complexity of α is $\Omega(n^3)$, which concludes the proof of the lower bound. $\square$

Given that $\mathcal{ABC}$ enables any Byzantine consensus protocol to obtain accountability with $O(n^3)$ accountability complexity (by Theorem 2), $\mathcal{ABC}$ suffices for optimal accountability with respect to the accountability complexity.

We conclude the subsection by stating the following result.

Corollary 2. Let bc_{opt} be a Byzantine consensus protocol with the optimal communication complexity X_{opt}, where $X_{opt} \geq O(n^2)$ (due to [14]). Let abc_{opt} be a protocol obtained by applying $\mathcal{ABC}$ to bc_{opt}. The following holds for abc_{opt}:

1) abc_{opt} solves the Byzantine consensus problem with the optimal communication complexity X_{opt}.
2) abc_{opt} obtains accountability with the optimal accountability complexity $O(n^3)$.

IV. Discussion

In this section, we first discuss the applicability of $\mathcal{ABC}$ to Byzantine consensus protocols satisfying different variants of the validity property (§IV-A). Secondly, we provide an alternative definition of the accountability complexity and briefly discuss it (§IV-B).

A. Different Variants of the Validity Property

The (accountable) Byzantine consensus problem (as defined in §I) specifies the validity property, which ensures that if all correct processes propose the same value, then only that value could be decided by a correct process. In the literature, there are many variants of the validity property; the one we use is traditionally called *strong validity*. Throughout the rest of this subsection, we refer to "our" validity property as strong validity. Other most notable variants of the validity property include:

- *Weak Validity:* If all processes are correct and if a correct process decides value v, then v is proposed by a (correct) process [3], [21], [25].
- *External Validity:* A value decided by a correct process satisfies a predefined $valid$ predicate [6].

Traditionally, the external validity property is accompanied by an additional validity property (e.g., [6]) to avoid trivial solutions in which all correct processes immediately decide a predefined valid value.

First and foremost, the correctness of $\mathcal{ABC}$ *does not* depend on a specific variant of the validity property. In other words, if a Byzantine consensus protocol bc satisfies weak (resp., external) validity, then abc, where abc is obtained by applying $\mathcal{ABC}$ to bc, is an accountable Byzantine consensus protocol satisfying weak (resp., external) validity.

However, does $\mathcal{ABC}$ preserve its ability to provide optimal accountability (with respect to the accountability complexity) if applied to a Byzantine consensus algorithm satisfying weak or external validity (rather than strong validity we considered throughout the paper)? In other words, is there an accountable Byzantine consensus protocol satisfying weak or external validity with the accountability complexity less than $O(n^3)$? Unfortunately, the answer is negative.

Namely, any accountable Byzantine consensus protocol satisfying weak validity has $\Omega(n^3)$ accountability complexity.[4] In order to prove this claim, it suffices to show that *partitioningExecution* (defined in §III-C) is possible as the argument presented in the proof of Theorem 3 can then be used to illustrate the cubic lower bound. To this end, we prove that executions e_1 and e_2 (used to "build" *partitioningExecution* in §III-C) are possible.

Theorem 4. *Let abc be an accountable Byzantine consensus protocol satisfying the weak validity property. Then, executions e_1 and e_2 (defined in §III-C) are possible.*

Proof. Without loss of generality, we prove that e_1 is a possible execution. By contradiction, suppose that e_1 execution is impossible. Therefore, processes from the $A \cup B$ set decide a value $v^* \neq v$ in the described execution. However, there exists another execution e_1' such that (1) all processes are correct, (2) all processes propose v, (3) all processes from the $A \cup B$ set observe the same environment as in e_1 until time t_1, and (4) communication between groups $A \cup B$ and C is delayed until t_1. Hence, executions e_1 and e_1' are indistinguishable to processes from $A \cup B$ until time t_1, which implies that a correct process $a \in A \cup B$ decides $v^* \neq v$ in e_1'. Thus, the weak validity property of abc is violated in e_1'. The contradiction is reached and the theorem holds. $\square$

Since executions e_1 and e_2 can be constructed (by Theorem 4), *partitioningExecution* is possible. Hence, the proof of Theorem 3 shows that abc, an accountable Byzantine consensus protocol satisfying weak validity, has $\Omega(n^3)$ accountability complexity. More generally, all accountable Byzantine

[4] As we have already mentioned, if external validity is not accompanied by another validity property, there exists a simple algorithm that "always" (i.e., irrespectively of the number of faults) solves Byzantine consensus. That is the reason we do not consider accountable Byzantine consensus protocols satisfying *solely* external validity.

consensus protocols that allow *partitioningExecution* to exist have $\Omega(n^3)$ accountability complexity.

B. Accountability Complexity: Alternative Definition

As already mentioned in §II, accountable Byzantine consensus protocols may have infinite communication complexity in executions with more than t_0 faulty processes. This is why the communication complexity metric is not suitable for such corrupted executions. However, we could measure the number of bits exchanged by correct processes *after* they have become aware that the ongoing execution is corrupted (i.e., that it contains more than t_0 faulty processes). Solving consensus in corrupted executions is not required by the definition of the problem (see §I). Therefore, all efforts towards solving consensus can be dropped upon a realization that the ongoing execution is corrupted, implying that only "accountability-related" work must be done hereafter.

Let us formally introduce the *awareness-based* accountability complexity, a possible alternative to the definition given in §II. A correct process P_i *becomes aware* that the ongoing execution e is *corrupted* if and only if no execution e' exists such that (1) P_i behaves in e' as it behaves in e, and (2) e' contains up to t_0 faulty processes. Intuitively, a correct process becomes aware that the current execution is "bad" (i.e., contains more than t_0 faulty processes) if no "good" (i.e., with up to t_0 faults) execution is possible given the process' ongoing behavior.

Next, we define the awareness-based accountability complexity of an execution e with $t_0 < t < n - 1$ (recall that we do not consider executions with more than $n - 2$ faults). Let $w_i(e)$ denote the number of words sent in messages by a correct process P_i in e *after* P_i has become aware that e is corrupted; if P_i never becomes aware that e is corrupted, then $w_i(e) = 0$. The awareness-based accountability complexity of e is the sum of $w_i(e)$, for every process P_i correct in e. Furthermore, the awareness-based accountability complexity of an accountable Byzantine consensus protocol is the maximum awareness-based accountability complexity across all executions with more than t_0 (and less than $n-1$) faulty processes. Lastly, the complexity of an accountable Byzantine consensus protocol abc could be seen as a pair $(cc(abc), ac(abc))$, where $cc(abc)$ denotes the communication complexity of abc in the common case (i.e., in executions with up to t_0 faulty processes) and $ac(abc)$ denotes the awareness-based accountability complexity of abc.

Our $\mathcal{ABC}$ transformation introduces cubic awareness-based accountability complexity as the communication complexity of accountable confirmer (see Algorithm 2) is $O(n^3)$ in the degraded case.[5] Specifically, if bc solves the Byzantine consensus problem with X_{bc} communication complexity, then the complexity of abc, where abc is obtained by applying $\mathcal{ABC}$ to bc, is $(X_{bc}, O(n^3))$.

[5]Our $\mathcal{ABC}$ transformation needs to be slightly modified to always ensure cubic awareness-based accountability complexity. Namely, once a correct process becomes aware that the ongoing execution is corrupted, it stops executing the original Byzantine consensus protocol and executes *solely* the accountable confirmer protocol.

V. Generalized $\mathcal{ABC}$ Transformation

We have shown that $\mathcal{ABC}$ enables Byzantine consensus protocols to obtain accountability. This section generalizes our $\mathcal{ABC}$ transformation and defines its applicability. Namely, we specify a class of distributed computing problems named *easily accountable agreement tasks* and we prove that generalized $\mathcal{ABC}$ enables accountability in such tasks.

We introduce agreement tasks in §V-A. Then, we define the class of easily accountable agreement tasks (§V-B) and prove the correctness of generalized $\mathcal{ABC}$ transformation applied to such agreement tasks (§V-C).

A. Agreement Tasks

Agreement tasks represent an abstraction of distributed input-output problems executed in a Byzantine environment. Specifically, each process has its *input value.* We assume that "$\perp$" denotes the special input value of a process that specifies that the input value is non-existent. A process may eventually halt; if a process halts, it produces its *output value.* The "$\perp$" output value of a process means that the process has not yet halted (and produced its output value). We denote by I_i (resp., O_i) the input (resp., output) value of process P_i. We note that some processes might never halt if permitted by the definition of an agreement task. We provide the formal explanation in the rest of the subsection.

An agreement task $\mathcal{A}$ is parameterized with the upper bound $t_{\mathcal{A}}$ on number of faulty processes that are tolerated. In other words, the specification of an agreement task assumes that no more than $t_{\mathcal{A}}$ processes are faulty in any execution.

Any agreement task could be defined as a relation between input and output values of processes. Since we assume that processes might fail, we only care about input and output values of correct processes. Hence, an agreement task could be defined as a relation between input and output values of *correct* processes.

An *input configuration* of an agreement task $\mathcal{A}$ is $\nu_I = \{(P_i, I_i) \text{ with } P_i \text{ is correct}\}$, where $|\nu_I| \geq n - t_{\mathcal{A}}$: an input configuration consists of input values of all correct processes. Similarly, an *output configuration* of an agreement task is $\nu_O = \{(P_i, O_i) \text{ with } P_i \text{ is correct}\}$, where $|\nu_O| \geq n - t_{\mathcal{A}}$: it contains output values of correct processes. We denote by $\theta(\nu_O) = |\{O_i \,|\, (P_i, O_i) \in \nu_O \wedge O_i \neq \perp\}|$ the number of distinct non-$\perp$ values in the ν_O output configuration.

Finally, we define an agreement task $\mathcal{A}$ as tuple $(\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$, where:

- $\mathcal{I}$ denotes the set of all input configurations of $\mathcal{A}$.
- $\mathcal{O}$ denotes the set of all output configurations of $\mathcal{A}$ such that $\theta(\nu_O) \leq 1$, for every $\nu_O \in \mathcal{O}$.
- $\Delta : \mathcal{I} \to 2^{\mathcal{O}}$, where $\nu_O \in \Delta(\nu_I)$ if and only if the output configuration $\nu_O \in \mathcal{O}$ is valid given the input configuration $\nu_I \in \mathcal{I}$.
- $t_{\mathcal{A}} \leq \lceil n/3 \rceil - 1$ denotes the maximum number of faulty processes the task assumes.

As seen from the definition, correct processes that halt always output the same value in agreement tasks. Moreover, we define agreement tasks to tolerate less than $n/3$ faults.

Without loss of generality, we assume that $\Delta(\nu_I) \neq \emptyset$, for every input configuration $\nu_I \in \mathcal{I}$. Moreover, for every $\nu_O \in \mathcal{O}$, there exists $\nu_I \in \mathcal{I}$ such that $\nu_O \in \Delta(\nu_I)$.

We note that some problems that are traditionally considered as "agreement" problems do not fall into our classification of agreement tasks. For instance, Byzantine lattice agreement [13] or k-set agreement [8] are *not* agreement tasks per our definition since the number of distinct non-$\perp$ values that can be outputted is greater than 1.

Solutions: We say that a distributed protocol $\Pi_{\mathcal{A}}$ *solves* an agreement task $\mathcal{A} = (\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$ if and only if, in every execution with up to $t_{\mathcal{A}}$ faults, there exists (an unknown) time T_D such that $\nu_O \in \Delta(\nu_I)$, where $\nu_I \in \mathcal{I}$ denotes the input configuration that consists of input values of all correct processes and $\nu_O \in \mathcal{O}$ denotes the output configuration that (1) consists of output values (potentially $\perp$) of all correct processes, and (2) no correct process P_i with $O_i = \perp$ updates its output value after T_D.

Finally, we say that a distributed protocol $\Pi_{\mathcal{A}}^{A}$ solves an agreement task $\mathcal{A} = (\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$ *with accountability* if and only if the following holds:

- *$\mathcal{A}$-Solution:* $\Pi_{\mathcal{A}}^{A}$ solves $\mathcal{A}$.
- *Accountability:* If two correct processes output different values, then every correct process eventually detects at least $t_{\mathcal{A}} + 1$ faulty processes and obtains proof of culpability of all detected processes.

B. Easily Accountable Agreement Tasks

Fix an agreement task $\mathcal{A} = (\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$. We say that $\mathcal{A}$ is an *easily accountable agreement task* if and only if one of the following conditions is satisfied:

1) *"All-or-None-Decidability":* There does not exist $\nu_O \in \mathcal{O}$ such that $(P_i, O_i \neq \perp) \in \nu_O$ and $(P_j, O_j = \perp) \in \nu_O$; or
2) *"Partial-Decidability":* For every $\nu_I \in \mathcal{I}$ such that there exists $\nu_O \in \Delta(\nu_I)$, where $(P_i, O_i = v \neq \perp) \in \nu_O$ and $(P_j, O_j = \perp) \in \nu_O$, the following holds:

$$\text{for every } c \in \mathbb{P}(\{P_i \mid (P_i, I_i) \in \nu_I\}), \exists \nu_O' \in \Delta(\nu_I),$$
$$\text{where } \forall P_i \in c : (P_i, O_i = v) \in \nu_O' \text{ and}$$
$$\forall P_j \in \{P_k \mid (P_k, I_k) \in \nu_I\} \setminus c : (P_j, O_j = \perp) \in \nu_O'.$$

"All-or-None-Decidability" characterizes all the problems in which either every process halts or none does. For instance, Byzantine consensus [19] and Byzantine reliable broadcast [5] satisfy "All-or-None-Decidability".

On the other hand, some agreement tasks permit that some processes halt, whereas others do not. We say that these tasks satisfy "Partial-Decidability" if and only if it is allowed for *any* subset of correct processes to halt (and output a value). Note that "Partial-Decidability" covers the case where no correct process ever halts. Byzantine consistent broadcast [5] is the only agreement task we are aware of that satisfies "Partial-Decidability" (in the case of a Byzantine sender). However, the significance of Byzantine consistent broadcast (e.g., for implementing cryptocurrencies [16]) motivated us to consider the "Partial-Decidability" property.

Algorithm 4 Generalized $\mathcal{ABC}$ Transformation - Code For Process P_i

```
 1: Implements:
 2:     Agreement Task $\mathcal{A}$ With Accountability,
        instance $a - \mathcal{A}$
 3: Uses:
 4:     ▷ Protocol to be transformed
 5:     Protocol that solves agreement task $\mathcal{A}$, instance $\Pi_{\mathcal{A}}$
 6:     Accountable Confirmer, instance $ac$
 7: upon event $\langle a - \mathcal{A}, Input \mid input \rangle$ do                ▷ Input
 8:     trigger $\langle \Pi_{\mathcal{A}}, Input \mid input \rangle$
 9: upon event $\langle \Pi_{\mathcal{A}}, Output \mid output \rangle$ do
10:     trigger $\langle ac, Submit \mid output \rangle$
11: upon event $\langle ac, Confirm \mid confirmation \rangle$ do
12:     trigger $\langle a - \mathcal{A}, Output \mid confirmation \rangle$          ▷ Output
13: upon event $\langle ac, Detect \mid F, proof \rangle$ do
14:     trigger $\langle a - \mathcal{A}, Detect \mid F, proof \rangle$            ▷ Detection
```

C. Correctness of Generalized $\mathcal{ABC}$ Transformation

We now prove the correctness of our generalized $\mathcal{ABC}$ transformation (Algorithm 4). First, we show that Algorithm 4 solves an easily accountable agreement task $\mathcal{A}$ if $\mathcal{A}$ satisfies "All-or-None-Decidability".

Lemma 1. Let $\mathcal{A} = (\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$ be an easily accountable agreement task that satisfies "All-or-None-Decidability". Algorithm 4 solves $\mathcal{A}$.

Proof. If no correct process ever outputs a value at line 9, then no correct process confirms any value from accountable confirmer (because no correct process submits any value to accountable confirmer at line 10). Hence, no correct process produces any output at line 12, which concludes the proof in this scenario.

Otherwise, each correct process eventually outputs a value at line 9. Moreover, all correct processes output the exact same value v (since $\mathcal{A}$ is an agreement task). Therefore, all correct processes submit the same value v to accountable confirmer (line 10). By terminating convergence of accountable confirmer, all correct processes eventually confirm value v (line 11) and output it (line 12). Once this happens, the agreement task $\mathcal{A}$ is solved, which concludes the lemma. □

Now, we prove that Algorithm 4 solves an easily accountable agreement task $\mathcal{A}$ if $\mathcal{A}$ satisfies "Partial-Decidability".

Lemma 2. Let $\mathcal{A} = (\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$ be an easily accountable agreement task that satisfies "Partial-Decidability". Algorithm 4 solves $\mathcal{A}$.

Proof. Let ν_I denote a specific input configuration of $\mathcal{A}$. We consider two cases:

- If no or all correct processes output a value at line 9, the proof is identical to the proof of Lemma 1.
- Otherwise, there exists a correct process that outputs a value v at line 9 and another correct process that does not output any value at line 9. Since $\mathcal{A}$ is an agreement task, any correct process that outputs a value at line 9

outputs the value v. Moreover, any correct process that outputs a value at line 12 outputs the value v (ensured by validity of accountable confirmer). Finally, once the system stabilizes at time T_D (the system stabilizes at time T_D if and only if no correct process P_i with $O_i = \perp$ updates its output value after T_D), the fact that any subset of correct processes could halt and that all halted processes output v implies that Algorithm 4 solves $\mathcal{A}$.

The lemma holds. □

Finally, we are ready to prove that Algorithm 4 solves $\mathcal{A}$ with accountability, where $\mathcal{A}$ is an easily accountable agreement task, which means that generalized $\mathcal{ABC}$ is correct.

Theorem 5. *Let $\mathcal{A} = (\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$ be an easily accountable agreement task. Algorithm 4 solves $\mathcal{A}$ with accountability.*

Proof. Algorithm 4 satisfies $\mathcal{A}$-solution by Lemmas 1 and 2. Furthermore, Algorithm 4 ensures accountability because of the fact that accountable confirmer ensures accountability and $t_{\mathcal{A}} \leq t_0$. Thus, the theorem holds. □

VI. Conclusion

We presented $\mathcal{ABC}$, a generic and simple transformation that allows Byzantine consensus protocols to obtain accountability. Besides its simplicity, $\mathcal{ABC}$ is efficient: it is sufficient to obtain an accountable Byzantine consensus protocol that is (1) optimal (with respect to the communication complexity) in solving consensus whenever consensus is solvable, and (2) optimal (with respect to the accountability complexity) in obtaining accountability whenever disagreement occurs. Finally, we show that $\mathcal{ABC}$ can easily be generalized to other agreement problems (e.g., Byzantine reliable broadcast, Byzantine consistent broadcast). Future work includes (1) designing similarly simple and efficient transformations for problems not covered by the generalized $\mathcal{ABC}$ transformation, like Byzantine lattice and k-set agreement problems, and (2) circumventing the cubic accountability complexity bound using randomization techniques.

Acknowledgments

This work is supported in part by the Australian Research Council Future Fellowship funding scheme (#180100496), and by Singapore MOE Grant MOE2018-T2-1-160.

References

[1] Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., and Tomescu, A. Reaching Consensus for Asynchronous Distributed Key Generation. In *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)* (2021), pp. 363–373.

[2] Abraham, I., Malkhi, D., and Spiegelman, A. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)* (2019), pp. 337–346.

[3] Buchman, E., Kwon, J., and Milosevic, Z. The latest gossip on BFT consensus. Tech. Rep. 1807.04938, arXiv, 2018.

[4] Buterin, V., and Griffith, V. Casper the Friendly Finality Gadget. *arXiv preprint arXiv:1710.09437* (2017).

[5] Cachin, C., Guerraoui, R., and Rodrigues, L. *Introduction to reliable and secure distributed programming.* Springer Science & Business Media, 2011.

[6] Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. Secure and Efficient Asynchronous Broadcast Protocols. In *Proceedings of the Annual International Cryptology Conference (CRYPTO)* (2001), Springer, pp. 524–541.

[7] Castro, M., and Liskov, B. Practical Byzantine Fault Tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI)* (1999), p. 173–186.

[8] Chaudhuri, S. More Choices Allow More Faults: Set Consensus Problems In Totally Asynchronous Systems. *Information and Computation 105*, 1 (1993), 132–158.

[9] Civit, P., Gilbert, S., and Gramoli, V. Brief Announcement: Polygraph: Accountable Byzantine Agreement. In *Proceedings of the 34th International Symposium on Distributed Computing (DISC)* (2020), vol. 179 of *LIPIcs*, pp. 45:1–45:3.

[10] Civit, P., Gilbert, S., and Gramoli, V. Polygraph: Accountable Byzantine agreement. In *Proceedings of the IEEE 41st International Conference on Distributed Computing Systems (ICDCS)* (2021), pp. 403–413.

[11] Crain, T., Gramoli, V., Larrea, M., and Raynal, M. DBFT: Efficient Leaderless Byzantine Consensus and its Application to Blockchains. In *Proceedings of the IEEE 17th International Symposium on Network Computing and Applications (NCA)* (2018), pp. 1–8.

[12] Crain, T., Natoli, C., and Gramoli, V. Red Belly: A Secure, Fair and Scalable Open Blockchain. In *Proceedings of the 42nd IEEE Symposium on Security and Privacy (SP)* (2021), pp. 466–483.

[13] de Souza, L. F., Kuznetsov, P., Rieutord, T., and Tucci Piergiovanni, S. Accountability and Reconfiguration: Self-Healing Lattice Agreement. In *Proceedings of the 25th International Conference on Principles of Distributed Systems (OPODIS)* (2021), pp. 25:1–25:23.

[14] Dolev, D., and Reischuk, R. Bounds on information exchange for Byzantine Agreement. *Journal of the ACM (JACM) 32*, 1 (1985), 191–204.

[15] Dwork, C., Lynch, N., and Stockmeyer, L. Consensus in the presence of partial synchrony. *Journal of the ACM (JACM) 35*, 2 (1988), 288–323.

[16] Guerraoui, R., Kuznetsov, P., Monti, M., Pavlovič, M., and Seredinschi, D.-A. The Consensus Number of a Cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)* (2019), pp. 307–316.

[17] Haeberlen, A., Kouznetsov, P., and Druschel, P. PeerReview: practical accountability for distributed systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP)* (2007), ACM, pp. 175–188.

[18] Haeberlen, A., and Kuznetsov, P. The Fault Detection Problem. In *Proceedings of the 13th International Conference on Principles of Distributed Systems (OPODIS)* (2009), Springer, pp. 99–114.

[19] Lamport, L., Shostak, R., and Pease, M. The Byzantine Generals Problem. In *Concurrency: the Works of Leslie Lamport.* 2019, pp. 203–226.

[20] Libert, B., Joye, M., and Yung, M. Born and raised distributively: fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theor. Comput. Sci. 645* (2016), 1–24.

[21] Milosevic, Z., Hutle, M., and Schiper, A. Unifying Byzantine Consensus Algorithms with Weak Interactive Consistency. In *International Conference On Principles Of Distributed Systems (OPODIS)* (2009), Springer, pp. 300–314.

[22] Pedrosa, A. R., and Gramoli, V. Trap: The Bait of Rational Players to Solve Byzantine Consensus. In *Proceedings of the 17th ACM Asia Conference on Computer and Communications Security (ASIACCS)* (2022).

[23] Sheng, P., Wang, G., Nayak, K., Kannan, S., and Viswanath, P. BFT Protocol Forensics. In *Computer and Communication Security (CCS)* (2021).

[24] Spiegelman, A. In Search for an Optimal Authenticated Byzantine Agreement. In *Proceedings of the 35th International Symposium on Distributed Computing (DISC)* (2021), pp. 38:1–38:19.

[25] Yin, M., Malkhi, D., Reiter, M. K., Golan-Gueta, G., and Abraham, I. HotStuff: BFT Consensus with Linearity and Responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing (PODC)* (2019), pp. 347–356.