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

PIERRE CIVIT, Sorbonne University, pierre.civit@lip6.fr
SETH GILBERT, NUS Singapore, seth.gilbert@comp.nus.edu.sg
VINCENT GRAMOLI, University of Sydney and EPFL, vincent.gramoli@sydney.edu.au
RACHID GUERRAOUI, EPFL, rachid.guerraoui@epfl.ch
JOVAN KOMATOVIC, EPFL, jovan.komatovic@epfl.ch

It is known that the Byzantine consensus problem among n processes cannot be solved in a non-synchronous system if the number of faulty processes exceeds t_0, where $t_0 = \lceil n/3 \rceil - 1$ [15]. Indeed, if the number of faulty processes is greater than the t_0 threshold, correct processes might never decide or (even worse) correct processes might decide and disagree. We focus in this paper on the latter case, where disagreement occurs. Specifically, we investigate the *accountable* Byzantine consensus problem in non-synchronous systems: the problem of solving Byzantine consensus whenever possible (i.e., whenever the number of faulty processes does not exceed the t_0 bound) and otherwise allowing correct processes to obtain a proof of culpability of (at least) $t_0 + 1$ faulty processes whenever correct processes disagree. We present three complementary contributions:

(i) We give a simple transformation named $\mathcal{ABC}$ that enables any Byzantine consensus protocol to obtain accountability. Besides being simple, $\mathcal{ABC}$ is also efficient: it induces an overhead of (1) two all-to-all communication rounds and $O(n^2)$ exchanged bits of information in all executions with up to t_0 faults[1], and (2) three all-to-all communication rounds and $O(n^3)$ exchanged bits of information otherwise. Therefore, any protocol that solves the Byzantine consensus problem with quadratic (or greater) communication complexity retains its complexity in solving the problem after our transformation.

(ii) We show that $\mathcal{ABC}$, despite its simplicity, allows for *optimal* communication complexity in solving the accountable Byzantine consensus problem. That is, (1) we prove that any accountable Byzantine consensus incurs cubic communication complexity whenever disagreement occurs, and (2) we demonstrate that the lower bound is tight by applying $\mathcal{ABC}$ to any cubic Byzantine consensus protocol (e.g., binary DBFT [8]).

(iii) We show that $\mathcal{ABC}$ is not limited to the Byzantine consensus problem. Specifically, we define a class of *easily accountable agreement tasks* and we prove that generalized $\mathcal{ABC}$ transformation indeed provides accountability for such tasks. Important distributed tasks, like Byzantine reliable [3] and Byzantine consistent broadcast [3], fall into this class.

1 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 [15]. This limitation motivated researchers to investigate *accountable* variants of these problems. 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 behavior. If 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 of the system.

This paper primarily focuses on obtaining accountability in Byzantine consensus protocols that operate in non-synchronous systems. The Byzantine consensus problem [15] is defined among n processes while tolerating up to

[1]Since our transformation relies on the existence of a threshold signature scheme, we assume that the cost of obtaining such scheme is amortized and, thus, negligible.

$t_0 = \lceil n/3 \rceil - 1$ Byzantine (malicious) processes. A process initially *proposes* a value and eventually *decides* a value such that the following properties hold:

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

The conjunction of the aforementioned properties can only be ensured if the number of faulty processes does not exceed t_0 [15]. If indeed the faulty processes overpopulate the system, any of these properties might be violated. This work focuses on cases when 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 done 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 every correct process eventually decides.
- *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, a correct process can decide only that value.
- *Accountability:* If two correct processes decide different values, then every correct process eventually detects at least $t_0 + 1$ faulty processes and obtains a proof of culpability of all detected processes.

The contributions of the paper are threefold:

(i) We present a generic and simple transformation - $\mathcal{ABC}$ - that allows *any* Byzantine consensus protocol to obtain accountability. Our transformation is also efficient: it introduces an overhead of (1) only two all-to-all communication rounds and $O(n^2)$ exchanged bits of information in all executions with at most t_0 faulty processes (i.e., in the common case), and (2) three all-to-all communication rounds and $O(n^3)$ exchanged bits of information otherwise (i.e., in the degraded case). Therefore, any protocol that solves the Byzantine consensus problem with quadratic (or greater) communication complexity retains its complexity in solving the problem after $\mathcal{ABC}$.

$\mathcal{ABC}$ 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 processes eventually broadcast and receive $n - t_0$ matching CONFIRM messages. However, the important mechanism illustrated in Algorithm 1 is that faulty processes *must* send conflicting CONFIRM messages in order to cause disagreement. Hence, whenever correct processes disagree, they only need to exchange received CONFIRM messages to obtain accountability.

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

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

(ii) We show that our $\mathcal{ABC}$ transformation, despite its simplicity, suffices for obtaining *optimal* communication complexity in accountable Byzantine consensus protocols. Namely, we prove that any accountable Byzantine consensus incurs cubic communication complexity whenever disagreement occurs and we demonstrate that the lower bound is tight by applying $\mathcal{ABC}$ to any cubic Byzantine consensus protocol (e.g., DBFT binary consensus [8]).

(iii) We prove 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 indeed provides accountability for such tasks. Important distributed tasks, like Byzantine reliable [4] and Byzantine consistent [4] broadcast, fall into the class of easily accountable agreement tasks.

Related Work. The work on accountability in distributed systems was pioneered in [13]. The authors presented PeerReview - a generic accountability layer for distributed systems. Same authors initiated the formal study of Byzantine failures in the context of accountability [14]. Recently, with the expansion of blockchain systems, the interest in accountable distributed protocols resurfaced once again. Polygraph [7] - the first accountable Byzantine consensus protocol - was introduced by Civit *et al.* The Polygraph protocol solves Byzantine consensus [15] whenever possible (i.e., whenever the number of faulty processes is less than one third of all processes in the system) and enables accountability whenever two correct processes disagree. Casper [2] is another system designed around the goal of obtaining accountability. Most recently, authors of [18] investigated the possibility of obtaining accountability in protocols based on PBFT [5] in scenarios in which the system is not severely corrupted. The commonality between the discussed prior work is employing sophisticated mechanisms for obtaining accountability. In contrast, we give significantly simpler solution to the problem.

Roadmap. We present the system model in §2. We devote §3 to our $\mathcal{ABC}$ transformation. Specifically, we first introduce the novel accountable confirmer problem (§3.1), the crucial building block of $\mathcal{ABC}$. Then, we present $\mathcal{ABC}$ and prove its correctness (§3.2). In §3.3, we demonstrate that $\mathcal{ABC}$ suffices for obtaining optimal communication complexity in accountable Byzantine consensus protocols. We define easily accountable agreement tasks and prove the applicability of generalized $\mathcal{ABC}$ to such tasks in §4. Finally, we conclude the paper in §5.

2 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:

- asynchronous systems, where the bound does not exist, and
- partially synchronous systems [11], where the bound holds only eventually.

All our results given in the present paper assume a non-synchronous system.

Each process is assigned its *local protocol* to execute. A local protocol of a process defines steps to be taken by the process during a run of the system. The collection of all local protocols assigned to processes is referred to as 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, we say that the process is *correct*. We assume that any message sent by a correct process to a correct process is eventually received, i.e., we assume that communication is *reliable*. An *execution* of the system 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. We denote by t the actual number of faulty processes in an execution. Finally, we denote by $\mathbb{P}(X)$ the power set of a set X.

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

Moreover, we assume a (k, n)-threshold signature scheme [16], where $k = n - \lceil n/3 \rceil + 1$. 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 $ShareSign_i(m)$. Moreover, a partial signature *tsignature* of a message m produced by process P_i could be verified with $ShareVerify_i(m, tsignature)$. Finally, set $S = \{tsignature_i\}$ of partial signatures, where $|S| = k$ and, for each $tsignature_i \in S$, $tsignature_i = ShareSign_i(m)$, could be combined into a *single* digital signature by invoking $Combine(S)$; a combined digital signature *tcombined* of message m could be verified with $Verify(m, tcombined)$. In the rest of the paper, we assume that the cost of obtaining the threshold signature scheme 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. On the other hand, if the number of faulty processes does exceed $n - k$, then the threshold private key of a process *can* be revealed and faulty processes might forge a partial signature of a correct process.

Proof of Culpability. We say that a set $\mathcal{S}$ of properly authenticated messages sent by a process P_i is a *proof of culpability* of P_i if and only if there does not exist an execution e of the system where (1) P_i sends all the messages from the $\mathcal{S}$ set, and (2) P_i is correct. Observe that a proof of culpability of a process contains messages signed by the process with its PKI private key. Indeed, 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 $n - k$, where $k = n - \lceil n/3 \rceil + 1$) which implies that a proof of culpability of a correct process can *never* be obtained.

Complexity Measure. In this work, as in many in distributed computing, we care about the *communication complexity* which is the maximum number of authenticators sent by all correct processes combined across all executions of the system. An *authenticator* is either a partial signature or a signature.

3 $\mathcal{ABC}$ TRANSFORMATION

This section presents $\mathcal{ABC}$, our transformation that enables any Byzantine consensus protocol to obtain accountability. To this end, we first introduce the *accountable confirmer* problem and give its implementation (§3.1). Then, we construct our $\mathcal{ABC}$ transformation around accountable confirmer (§3.2). In §3.3, we prove that $\mathcal{ABC}$ allows for obtaining optimal communication complexity in accountable Byzantine consensus protocols. Finally, we conclude the section with a brief discussion about the applicability of $\mathcal{ABC}$ and communication optimality it provides (§3.4).

3.1 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]. Accountable confirmer exposes the following interface: (1) request *submit*(v) - a process *submits* value v, (2) indication *confirm*(v') - a process *confirms* value v', and (3) indication *detect*$(F, proof)$ - a process *detects*

[2]Recall that $t_0 = \lceil n/3 \rceil - 1$ is the number of faulty processes tolerated by the Byzantine consensus problem.

processes from the set F such that *proof* represents a proof of culpability of all processes that belong to F. 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 every correct process.
- *Agreement:* If the number of faulty processes does not exceed t_0, then no two correct processes confirm different values.
- *Validity:* Value confirmed by a correct process is 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 a 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.

Implementation. We now give an implementation of the accountable confirmer problem (Algorithm 2). The implementation takes advantage of threshold signatures (see §2) in order to obtain quadratic communication complexity in the common case (i.e., in executions with up to t_0 faulty processes). In the degraded case (i.e., in executions with more than t_0 faulty processes), the communication complexity is cubic.

Each process initially broadcasts the value it submitted in a SUBMIT message (line 17): the SUBMIT message contains the value and the 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 signed (line 6), (2) verifies the partial signature (line 19), and (3) checks whether the received value is equal to its submitted value (line 19). If all of these checks pass, the process stores the received partial signature (line 20) and the entire message (line 21). Once a process stores partial signatures from (at least) $n - t_0$ processes (line 23), the process confirms its submitted value (line 25) and informs other processes about its confirmation by combining the received partial signatures into a *light certificate* (line 26). The role of threshold signatures in our implementation is to allow a light certificate to contain a *single* signature, thus obtaining quadratic overall communication complexity if $t \leq t_0$.

Once a process receives two conflicting light certificates (line 31), the process concludes that correct processes might have indeed confirmed different values[4]. If the process has already confirmed its value, the process broadcasts the set of (at least) $n - t_0$ properly authenticated $[\text{SUBMIT}, v, *]$ messages (line 33), where v is the value confirmed (and submitted) by the process; such set of messages is a *full certificate* for value v. Finally, once a process receives two conflicting full certificates (line 38), the process obtains a proof of culpability of (at least) $t_0 + 1$ faulty processes (line 41), which ensures accountability. Indeed, each full certificate contains $n - t_0$ properly authenticated messages: every process whose message is in both full certificates is faulty and these messages represent a proof of its misbehavior (recall that no faulty process *ever* obtains the PKI private key of a correct process).

[3]Note that it is *not* guaranteed that any correct process eventually confirms a value if correct processes submit different values (even if the number of faulty processes does not exceed t_0).
[4]Note that the process is not certain that correct processes have confirmed different values because light certificates could be sent by faulty processes (possible only if $t > t_0$).

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 authenticated $[\textsc{submit}, v, *]_{\sigma_*}$ 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 properly authenticated messages sent by the same process P_i. (m_1, m_2) is a *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*
- $O(n^3)$ *communication complexity in the degraded case.*

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 23 eventually triggers at each correct process. Since each correct process confirms only the value it has submitted (line 25), the property is satisfied by Algorithm 2.

We prove agreement by contradiction. Let a correct process P_i confirm value v, let another correct process P_j confirm 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 $t_0 < n/3$, we conclude that number of processes that have sent the submit message for both values must be greater than t_0. This implies that there are more than t_0 faulty processes, which contradicts the fact that $t \leq t_0$. Therefore, the agreement property is ensured.

Validity trivially follows from the fact that each correct process confirms only the value it has submitted (line 25).

We now prove accountability. Let a correct process P_i confirm value v and let another correct process P_j confirm value $v' \neq v$. The rule at lines 31 and 32 is eventually triggered at each correct process that confirms a value. Once the rule is triggered at P_i (resp., P_j), the process broadcasts its full certificate to all processes (line 33). Eventually, the rule at line 38 is triggered at each correct process, which ensures accountability.

Finally, we prove the claimed communication 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 ever sends its full certificate.
- If $t > t_0$, the communication complexity is cubic. Indeed, broadcasting of a full certificate (that contains $O(n)$ authenticators) dominates the communication complexity in this case. Therefore, each correct process sends $O(n)$ authenticators to all processes (line 33), which results in cubic overall communication complexity.

The proof of the communication complexity concludes the theorem. □

Algorithm 2 Accountable Confirmer - Code for Process P_i

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

3.2 $\mathcal{ABC}$: Byzantine Consensus + Accountable Confirmer = Accountable Byzantine Consensus

We now define 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.

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

```
Implements:
    Accountable Byzantine Consensus, instance abc
Uses:
    Byzantine Consensus, instance bc                ▷ Byzantine consensus protocol to be transformed
    Accountable Confirmer, instance ac
upon event ⟨abc, Propose | proposal⟩ do
    trigger ⟨bc, Propose | proposal⟩
upon event ⟨bc, Decide | decision⟩ do
    trigger ⟨ac, Submit | decision⟩
upon event ⟨ac, Confirm | confirmation⟩ do
    trigger ⟨abc, Decide | confirmation⟩
upon event ⟨ac, Detect | F, proof⟩ do
    trigger ⟨abc, Detect | F, proof⟩
```

The following theorem shows that Algorithm 3 solves the accountable Byzantine consensus problem, which implies that $\mathcal{ABC}$ indeed allows Byzantine consensus protocols to obtain accountability.

Theorem 2. *Algorithm 3 solves the accountable Byzantine consensus problem.*

Proof. Consider an execution where $t \leq t_0$. All correct processes eventually decide the same value v from Byzantine consensus at line 8 (by termination and agreement of Byzantine consensus). Moreover, if all correct processes have proposed the same value (line 6), 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 10) and decide from the accountable Byzantine consensus (line 11). Hence, Algorithm 3 satisfies termination, agreement and validity if $t \leq t_0$.

If correct processes disagree (i.e., decide different values at line 11), then these processes have confirmed different values from accountable confirmer (line 10). 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 12). Thus, accountability is satisfied by Algorithm 3, which concludes the theorem. □

Finally, we note that $\mathcal{ABC}$ does not worsen the communication complexity of a Byzantine consensus protocol that solves Byzantine consensus optimally. 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 al.* [10]. Given the fact that accountable confirmer has quadratic communication complexity in the common case (Theorem 1), any Byzantine consensus protocol with quadratic communication complexity *retains* its complexity after our transformation. In other words, any optimal Byzantine consensus protocol still matches the Dolev-Reischuk lower bound after our transformation[5].

Corollary 1. $\mathcal{ABC}$ applied to any quadratic Byzantine consensus protocol gives an accountable Byzantine consensus protocol with quadratic communication complexity in the common case.

[5]We emphasize that the proof of quadratic lower bound presented in [10] *does not* consider cryptographic primitives like threshold signatures. Therefore, even though our transformation preserves quadratic communication complexity in all executions with up to t_0 faults, it does so by utilizing threshold signatures and, as such, is not covered by the proof given in [10].

3.3 $\mathcal{ABC}$ Suffices For Optimal Accountability

This subsection proves that any distributed protocol that solves the accountable Byzantine consensus problem incurs cubic communication cost. Moreover, we show that the lower bound is tight by applying $\mathcal{ABC}$ (§3.2) to any cubic (or sub-cubic) Byzantine consensus protocol (Corollary 2). Therefore, we show that our simple transformation allows Byzantine consensus protocols to obtain accountability *optimally* with respect to the communication complexity.

Let Π^A 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, Π^A ensures termination, agreement and validity; if disagreement occurs, each correct process eventually detects at least $t_0 + 1$ faulty processes (and obtains a proof of culpability of all detected processes). Without loss of generality, we assume that $n = 3t_0 + 1$.

We start by separating processes that execute Π^A 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 Π^A solves the Byzantine consensus problem, the following two executions exist:

- e_1: All processes from group C are faulty and silent throughout the entire execution. Moreover, all processes from the $A \cup B$ set propose value v. Since $|C| = t_0$, Π^A 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.
- e_2: All processes from group A are faulty and silent throughout the entire execution. Moreover, all processes from the $B \cup C$ set propose value $v' \neq v$. Since $|A| = t_0$, Π^A 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.

Now, we can devise another execution e in the following manner:

- Processes from group A and processes from group C are correct, whereas processes from group B are faulty. Moreover, all processes from group A propose v, whereas all processes from group C propose $v' \neq v$.
- Processes from group B behave towards processes from group A as in execution e_1 and processes from group B behave towards processes from group C as in execution e_2.
- All messages between processes from group A and group C are delayed until time $max(t_1, t_2)$.

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

Finally, we denote by *partitioningExecution* the prefix of execution e up to time $max(t_1, t_2)$ (Part (a) of Figure 1 depicts *partitioningExecution*). Observe that the following holds for execution *partitioningExecution*:

- All processes from group A decide v in *partitioningExecution*.
- All processes from 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 communication complexity for solving the accountable Byzantine consensus protocol.

Theorem 3. *The communication complexity of* Π^A *is* $\Omega(n^3)$.

Proof. The proof is built on top of *partitioningExecution* we constructed above. Namely, *partitioningExecution* is convenient for proving the cubic lower bound since the only way for correct processes (i.e., processes from the $A \cup C$ set) to ensure accountability is to exchange information among themselves. Indeed, faulty processes (i.e., processes from group B) appear correct to all processes from group A (resp., group C). Therefore, no faulty process is detected in *partitioningExecution* because of the fact that no communication is established between groups A and C.

Recall that each correct process needs to obtain a proof of culpability of (at least) $t_0 + 1 = O(n)$ faulty processes. If processes $a \in A$ and $c \in C$ aim to collaboratively obtain a proof of culpability of $t_0 + 1$ processes, both a and c need to send (at least) $t_0 + 1 = \Omega(n)$ authenticators. Moreover, a proof of culpability of $t_0 + 1$ processes must contain (at least) $\Omega(n)$ authenticators.

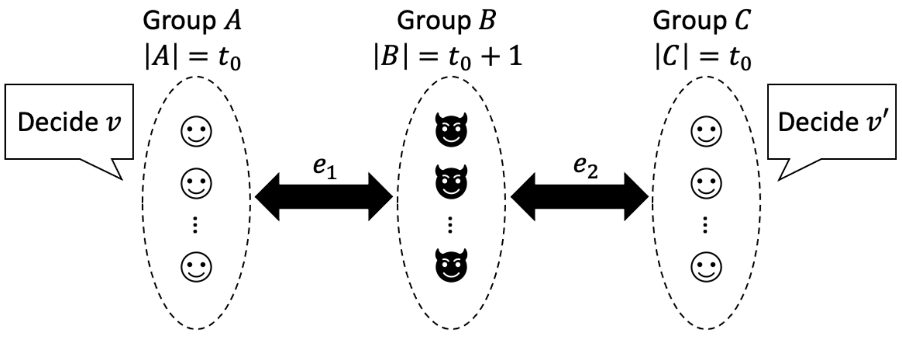
We now devise a continuation of *partitioningExecution* which ensures that correct processes do send $\Omega(n^3)$ authenticators. We start by stating that there is a single correct process in group A - we denote this process by a. Other processes from group A are Byzantine and they do not send any message to a in the continuation of *partitioningExecution*. All processes from group C are correct and all processes from group B are faulty and silent. Finally, all messages sent between processes from group C are delayed in the continuation of *partitioningExecution*.

Let $c_1 \in C$ be a process from group C; recall that c_1 is correct. In the continuation of *partitioningExecution*, c_1 eventually obtains a proof of culpability Σ of $t_0 + 1$ processes by communicating with a *single* process $a_1 \in A$. Specifically, no message is received by c_1 from any process that belongs to group A before c_1 obtains Σ by communicating with a_1. Importantly, c_1 cannot distinguish the current execution from one in which only correct processes are c_1 and a_1 (recall that accountability must be ensured even in the scenario with only two correct processes). However, after the communication with a_1, process c_1 cannot distinguish the current execution from one in which (1) a_1 is faulty (and just behaves correctly towards c_1), and (2) there exist other processes from group A that are correct, disagree with c_1 and need to detect faulty processes. Therefore, c_1 needs to communicate with other processes from group A. The aforementioned construction of the continuation of *partitioningExecution* is repeated for all processes $a_i \in A$: (1) process c_1 sends $\Omega(n)$ authenticators in order to allow process $a_i \in A$ to obtain Σ, and (2) before process a_i obtains Σ, process c_1 does not hear from any other process from group A from which it has not heard yet (i.e., process c_1 communicates with processes from group A in "one-by-one" fashion). Recall that process c_1 does not hear from any process from group C until c_1 has "helped" each process from group A to obtain Σ (i.e., all processes from group C might be faulty as seen from the perspective of c_1). Finally, we conclude that c_1 communicates quadratic number of authenticators in the execution ($\Omega(n)$ authenticators per $t_0 = O(n)$ processes).

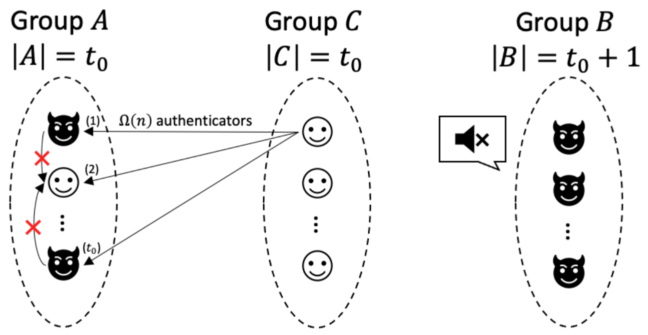
We apply the same reasoning for correct process $c_2 \in C$. First, process c_2 hears from any process from group A only after c_1 has already ensured that all processes from group A obtain Σ. All $t_0 - 1$ faulty processes from group A behave towards c_2 as if they receive the information from group C for the first time. Moreover, c_2 communicates with all processes from group A in "one-by-one" fashion. Note that a, the only correct process from group A, might not behave towards c_2 as if it hears the information from group C for the first time. However, process c_2 cannot be certain that a is indeed correct; the reason is that a has previously received information only from other processes from group C in the continuation of *partitioningExecution* and it is possible that all of these processes are faulty (as seen from the perspective of c_2; recall that c_2 has not heard from other processes from group C thus far), which implies that a might be faulty. Hence, c_2 also needs to ensure that each process from group A obtains a proof of culpability, which results in the fact that c_2 also sends quadratic number of authenticators.

Finally, the construction mechanism we presented for c_2 is repeated for all other processes from group C. Therefore, each process from group C sends quadratic number of authenticators. Since $|C| = t_0 = O(n)$, the total communication complexity of Π^A is $\Omega(n^3)$ (Part (b) of Figure 1 provides a visual depiction of the execution considered in the proof). □

The consequence of theorems 1 and 3 is that $\mathcal{ABC}$ allows any cubic (or sub-cubic) Byzantine consensus protocol (e.g., DBFT binary consensus [8]) to solve the accountable Byzantine consensus problem optimally.



(a) The *partitioningExecution* execution in which processes from groups A and C disagree.



(b) The continuation of *partitioningExecution* that incurs cubic number of authenticators sent by correct processes (the communication pattern of a single process from group C is depicted).

Fig. 1. Illustration of Theorem 3.

Corollary 2. $\mathcal{ABC}$ applied to any cubic (or sub-cubic) Byzantine consensus protocol gives an accountable Byzantine consensus protocol with the asymptotically optimal cubic communication complexity.

Direct consequence of corollaries 1 and 2 is that $\mathcal{ABC}$, when applied to any quadratic Byzantine consensus, obtains a protocol that:

- solves the Byzantine consensus problem with quadratic communication complexity, which matches the lower bound given by Dolev *et al.* [10], and
- solves the accountable Byzantine consensus problem with cubic communication complexity, which matches the lower bound given in the paper (Theorem 3).

3.4 Discussion

The (accountable) Byzantine consensus problem (as defined in §1) 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 [1, 17, 19].
- *External Validity:* A value decided by a correct process satisfies the predefined *valid* predicate [4].

Importantly, the correctness of $\mathcal{ABC}$ *does not* depend on the specific variant of the validity property.

However, the specific variant of the considered validity property plays a role in showing that our transformation allows for optimal solution to the accountable Byzantine consensus problem. As seen in §3.3, our proof of the cubic lower bound relies on the possibility of devising *partitioningExecution*. Indeed, *partitioningExecution* could be obtained as a consequence of the strong validity property (see §3.3). Still, if one assumes weak or external validity, there is no guarantee that such execution exist. Thus, the lower bound presented in §3.3 *does not* apply to Byzantine consensus problems that does not ensure strong validity, but some other variant of the property.

4 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 §4.1. Then, we define the class of easily accountable agreement tasks (§4.2) and prove the correctness of generalized $\mathcal{ABC}$ transformation applied to such agreement tasks (§4.3).

4.1 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 "$\bot$" 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 "$\bot$" 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 $v_I = \{(P_i, I_i) \text{ with } P_i \text{ is correct}\}$, where $|v_I| \geq n - t_{\mathcal{A}}$: an input configuration consists of input values of (all and exclusively) correct processes. Similarly, an *output configuration* of an agreement task is denoted by $v_O = \{(P_i, O_i) \text{ with } P_i \text{ is correct}\}$, where $|v_O| \geq n - t_{\mathcal{A}}$: it contains output values of correct processes. We denote by $\theta(v_O) = |\{O_i \mid (P_i, O_i) \in v_O \wedge O_i \neq \bot\}|$ the number of distinct non-$\bot$ values in the v_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 possible input configurations of $\mathcal{A}$.
- $\mathcal{O}$ denotes the set of all possible output configurations of $\mathcal{A}$ such that, for every $v_O \in \mathcal{O}$, $\theta(v_O) \leq 1$.
- $\Delta : \mathcal{I} \rightarrow 2^{\mathcal{O}}$, where $v_O \in \Delta(v_I)$ if and only if the output configuration $v_O \in \mathcal{O}$ is valid given the input configuration $v_I \in \mathcal{I}$.
- $t_{\mathcal{A}} \leq \lceil n/3 \rceil - 1$ 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 $\Delta(v_I) \neq \emptyset$, for every input configuration $v_I \in \mathcal{I}$. Moreover, for every $v_O \in \mathcal{O}$, there exists $v_I \in \mathcal{I}$ such that $v_O \in \Delta(v_I)$.

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 [9] or k-set agreement [6] 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 there exists (an unknown) time T_D such that $v_O \in \Delta(v_I)$, where $v_I \in \mathcal{I}$ denotes the input configuration that consists of input values of all correct processes and $v_O \in \mathcal{O}$ 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 $\Pi^A_{\mathcal{A}}$ solves an *accountable* agreement task $\mathcal{A} = (\mathcal{I}, \mathcal{O}, \Delta, t_{\mathcal{A}})$ if and only if the following holds:

- *$\mathcal{A}$-Solution:* If $t \leq t_{\mathcal{A}}$, $\Pi^A_{\mathcal{A}}$ 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 a proof of culpability of all detected processes.

4.2 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 $v_O \in \mathcal{O}$ such that $(P_i, O_i \neq \perp) \in v_O$ and $(P_j, O_j = \perp) \in v_O$; or
(2) *"Partial-Decidability":* For every $v_I \in \mathcal{I}$ such that there exists $v_O \in \Delta(v_I)$ where $(P_i, O_i = v \neq \perp) \in v_O$ and $(P_j, O_j = \perp) \in v_O$, the following holds:

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

"All-or-None-Decidability" characterizes all the problems in which either every process halts or none does. For instance, Byzantine consensus [15] and Byzantine reliable broadcast [3] 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 in which no correct process ever halts. Byzantine consistent broadcast [3] is the single agreement task we are aware of that satisfies "Partial-Decidability". However, the significance of Byzantine consistent broadcast (e.g., for implementing cryptocurrencies [12]) motivated us to consider the "Partial-Decidability" property.

4.3 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 problem $\mathcal{A}$ when $t \leq t_{\mathcal{A}}$ if $\mathcal{A}$ satisfies the "All-or-None-Decidability" property.

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". Then, Algorithm 4 solves $\mathcal{A}$ if $t \leq t_{\mathcal{A}}$.

Proof. In no correct process ever outputs a value at line 8, then the lemma trivially holds.

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

```
Implements:
    Accountable Agreement Task $\mathcal{A}$, instance $a - \mathcal{A}$
Uses:
    Protocol that solves agreement task $\mathcal{A}$, instance $\Pi_{\mathcal{A}}$        ▹ Protocol to be transformed
    Accountable Confirmer, instance $ac$
upon event $\langle a - \mathcal{A}, Input \mid input \rangle$ do
    trigger $\langle \Pi_{\mathcal{A}}, Input \mid input \rangle$
upon event $\langle \Pi_{\mathcal{A}}, Output \mid output \rangle$ do
    trigger $\langle ac, Submit \mid output \rangle$
upon event $\langle ac, Confirm \mid confirmation \rangle$ do
    trigger $\langle a - \mathcal{A}, Output \mid confirmation \rangle$
upon event $\langle ac, Detect \mid F, proof \rangle$ do
    trigger $\langle a - \mathcal{A}, Detect \mid F, proof \rangle$
```

Otherwise, each correct process eventually outputs a value at line 8. Moreover, all correct processes output the exact same value v (since $\mathcal{A}$ is an agreement task). Therefore, all correct process submit the same value v to accountable confirmer (line 9). By terminating convergence of accountable confirmer, all correct processes eventually confirm value v (line 10) and output it (line 11). 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 $\mathcal{A}$ when $t \leq t_{\mathcal{A}}$ if $\mathcal{A}$ satisfies the "Partial-Decidability" property.

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

Proof. Let v_I denotes the specific input configuration of $\mathcal{A}$. We consider two cases:

- There does not exist $v_O \in \Delta(v_I)$ such that $(P_i, O_i \neq \bot) \in v_O$ and $(P_j, O_j = \bot) \in v_O$: In this case, the proof is identical to the proof of Lemma 1.
- Otherwise: Since $\mathcal{A}$ is an easily accountable agreement task, we conclude that all processes that output a value at line 8 output the same value v. Therefore, any process that outputs a value at line 11 outputs the value v (ensured by validity of accountable confirmer). Finally, once the system stabilizes at time T_S (the system stabilizes at time T_S if and only if no correct process P_i with $O_i = \bot$ updates its output value after T_S), the fact that any subset of processes could halt and the fact that all halted processes output v imply that Algorithm 4 solves $\mathcal{A}$.

The lemma holds since it is satisfied in both possible cases. □

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

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

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. □

5 CONCLUSION

We presented $\mathcal{ABC}$, the generic and simple transformation that allows Byzantine consensus protocols to obtain accountability. Besides its simplicity, $\mathcal{ABC}$ is efficient: it introduces an overhead of (1) $O(n^2)$ exchanged bits of information in the common case, and (2) $O(n^3)$ exchanged bits of information in the degraded case. 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 transformation for problems not covered by our generalized $\mathcal{ABC}$ transformation, like Byzantine lattice and k-set agreement problems, and (2) circumventing the cubic lower bound using randomization techniques.

REFERENCES

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

[2] Buterin, V., and Griffith, V. Casper the friendly finality gadget. *arXiv preprint arXiv:1710.09437* (2017).

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

[4] Cachin, C., Kursawe, K., Petzold, F., and Shoup, V. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference* (2001), Springer, pp. 524–541.

[5] Castro, M., Liskov, B., et al. Practical byzantine fault tolerance. In *OSDI* (1999), vol. 99, pp. 173–186.

[6] Chaudhuri, S. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation 105*, 1 (1993), 132–158.

[7] Civit, P., Gilbert, S., and Gramoli, V. Brief announcement: Polygraph: Accountable byzantine agreement. In *34th International Symposium on Distributed Computing, DISC 2020, October 12-16, 2020, Virtual Conference* (2020), H. Attiya, Ed., vol. 179 of *LIPIcs*, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, pp. 45:1–45:3.

[8] Crain, T., Gramoli, V., Larrea, M., and Raynal, M. Dbft: Efficient leaderless byzantine consensus and its application to blockchains. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)* (2018), IEEE, pp. 1–8.

[9] de Souza, L. F., Kuznetsov, P., Rieutord, T., and Tucci Piergiovanni, S. Accountability and reconfiguration: Self-healing lattice agreement. *CoRR abs/2105.04909* (2021).

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

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

[12] Guerraoui, R., Kuznetsov, P., Monti, M., Pavlovic, M., and Seredinschi, D.-A. At2: asynchronous trustworthy transfers. *arXiv preprint arXiv:1812.10844* (2018).

[13] Haeberlen, A., Kouznetsov, P., and Druschel, P. Peerreview: practical accountability for distributed systems. In *Proceedings of the 21st ACM Symposium on Operating Systems Principles 2007, SOSP 2007, Stevenson, Washington, USA, October 14-17, 2007* (2007), T. C. Bressoud and M. F. Kaashoek, Eds., ACM, pp. 175–188.

[14] Haeberlen, A., and Kuznetsov, P. The fault detection problem. In *Principles of Distributed Systems, 13th International Conference, OPODIS 2009, Nîmes, France, December 15-18, 2009. Proceedings* (2009), T. F. Abdelzaher, M. Raynal, and N. Santoro, Eds., vol. 5923 of *Lecture Notes in Computer Science*, Springer, pp. 99–114.

[15] Lamport, L., Shostak, R., and Pease, M. The byzantine generals problem. In *Concurrency: the Works of Leslie Lamport.* 2019, pp. 203–226.

[16] 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.

[17] Milosevic, Z., Hutle, M., and Schiper, A. Unifying byzantine consensus algorithms with weak interactive consistency. In *International Conference On Principles Of Distributed Systems* (2009), Springer, pp. 300–314.

[18] Sheng, P., Wang, G., Nayak, K., Kannan, S., and Viswanath, P. BFT protocol forensics. *CoRR abs/2010.06785* (2020).

[19] 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* (2019), pp. 347–356.