ABSTRACT

Byzantine fault-tolerant (BFT) protocols allow a group of replicas to come to consensus even when some of the replicas are Byzantine faulty. There exist multiple BFT protocols to securely tolerate an optimal number of faults $t$ under different network settings. However, if the number of faults $f$ exceeds $t$ then security could be violated. In this paper we mathematically formalize the study of forensic support of BFT protocols: we aim to identify (with cryptographic integrity) as many of the malicious replicas as possible and in as distributed manner as possible. Our main result is that forensic support of BFT protocols depends heavily on minor implementation details that do not affect the protocol’s security or complexity. Focusing on popular BFT protocols (PBFT, HotStuff, Algorand) we exactly characterize their forensic support, showing that there exist minor variants of each protocol for which the forensic supports vary widely. We show strong forensic support capability of LibraBFT, the consensus protocol of Diem cryptocurrency; our lightweight forensic module implemented on a Diem client is open-sourced [4] and is under active consideration for deployment in Diem. Finally, we show that all secure BFT protocols designed for 2t + 1 replicas communicating over a synchronous network forensic support is inherently nonexistent; this impossibility result holds for all BFT protocols and even if one has access to the states of all replicas (including Byzantine ones).

CCS CONCEPTS

• Security and privacy → Distributed systems security; • Computer systems organization → Dependable and fault-tolerant systems and networks.

KEYWORDS

forensics; BFT protocols; blockchains

*The first two authors contributed equally to this work.
Table 2: Summary of results; the forensic support values of Table 1. We emphasize that each of the protocol variants is safe and live when \( f \leq t \) (here \( n = 3t + 1 \) for all the protocols considered) but their forensic supports are quite different.

Security attacks on BFT protocols can be far more subtle than simply double voting or overt equivocation (when the culpable replicas are readily identified), and the forensic analysis is correspondingly subtle. This paper brings to fore a new dimension for forensic protocol analysis in which the culprit is identified only after a successful safety attack (agreement violation), and forensic support can be used to identify the culprit.

We analyze the forensic support of classical and state-of-the-art BFT protocols for Byzantine Agreement. Our main findings, all in the settings of safety violation (agreement violation), are the following, summarized in Table 2.

- **Parameter \( d \)**. We show that the number of culpable replicas \( d \) that can be identified is either 0 or as large as \( t + 1 \). In other words, if at least one replica can be identified then we can also identify the largest possible number, \( t + 1 \) replicas; the only exception is for HotStuff-null, where in a successful safety attack, we can identify the culpability of one malicious replica.
- **Parameter \( m \)**. We show that the maximum number of Byzantine replicas \( m \) allowed for non-trivial forensic support (i.e., \( d > 0 \)) cannot be more than \( 2t \). Furthermore, any forensic support feasible with \( m \) is also feasible with \( m \) being its largest value \( 2t \), i.e., if \( (m, k, d) \) forensic support is feasible, then \( (2t, k, d) \) is also feasible.
- **Parameter \( k \)**. Clearly at least one replica’s transcript is needed for forensic analysis, so \( k = 1 \) is the least possible value. This suffices for several of the BFT protocol variants. However for HotStuff-hash, \( k \) needs to be at least \( t + 1 \) for any nontrivial forensic analysis.
- **Strong forensic support**. The first three items above imply that the strongest possible forensic support is \( (2t, 1, t + 1) \). Further, the BFT protocols in Table 2 that achieve any non-trivial forensic support automatically achieve the strongest possible forensic support (the only exception is HotStuff-hash, for which the forensic support we identified is the best possible).
- **Impossibility**. For certain variants of BFT protocols (PBFT-MAC, HotStuff-null, and Algorand), even with transcripts from all honest replicas, non-trivial forensic support is simply not possible, i.e., \( d = 0 \) even if \( m \) is set to its smallest and \( k \) set to its largest possible values \( (t + 1) \) and \( 2t \) respectively; again, HotStuff-null allows the culpability of a single malicious replica.
- **Practical impact**. Forensic support is of immediate interest to practical blockchain systems; we conduct a forensic support analysis of LibraBFT, the consensus protocol in the new cryptocurrency Diem, and show in-built strong forensic support. We have implemented the corresponding forensic analysis algorithm inside of a Diem client and built a forensics dashboard; our reference implementation is available open-source [4] and is under active consideration for deployment in Diem.

**BFT with \( n = 2t + 1 \)**. For a secure BFT protocol operating on a synchronous network, the ideal setting is \( n = 2t + 1 \). For every such protocol we show that at most one culpable replica can be identified (i.e., \( d \) is at most \( 1 \)) even if we have access to the state of all honest nodes, i.e., \( k = t \).

Outline. We describe our results in the context of related work in §2; to the best of our knowledge, this is the first paper to systematically study BFT protocol forensics. The formal forensic support problem statement and security model is in §3. The forensic support of PBFT, HotStuff and Algorand (and variants) are explored in §4, §5, §6, respectively. Our forensic study of LibraBFT and implementation of the corresponding forensic protocol is the focus of §7. In appendix §A, we present the forensic support analysis for VABA, a state-of-the-art efficient BFT for asynchronous network conditions. The impossibility of forensic support for all BFT protocols operating in the classical \( n = 2t + 1 \) synchronous network setting is shown in §8. Our choice of the 5 protocols studied here (PBFT, HotStuff, Algorand, LibraBFT, VABA) is made with the goal of covering a variety of settings: (a) partially synchronous vs asynchronous; (b) authenticated vs non-authenticated; (c) player replaceable vs irreplaceable; (d) chained version vs single-shot version; (e) variants that communicate differing amounts of auxiliary information. Stitching the results across the 5 different protocols into a coherent theory of forensic support of abstract BFT protocols is an exciting direction of future work; this is discussed in §8.
2 RELATED WORK

BFT protocols. PBFT [10, 11] is the first practical BFT SMR protocol in the partially synchronous setting, with quadratic communication complexity of view change. HotStuff [32] is the first partially synchronous SMR protocol that enjoys both a linear communication of view change and optimistic responsiveness. Streamlet [12] is another SMR protocol known for its simplicity and textbook construction. Asynchronous Byzantine agreement is solved by [2] with asymptotically optimal communication complexity and round number. Synchronous protocols such as [1, 3] aim at optimal latency. Algorand [14, 18] designs a committee self-selection mechanism, and the Byzantine agreement protocol run by the committee decides the output for all replicas.

Beyond one-third faults. The seminal work of [17] shows that it is impossible to solve BA when the adversary corrupts one-third replicas for partially synchronous communication (the same bound holds for SMR). In BFT2F [22], a weaker notion of safety is defined, and a protocol is proposed such that when the adversary corrupts more than one-third replicas, the weaker notion of safety remains secure whereas the original safety might be violated. However, the weaker notion of safety does not protect the system against common attacks, e.g., double-spending attack in distributed payment systems. Flexible BFT [23] considers the case where clients have different beliefs in the number of faulty replicas and can act to confirm accordingly. Its protocol works when the sum of Byzantine faults and alive-but-corrupt faults, a newly defined type of faults, are beyond one-third. Two recent works [21, 31] propose BFT SMR protocols that can tolerate more than one-third Byzantine faults after some specific optimistic period. The goal of these works is to mask the effects of faults, even when they are beyond one-third, quite different from the goals of forensic analysis.

Distributed system forensics. Accountability has been discussed in seminal works [19, 20] for distributed systems in general. In the lens of BFT consensus protocols, accountability is defined as the ability for a replica to prove the culpability of a certain number of Byzantine replicas in [15]. Polygraph, a new BFT protocol with high communication complexity $O(n^4)$, is shown to attain this property in [15]. Reference [28] extends Polygraph to an SMR protocol, and devises a finality layer to "merge" the disagreement. Finality and accountability are also discussed in other recent works; examples include Casper [8], GRANDPA [29], and Ebb-and-flow [26]. Casper and GRANDPA identify accountability as a central problem and design their consensus protocols around this goal. Ebb-and-flow [26] observes that accountability is immediate in BFT protocols for safety violations via equivocating votes; however, as pointed out in [15], safety violations can happen during the view change process and this is the step where accountability is far more subtle.

Reference [15] argues that PBFT is not accountable and cannot be modified to be accountable without significant change/cost. We point out that the definition of accountability in [15] is rather narrow: two replicas with differing views have to themselves be able to identify culpability of malicious replicas. On the other hand, in forensic support, we study the number of honest replicas (not necessarily restricted to the specific two replicas which have identified a security breach) that can identify the culpable malicious replicas. Thus the definition of forensic support is more flexible than the one on accountability. Moreover, our work shows that we can achieve forensic support for protocols such as PBFT and HotStuff without incurring additional communication complexity other than (i) sending a proof of culpability to the client in case of a safety violation, and (ii) the need to use aggregate signatures instead of threshold signatures.

3 PROBLEM STATEMENT AND MODEL

The goal of state machine replication (SMR) is to build a replicated service that takes requests from clients and provides the clients with the interface of a single non-faulty server, i.e., each client receives the same totally ordered sequence of values. To achieve this, the replicated service uses multiple servers, also called replicas, some of which may be Byzantine, where a faulty replica can behave arbitrarily. A secure state machine replication protocol satisfies two guarantees. Safety: Any two honest replicas cannot output different sequences of values. Liveness: A value sent by a client will eventually be output by honest replicas.

SMR setting also has external validity, i.e., replicas only output non-duplicated values sent by clients. These values are eventually learned by the clients. Depending on the context, a replica may be interested in learning about outputs too. Hence, whenever we refer to a client for learning purposes, it can be an external entity or a server replica. A table of notations is in Table 1.

In this paper, for simplicity, we focus on the setting of outputting a single value instead of a sequence of values. The safety and liveness properties of SMR can then be expressed using the following definition:

Definition 3.1 (Validated Byzantine Agreement). A protocol solves validated Byzantine agreement among $n$ replicas tolerating a maximum of $t$ faults, if it satisfies the following properties:

(Agreement) Any two honest replicas output values $v$ and $v'$, then $v = v'$.

(Validity) If an honest replica outputs $v$, $v$ is an externally valid value, i.e., $v$ is signed by a client.

(Termination) All honest replicas start with externally valid values, and all messages sent among them have been delivered, then honest replicas will output a value.

Forensic support. Traditionally, consensus protocols provide guarantees only when $f \leq t$. When $f > t$, there can be a safety or liveness violation; this is the setting of study throughout this paper. Our goal is to provide forensic support whenever there is a safety violation (or agreement violation) and the number of Byzantine replicas in the system are not too high. In particular, if the actual number of Byzantine faults are bounded by $m$ (for some $m > t$) and there is a safety violation, we can detect $d$ Byzantine replicas using a forensic protocol. The protocol takes as input, the transcripts of honest parties, and outputs and irrefutable proof of $d$ culprits. With the irrefutable proof, any party (not necessarily in the BFT system) can be convinced of the culprits’ identities without any assumption on the number of honest replicas. However, if even with transcripts from all honest replicas, no forensic protocol can output such a proof, the consensus protocol has no forensic support (denoted as “None” in Table 2). Note that when we say a protocol
has no forensic support, we are referring to an impossibility w.r.t.
providing irrefutable proof for d culprits (more precisely, in the
context of Definition 3.2). In a general sense, there are other ways
to provide forensics related to hardware, software, and network
except for non-repubidation in the protocol.

To provide forensic support, we consider a setting where a client
observes the existence of outputs for two conflicting (unequal)
values. By running a forensic protocol, the client sends (possibly a
subset of) these conflicting outputs to all replicas and waits for their
replies. Some of these replies may be “witnesses” and may have
(partial) information required to construct the irrefutable proof.
After receiving responses from the replicas, the client constructs
the proof. We denote by k the total number of transcripts from
different honest replicas that are stored by the client to construct
the proof.

Definition 3.2. (m, k, d)-Forensic Support. If t < f ≤ m and
two honest replicas output conflicting values, then using the tran-
scripts of all messages received from k honest replicas during the
protocol, a client can provide an irrefutable proof of culpability of
at least d Byzantine replicas.

Other assumptions. We consider forensic support for multiple
protocols each with their own network assumptions. For PBFT and
HotStuff, we assume a partially synchronous network [17]. For VABA [2] and Algorand [14], we suppose asynchronous and
synchronous networks respectively.

We assume all messages are digitally signed except for one vari-
ant of PBFT (§4.3) that sometimes relies on the use of Message
Authenticated Codes (MACs). Some protocols, e.g., HotStuff, VABA,
use threshold signatures. For forensic purposes, we assume multi-
signatures [7] instead (possibly worsening the communication com-
plexity in the process). Whenever the number of signatures exceeds
a threshold, the resulting aggregate signature is denoted by a pair
σ = (σagg, ε), where ε is a bitmap indicating whose signatures
are included in σagg. We define the intersection of two aggregated
messages to be the set of replicas who sign both messages, i.e.,
σ ∩ σ′ := {i | σ[i] ∧ σ′[i] = 1}. An aggregate signature serves
as a quorum certificate (QC) in our protocols, and we will use the
two terms interchangeably. We assume a collision resistant crypto-
graphic hash function.

4 FORENSIC SUPPORT FOR PBFT

PBFT is a classical partially synchronous consensus protocol that
provides an optimal resilience of t Byzantine faults out of n = 3t + 1
replicas. However, when the actual number of faults f exceeds t, it
does not provide any safety or liveness. In this section, we show
that when f > t and in case of a safety violation, the variant of the
PBFT protocol (referred to as PBFT-PK) where all messages sent
by parties are signed, has the strongest forensic support. Further,
we show that for an alternative variant where parties sometimes
only use MACs (referred to as PBFT-MAC), forensic support is
impossible.

4.1 Overview

We start with an overview focusing on a single-shot version of
PBFT, i.e., a protocol for consensus on a single value. The protocol
described here uses digital signatures to authenticate all messages
and routes messages through leaders as shown in [27]; however we
note that our arguments for PBFT-PK also apply to the original
protocol in [11].

The protocol proceeds in a sequence of consecutive views de-
noted as view number e = 1, 2, · · · . A higher view is a view with a
larger view number. Each view has a unique leader. Each view of
PBFT progresses as follows:

- Pre-prepare. The leader proposes a NewView message contain-
ing a proposal v and a status certificate M (explained later)
to all replicas.
- Prepare. On receiving the first NewView message containing
a valid value v in a view e, a replica sends Prepare for v if it is
safe to vote based on a locking mechanism (explained later).
It sends this vote to the leader. The leader collects 2t + 1 such
votes to form an aggregate signature prepareQC. The leader sends
prepareQC to all replicas.
- Commit. On receiving a prepareQC in view v containing message
v, a replica locks on (v, e) and sends Commit to the leader.
The leader collects 2t + 1 such votes to form an aggregate signature
commitQC. The leader sends commitQC to all replicas.
- Reply. On receiving commitQC from the leader, replicas output
v and send a Reply (along with commitQC) to the client.

Once a replica locks on a value v in view v, the lock is the
leader collects 2t + 1 such values as a status certificate
commitQC. The leader sends commitQC to all replicas.

Uniqueness within a view. Within a view, safety is ensured by
votes in either round. Since a replica only votes once for the first
valid value it receives, by a quorum intersection argument, two
conflicting values cannot both obtain commitQC when f ≤ t.

Safety across views. Safety across views is ensured by the use
of locks and the status certificate. First, observe that if a replica
r outputs a value v in view v, then a quorum of replicas lock on
(v, e). When f ≤ t, this quorum includes a set H of at least t + 1
honest replicas. For any replica in H to update to a higher lock, they
need a prepareQC in a higher view e′ > e, which in turn requires a
vote from at least one of these honest replicas in view e′. However,
replicas in H will vote for a conflicting value v′ in a higher view
only if it is accompanied by a status certificate M containing 2t + 1
locks that are not on value v. When f ≤ t, honest replicas in M
intersect with honest replicas in H at least one replica – this honest
replica will not vote for a conflicting value v′.
4.2 Forensic Analysis for PBFT-PK

The agreement property for PBFT holds only when \( f \leq t \). When the number of faults are higher, this agreement property (and even termination) can be violated. In this section, we show how to provide forensic support for PBFT when the agreement property is violated. We show that, if two honest replicas output conflicting values \( v \) and \( v' \) due to the presence of \( t < f \) \( \leq m \) Byzantine replicas, our forensic protocol can detect \( t + 1 \) Byzantine replicas with an irrefutable proof. For each of the possible scenarios in which safety can be violated, the proof shows exactly what property of PBFT was not respected by the Byzantine replicas. The irrefutable proof explicitly uses messages signed by the Byzantine parties, and is thus only applicable to the variant PBFT-PK where all messages are signed.

**Intuition.** In order to build intuition, let us assume \( n = 3t + 1 \) and \( f = t + 1 \) and start with a simple scenario: two honest replicas output values \( v \) and \( v' \) in the same view. It must then be the case that a commitQC is formed for both \( v \) and \( v' \). Due to a quorum intersection argument, it must be the case that all replicas in the intersection have voted for two conflicting values to break the uniqueness property. Thus, all the replicas in the intersection are culpable. For PBFT-PK, the commitQC (as well as prepareQC) for the two conflicting values act as the irrefutable proof for detecting \( t + 1 \) Byzantine replicas.

![Figure 1: An example sequence of events in the PBFT-PK protocol that leads to replicas i and j outputting different values.](image)

When two honest replicas output conflicting values in different views, there are many different sequences of events that could lead to such a disagreement. One such sequence is described in Figure 1. The replicas are split into three sets: the blue set and the green set are honest replicas each of size \( t \) and the red replicas are Byzantine replicas of size \( t + 1 \).

- In view \( e \), replica \( i \) outputs \( v \) due to commitQC formed with the COMMIT from the blue honest set and the Byzantine red set. At the end of the view, replicas in the blue and red set hold locks \( (v, e) \) whereas the green set holds a lower lock for a different value.
- In the next few views, no higher locks are formed. Thus, the blue and the red set still hold locks \( (v, e) \).
- Suppose \( e^* \) is the first view where a higher lock is formed. At the start of this view, the leader receives locks from the honest green set who hold lower-ranked locks and the Byzantine red set who maliciously send lower-ranked locks. The set of locks received by the leader is denoted by \( M \) and suppose the highest lock was received for \( v' \). The leader proposes \( v' \) along with \( M \). This can make any honest replica “unlock” and vote for \( v' \) and form quorum certificates in this view.
- In some later view \( e' \), replica \( j \) outputs \( v' \).

With this sequence of events, consider the following questions:

1. Is this an admissible sequence of events?  
2. How do we find the culpable Byzantine replicas? What does the irrefutable proof consist of?  
3. How many replica transcripts do we need to construct the proof?

To answer the first question, the only nontrivial part in our example is the existence of a view \( e^* \) where a higher lock is formed. However, such a view \( e < e^* \leq e' \) must exist because replica \( j \) outputs in view \( e' \) and a higher lock must be formed at the latest in view \( e' \).

For the second question, observe that both the red replicas as well as the green replicas sent locks lower than \( (v, e) \) to the leader in \( e^* \). However, only the red replicas also sent commit messages for value \( v \) in view \( e \). Thus, by intersecting the set of commit messages for value \( v \) in view \( e \) and the messages forming the status certificate sent to the leader of \( e^* \), we can obtain a culpable set of \( t + 1 \) Byzantine replicas. So the proof for PBFT-PK consists of the commitQC in \( e \) and the status certificate in \( e^* \), which indicates that the replicas sent a lower lock in view \( e^* \) despite having access to a higher lock in a lower view \( e \).

For the third question, the NewView message containing the status certificate \( M \) in view \( e^* \) can act as the proof, so only one transcript needs to be stored.

**Forensic protocol for PBFT-PK.** Algorithm 1 describes the entire protocol to obtain forensic support atop PBFT-PK. For completeness, we also provide a complete description of the PBFT-PK protocol in Algorithm 5. Each replica keeps all received messages as transcripts. When two honest replicas output conflicting values in different views, there are many different sequences of events that could lead to such a disagreement. One such sequence is described in Figure 1. The replicas are split into three sets: the blue set and the green set are honest replicas each of size \( t \) and the red replicas are Byzantine replicas of size \( t + 1 \).

- In view \( e \), replica \( i \) outputs \( v \) due to commitQC formed with the COMMIT from the blue honest set and the Byzantine red set. At the end of the view, replicas in the blue and red set hold locks \( (v, e) \) whereas the green set holds a lower lock for a different value.
- In the next few views, no higher locks are formed. Thus, the blue and the red set still hold locks \( (v, e) \).
- Suppose \( e^* \) is the first view where a higher lock is formed. At the start of this view, the leader receives locks from the honest green set who hold lower-ranked locks and the Byzantine red set who maliciously send lower-ranked locks. The set of locks received by the leader is denoted by \( M \) and suppose the highest lock was received for \( v' \). The leader proposes \( v' \) along with \( M \). This can make any honest replica “unlock” and vote for \( v' \) and form quorum certificates in this view.
- In some later view \( e' \), replica \( j \) outputs \( v' \).

With this sequence of events, consider the following questions:

1. Is this an admissible sequence of events?  
2. How do we find the culpable Byzantine replicas? What does the irrefutable proof consist of?  
3. How many replica transcripts do we need to construct the proof?

To answer the first question, the only nontrivial part in our example is the existence of a view \( e^* \) where a higher lock is formed. However, such a view \( e < e^* \leq e' \) must exist because replica \( j \) outputs in view \( e' \) and a higher lock must be formed at the latest in view \( e' \).

For the second question, observe that both the red replicas as well as the green replicas sent locks lower than \( (v, e) \) to the leader in \( e^* \). However, only the red replicas also sent commit messages for value \( v \) in view \( e \). Thus, by intersecting the set of commit messages for value \( v \) in view \( e \) and the messages forming the status certificate sent to the leader of \( e^* \), we can obtain a culpable set of \( t + 1 \) Byzantine replicas. So the proof for PBFT-PK consists of the commitQC in \( e \) and the status certificate in \( e^* \), which indicates that the replicas sent a lower lock in view \( e^* \) despite having access to a higher lock in a lower view \( e \).

For the third question, the NewView message containing the status certificate \( M \) in view \( e^* \) can act as the proof, so only one transcript needs to be stored.

**Forensic protocol for PBFT-PK.** Algorithm 1 describes the entire protocol to obtain forensic support atop PBFT-PK. For completeness, we also provide a complete description of the PBFT-PK protocol in Algorithm 5. Each replica keeps all received messages as transcripts and maintains a set \( Q \) containing all received NewView messages (line 2). If a client observes the replies of two conflicting values, it first checks if two values are output in the same view (line 9). If yes, then any two commitQC for two different output values can provide a culpability proof for at least \( t + 1 \) replicas (lines 19-22). Otherwise, the client sends a request for a possible proof between two output views \( e, e' \) (lines 13). Each replica looks through the set \( Q \) for the NewView message in the smallest view \( e^+ > e \) such that the status certificate \( M \) contains the highest lock \( (e'', e''') \) where \( e'' > e \) and \( e'' < e' \) and sends it to the client (line 7). If inside \( M \) there are conflicting locks in the same view, the intersection of them proves at least \( t + 1 \) culprits (line 15), otherwise the intersection of \( M \) and the commitQC proves at least \( t + 1 \) culprits (line 18).

The following theorem sharply characterizes the forensic support capability of PBFT-PK. As long as \( m \leq 2t \), the best possible forensic support is achieved (i.e., \( k = 1 \) and \( d = t + 1 \)). Algorithm 1 can be used to irrefutably detect \( t + 1 \) Byzantine replicas. Conversely, if \( m > 2t \) then no forensic support is possible (i.e., \( k = n - f \) (messages from all honest nodes)) and \( d = 0 \).
The quorums Client can obtain the proof by storing the
Client can obtain the culpability proof based on two

Algorithm 1 Forensic protocol for PBFT-PK Byzantine agreement

1: as a replica running PBFT-PK
2: $Q \leftarrow$ all NewView messages in transcript
3: upon receiving (Request-Proof, $e, v, e'$) from a client do
4: for $m \in Q$ do
5: $(v', e'') \leftarrow$ the highest lock in $m.M$
6: if $(m.e \in (e, e')) \land (e'' \neq v \lor (e'' \leq e))$ then
7: send (NewView, $m$) to the client
8: as a client
9: upon receiving two conflicting Reply messages do
10: if the two messages are from different views then
11: $(\text{Reply}, e, v, \sigma) \leftarrow$ the message from lower view
12: $e' \leftarrow$ the view number of Reply from higher view
13: broadcast (Request-Proof, $e, v, e'$)
14: wait for: (NewView, $m$) s.t. $m.e \in (e, e'] \land (e'' \neq v) \land (e'' \leq e)$ where $(v'', e'')$ is the highest lock in $m.M$.
15: if in $m.M$ there are two locks $(e'', v_1, \sigma_1)$ and $(e'', v_2, \sigma_2)$ s.t. $v_1 \neq v_2$ then
16: output $\sigma_1 \cap \sigma_2$
17: else
18: output the intersection of senders in $m.M$ and signers of $\sigma$.
19: else
20: $(\text{Reply}, e, v, \sigma) \leftarrow$ first Reply message
21: $(\text{Reply}, e, v', \sigma') \leftarrow$ second Reply message
22: output $\sigma \cap \sigma'$

Theorem 4.1. With $n = 3t + 1$, when $f > t$, if two honest replicas output conflicting values, PBFT-PK provides $(2t + 1, t + 1)$-forensic support. Further $(2t + 1, n - f, d)$-forensic support is impossible with $d > 0$.

Proof. We prove the forward part of the theorem below. The converse (impossibility) is proved in §C.1. Suppose the values $v$ and $v'$ are output in views $e$ and $e'$ respectively.

Case $e = e'$. Culpability. The quorums commitQC for $v$ and commitQC for $v'$ intersect in $t + 1$ replicas. These $t + 1$ replicas should be Byzantine since the protocol requires a replica to vote for at most one value in a view.

Witnesses. Client can obtain the culpability proof based on two commitQC. No additional communication is needed in this case ($k = 0$).

Case $e \neq e'$. Culpability. If $e \neq e'$, then WLOG, suppose $e < e'$. Since $v$ is output in view $e$, it must be the case that $2t + 1$ replicas are locked on $(v, e)$ at the end of view $e$ (if they are honest). Now consider the first view $e < e'' \leq e$ in which a higher lock $(v'', e'')$ is formed (not necessarily known to any honest party) where $v'' \neq v$ (possibly $v'' = v'$). Such a view must exist since $v'$ is output in view $e' > e$ and a lock will be formed in at least view $e'$. Consider the status certificate $M$ sent by the leader of view $e''$ in its NewView message. $M$ must contain $2t + 1$ locks; each of these locks must be from view $e'' \leq e$, and a highest lock among them is $(v'', e'')$.

We consider two cases based on whether the status certificate contains two different highest locks: (i) there exist two locks $(v'', e'')$ and $(v'', e'')$ s.t. $o'' \neq o'''$ in $M$. (ii) $(o'', e'')$ is the only highest lock in $M$. For the first case, since two locks are formed in the same view, the two quorums forming the two locks in view $e''$ intersect in $t + 1$ replicas. These replicas are Byzantine since they voted for more than one value in view $e$.

For the second case, $(o'', e'')$ is the only highest lock in the status certificate $M$. $M$ intersects with the $2t + 1$ signers of commitQC in view $e$ at $t + 1$ Byzantine replicas. These replicas are Byzantine because they had a lock on $v \neq v''$ in view $e \geq e''$ but sent a different lock to the leader of view $e' > e$.

Witnesses. Client can obtain the proof by storing the NewView message containing the status certificate $M$ in $e'$. Only one witness is needed to provide the NewView message $(k = 1)$. The status certificate $M$ and the first commitQC act as the irrefutable proof. \[
\]

4.3 Impossibility for PBFT-MAC

We now show an impossibility for a variant of PBFT proposed in [11, Section 5]. The arguments here also apply to the variant in [10]. Compared to §4.1, the only difference in this variant is (i) Prepare and Commit messages are authenticated using MACs instead of signatures, and (ii) these messages are broadcast instead of routing them through the leader.

Intuition. The key intuition behind the impossibility relies on the absence of digital signatures which were used to "log" the state of a replica when some replica $i$ outputs a value. In particular, if we consider the example in Figure 1, while $i$ receives $2t + 1$ COMMIT messages for value $v$, these messages are not signed. Thus, if $t + 1$ Byzantine replicas vote for a different value $v'$, $v'$ can be output by a different replica. The absence of a verifiable proof stating the set of replicas that sent a COMMIT to replica $i$ prevents any forensic analysis. We formalize this intuition below.

Theorem 4.2. With $n = 3t + 1$, when $f > t$, if two honest replicas output conflicting values, $(t + 1, 2t, d)$-forensic support is impossible with $d > 0$ for PBFT-MAC.

Proof. Suppose the protocol provides forensic support to detect $d \geq 1$ replicas with irrefutable proof. To prove this result, we construct two worlds where a different set of $t + 1$ replicas are Byzantine in each world but a forensic protocol cannot be correct in both worlds. We fix $f = t + 1$, although the arguments will apply for any $f > t$.

Let there be four replica partitions $P, Q, R, \{x\}$. $|P| = |Q| = |R| = t$, and $x$ is an individual replica. In both worlds, the conflicting
outputs are presented in the same view $e$. Suppose the leader is a replica from set $R$.

**World 1.** Let $P$ and $x$ be Byzantine replicas in this world. The honest leader from set $R$ in view $e$ proposes $v'$. Parties in $P$, $x$ and $Q$ send $\text{Prepare}$ and $\text{Commit}$ messages (authenticated with MACs) for value $v'$. Due to partial synchrony, none of these messages arrive at $P$. At the end of view $e$, only $R$ and one replica $q$ in $Q$ receive enough $\text{Commit}$ messages and send replies to the client. So the client receives the first set of $t+1$ replies for value $v'$, which contain the same quorum $P, x, Q$.

The Byzantine parties in $P$ and $x$ simulate a proposal from the leader for $v$, and the sending of $\text{Prepare}$ and $\text{Commit}$ messages within $R, P$ and $x$. The simulation is possible due to the absence of a PKI. At the end of view $e$, $P$ and $x$ obtain enough $\text{Commit}$ messages and send replies to the client. Thus, the client receives the second set of $t + 1$ replies for value $v$, which contain the same quorum $P, R, x$.

Client starts the forensic protocol.

During the forensic protocol, Byzantine $P$ and $x$ only present the votes for $v$, forged votes from $R$ as their transcripts. Since $t+1$ parties have output each of $v$ and $v'$, there is a safety violation. Since the protocol has forensic support for $d \geq 1$, using these transcripts, the forensic protocol determines some subset of $P$ and $x$ are culpable.

**World 2.** Let $R$ and $q$ (one replica in $Q$) be Byzantine replicas in this world. The Byzantine leader in view $e$ proposes $v$ to $P, R$ and $x$. They send $\text{Prepare}$ and $\text{Commit}$ messages (authenticated with MACs) for value $v$. These messages do not arrive at $Q$. At the end of view $e$, parties in $P$ and $x$ output $v$. So the client receives the first set of $t + 1$ replies for value $v$, which contain the same quorum $P, R, x$. Client starts the forensic protocol.

Similarly, the leader sends $v'$ to $Q, R$ and $x$. The proposal does not arrive at $x$. Only $Q$ and $R$ send $\text{Prepare}$ and $\text{Commit}$ messages (authenticated with MACs) for $v'$. These messages do not arrive at $P$. However, $R$ and $q$ forge $\text{Prepare}$ and $\text{Commit}$ messages from $x$. At the end of view $e$, $R$ and $q$ output $v'$. So the client receives the second set of $t + 1$ replies for value $v'$, which contain the same quorum $R, x, Q$. Client starts the forensic protocol.

During the forensic protocol, Byzantine $R$ and $q$ sends the same transcripts as in World 1 by dropping votes for $v$ and forging votes from $x$. Again, since $t + 1$ parties have output each of $v$ and $v'$, there is a safety violation. However, observe that the transcript presented to the forensic protocol is identical to that in World 1. Thus, the forensic protocol outputs some subset of $P$ and $x$ as culpable replicas. In World 2, this is incorrect since replicas in $P$ and $x$ are honest. This completes the proof.

5 FORENSIC SUPPORT FOR HOTSTUFF

HotStuff [32] is a partially synchronous consensus protocol that provides an optimal resiliency of $n = 3t + 1$. The HotStuff protocol is similar to PBFT but there are subtle differences which allow it to obtain a lower communication complexity for both its steady state and view change protocols (assuming the presence of threshold signatures). Looking ahead, these differences significantly change the way forensics is conducted if a safety violation happens.

5.1 Overview

We start with an overview of the protocol. For simplicity, we discuss a single-shot version of HotStuff. The protocol proceeds in a sequence of consecutive views where each view has a unique leader. Each view of HotStuff progresses as follows:

- **Pre-PREPARE.** The leader proposes a $\text{NewView}$ message containing a proposal $v$ along with the $\text{highQC}$ (the highest $\text{prepareQC}$ known to it) and sends it to all replicas.

- **PREPARE.** On receiving a $\text{NewView}$ message containing a valid value $v$ in a view $e$ and a $\text{highQC}$, a replica sends $\text{Prepare}$ for $v$ if it is safe to vote based on a locking mechanism (explained later). It sends this vote to the leader. The leader collects $2t + 1$ such votes to form an aggregate signature $\text{prepareQC}$ in view $e$. The leader sends the view $e$ $\text{prepareQC}$ to all replicas.

- **PRECOMMIT.** On receiving a $\text{prepareQC}$ in view $e$ containing message $v$, a replica updates its highest $\text{prepareQC}$ to $(v, e)$ and sends $\text{PRECOMMIT}$ to the leader. The leader collects $2t + 1$ such votes to form an aggregate signature $\text{precommitQC}$. The leader sends $\text{precommitQC}$ to all replicas.

- **COMMIT.** On receiving $\text{precommitQC}$ in view $e$ containing message $v$ from the leader, a replica locks on $(v, e)$ and sends $\text{Commit}$ to the leader. The leader collects $2t + 1$ such votes to form an aggregate signature $\text{commitQC}$. The safety and liveness of HotStuff when $f \leq t$ follows from the following:

**Uniqueness within a view.** Since replicas only vote once in each round, a $\text{commitQC}$ can be formed for only one value when $f \leq t$.

**Safety and liveness across views.** Safety across views is ensured using locks and the voting rule for a $\text{NewView}$ message. Whenever a replica outputs a value, at least $2t + 1$ other replicas are locked on the value in the view. Observe that compared to PBFT, there is no status certificate $M$ in the $\text{NewView}$ message to “unlock” a replica. Thus, a replica only votes for the value it is locked on. The only scenario in which it votes for a conflicting value $v'$ is if the leader includes a $\text{prepareQC}$ for $v'$ from a higher view in $\text{NewView}$ message. This indicates that at least $2t + 1$ replicas are not locked on $v$ in a higher view, and hence it should be safe to vote for it. The latter constraint of voting for $v'$ is not necessary for safety, but only for liveness of the protocol.

**Variants of HotStuff.** In this paper, we study three variants of HotStuff, identical for the purposes of consensus but provide varied forensic support. The distinction among them is only in the information carried in $\text{Prepare}$ message. For all three versions, the message contains the message type $\text{Prepare}$, the current view number $e$ and the proposed value $v$. In addition, $\text{Prepare}$ in HotStuff-view contains $\epsilon_{QC}$, the view number of the $\text{highQC}$ in the $\text{NewView}$

---

The description of HotStuff protocol is slightly different from the basic algorithm described in [32, Algorithm 2] to be consistent with the description of PBFT in §4.1.
The status certificate in PBFT provides us with the following:

- If two conflicting values are output in the same view, Byzantine behavior may not exist in the messages in subsequent views.
- A prepareQC on value \( v \) for \( e \) and knowing \( e_q \) from the value \( v \) from the message \( \text{prepareQC} \) for \( v \) is formed.
- If any of these replicas also held a lock \((v,e)\) (by voting for replica \( i \)'s output) then these replicas have output the culpable act of not respecting the voting rule.

The only remaining part is to ensure that this is indeed the first view where a higher conflicting prepareQC is formed. The way to prove this is also the key difference among three variants of HotStuff. For HotStuff-view, prepareQC contains \( e_q \), which explicitly states the view number of highQC in the NewView message they vote for. If \( e_q < e \), prepareQC provides an irrefutable proof for culpable behavior. For HotStuff-hash, the hash information contained in prepare provides the necessary link to the NewView message they vote, so once the linked NewView message is accessible, the prepareQC and NewView together serve as the proof for culpable behavior. However for HotStuff-null, even if we receive both prepareQC and NewView messages that are formed in the same view, no proof can be provided to show a connection between them. A Byzantine node vote for the first higher prepareQC can always refuse to provide the NewView message they receive.

Thus, to summarize, the red set of replicas in view \( e^p \) are a potential set of culpable nodes of size \( t + 1 \). The irrefutable proof to hold them culpable constitutes two parts, (1) the first prepareQC containing their signed prepare messages, and (2) a proof to show this is indeed the first view. In the next two subsections we will introduce the forensic protocols for HotStuff-view and HotStuff-hash to provide different forensic supports, and how it is impossible for HotStuff-null to provide forensic support.

### 5.3 Forensic Protocols for HotStuff-view and HotStuff-hash

Forensic protocol for HotStuff-view. Algorithm 2 describes the protocol to obtain forensic support atop HotStuff-view. A complete description of the general HotStuff protocol is also provided in Algorithm 6. Each replica keeps all received messages as transcript and maintains a set \( P \) containing all received prepareQC from PRE-commit messages and highQC from NewView messages (line 2). If a client observes outputs of two conflicting values in the same view, it can determine the culprits using the two \text{REPLY} messages (line 15). Otherwise, the client sends a request to all replicas for a possible proof between two output views \( e, e' \) for \( e < e' \) (line 12). Each replica looks through the set \( P \) for prepareQC formed in views \( e < e' \leq e \). If there exists a prepareQC whose value is different from the value \( v \) output in \( e \) and whose \( e_q \) is less than or equal to \( e \), it sends a reply with this prepareQC to the client (line 6). The client waits for a prepareQC (line 13) formed between two output views. For HotStuff-view, if it contains a different value from the first output value and an older view number \( e_q < e \), the intersection of this prepareQC and the commitQC from the REPLY message in the lower view proves at least \( t + 1 \) culprits (line 14).
**Algorithm 2** Forensic protocol for HotStuff-view

1. as a replica running HotStuff-view
2. \( P \leftarrow \text{all prepareQC in transcript} \)
3. \( Q \leftarrow \text{all NewView messages in transcript} \)
4. \( \text{upon receiving (REQUEST-Proof, e, v, e') from a client do} \)
5. \( \text{for } q_e \in P \text{ do} \)
6. \( \text{if } (q_c.v \neq v) \wedge (q_c.e \in (e, e')) \wedge (q_c.e_qc \leq e) \text{ then} \)
7. \( \text{send (Proof-across-view, qc) to the client} \)
8. \( \text{as a client} \)
9. \( \text{upon receiving two conflicting Reply messages do} \)
10. \( \text{if two messages are from different views then} \)
11. \( (\text{Reply, e, v, } \sigma) \leftarrow \text{the message from lower view} \)
12. \( e' \leftarrow \text{the number of message from higher view} \)
13. \( \text{broadcast (REQUEST-Proof, e, v, e')} \)
14. \( \text{wait for (Proof-across-view, qc) s.t.} \)
15. \( (1) e < q_c.e \leq e', \text{ and} \)
16. \( (2) (q_c.v \neq v) \wedge (q_c.e_qc \leq e) \)
17. \( \text{output } q_c.\sigma \cap \sigma \)
18. \( \text{else} \)
19. \( (\text{Reply, e, v, } \sigma) \leftarrow \text{first Reply message} \)
20. \( (\text{Reply, e', v', } \sigma') \leftarrow \text{second Reply message} \)
21. \( \text{output } \sigma \cap \sigma' \)

**Algorithm 3** Forensic protocol for HotStuff-hash

1. as a replica running HotStuff-hash
2. \( P \leftarrow \text{all prepareQC in transcript} \)
3. \( Q \leftarrow \text{all NewView messages in transcript} \)
4. \( \text{upon receiving (REQUEST-Proof, e, v, e') from a client do} \)
5. \( \text{for } q_e \in P \text{ do} \)
6. \( \text{if } (q_c.v \neq v) \wedge (q_c.e \in (e, e')) \text{ then} \)
7. \( \text{send (Proof-across-view, qc) to the client} \)
8. \( \text{as a client} \)
9. \( \text{upon receiving two conflicting Reply messages do} \)
10. \( \text{if two messages are from different views then} \)
11. \( (\text{Reply, e, v, } \sigma) \leftarrow \text{the message from lower view} \)
12. \( e' \leftarrow \text{the number of message from higher view} \)
13. \( \text{broadcast (REQUEST-Proof, e, v, e')} \)
14. \( \text{upon receiving (NewView, m) do} \)
15. \( \text{if } (m.v \neq v) \wedge (m.e \in (e, e')) \wedge (m.\text{highQC} \cdot e \leq e) \text{ then} \)
16. \( \text{NewView } \leftarrow \text{NewView} \cup \{m\} \)
17. \( \text{wait for (Proof-across-view, qc) s.t.} \)
18. \( (1) e < q_c.e \leq e', \text{ and} \)
19. \( (2) (q_c.v \neq v) \wedge (\exists m \in \text{NewView}, \text{Hash}(m) = q_c.\text{hash}) \)
20. \( \text{output } q_c.\sigma \cap \sigma \)
21. \( \text{else} \)
22. \( (\text{Reply, e, v, } \sigma) \leftarrow \text{first Reply message} \)
23. \( (\text{Reply, e', v', } \sigma') \leftarrow \text{second Reply message} \)
24. \( \text{output } \sigma \cap \sigma' \)

**Forensic protocol for HotStuff-hash.** Algorithm 3 describes the protocol to obtain forensic support atop HotStuff-hash, which is similar to the protocol for HotStuff-view. For replicas running HotStuff-hash, besides \( P \), they also maintains the set \( Q \) for received \( \text{NewView} \) messages (line 3). When receiving a forensic request from clients, replicas look through \( P \) for \( \text{prepareQC} \) formed in views \( e < e' \leq e' \) and send all \( \text{prepareQC} \) whose values are different from the value \( v \) to the client (line 7). Besides, they also look through \( Q \) for a new \( \text{NewView} \) message formed in views \( e < e' \leq e' \) and send all \( \text{NewView} \) proposing a value different from \( v \) containing a highQC with view number \( e \) (line 10). For HotStuff-hash, when receiving such a \( \text{NewView} \) for different values, the message will be stored temporarily by the client until a \( \text{prepareQC} \) for the \( \text{NewView} \) message with a matching hash is received. The \( \text{NewView} \) and the \( \text{prepareQC} \) together form the desired proof; the intersection of the \( \text{prepareQC} \) and the \( \text{commitQC} \) provides at least \( t+1 \) culprits.

**Forensic proofs.** The following theorems characterize the forensic support capability of HotStuff-view and HotStuff-hash. As long as \( m \leq 2t \), HotStuff-view can achieve the best possible forensic support (i.e., \( k = 1 \) and \( d = t+1 \)). HotStuff-hash can achieve a medium forensic support (i.e., \( k = t+1 \) and \( d = t+1 \)). Conversely, if \( m > 2t \) then no forensic support is possible for both protocols (i.e., \( k = n-f \) and \( d = 0 \)).

**Theorem 5.1.** With \( n = 3t+1 \), when \( f > t \), if two honest replicas output conflicting values, HotStuff-view provides \( (2t, 1, t+1) \)-forensic support. Further \( (2t+1, n-f, d) \)-forensic support is impossible with \( d > 0 \).

**Proof.** We prove the forward part of the theorem below. The proof of converse (impossibility) is the same as §C.1. Suppose two conflicting values \( v \), \( v' \) are output in views \( e \), \( e' \) respectively.

**Case** \( e = e' \).

Culpability. The \( \text{commitQC} \) of \( v \) and \( \text{commitQC} \) of \( v' \) intersect in \( t+1 \) replicas. These \( t+1 \) replicas should be Byzantine since the protocol requires a replica to vote for at most one value in a view. Witnesses. Client can obtain a proof based on the two \( \text{Reply} \) messages, so additional witnesses are not necessary in this case.

**Case** \( e \neq e' \).

Culpability. If \( e \neq e' \), then WLOG, suppose \( e < e' \). Since \( v \) is output in view \( e \), it must be the case that \( 2t+1 \) replicas are locked on \( (v, e) \) at the end of view \( e \). Now consider the first view \( e < e' \leq e' \) in which a higher lock \( (v'', e'') \) is formed where \( v'' \neq v \) (possibly \( v'' = v' \)). Such a view must exist since \( e' \) is output in view \( e' > e \) and a lock will be formed in at least view \( e' \). For a lock to be formed, a higher \( \text{prepareQC} \) must be formed too. Consider the first view \( e < e'' \leq e' \) in which the corresponding \( \text{prepareQC} \) for \( v'' \) is formed. The leader in \( e'' \) broadcasts the \( \text{NewView} \) message containing a highQC on \( (v'', e'') \). Since this is the first time a higher \( \text{prepareQC} \) is formed and there is no higher \( \text{prepareQC} \) for \( v'' \) formed between view \( e \) and \( e'' \), we have \( e'' \leq e' \). The formation of the higher \( \text{prepareQC} \) indicates that \( 2t+1 \) replicas received the \( \text{NewView} \) message proposing \( v'' \) with highQC on \( (v'', e'') \) and consider it a valid proposal, i.e., the view number \( e'' \) is larger than their locks because the value is different.
Recall that the output value \( v \) indicates \( 2t+1 \) replicas are locked on \( (v,e) \) at the end of view \( e \). In this case, the \( 2t+1 \) votes in \( \text{prepareQC} \) in view \( e \) intersect with the \( 2t+1 \) votes in \( \text{commitQC} \) in view \( e \) at \( t+1 \) Byzantine replicas. These replicas should be Byzantine because they were locked on the value \( v \) in view \( e \) and vote for a value \( v'' \neq v \) in a higher view \( e' \) when the \text{NewView} message contained a highQC from a view \( e'' \leq e \). Thus, they have violated the voting rule.

Witnesses. Client can obtain a proof by storing a \text{prepareQC} formed between \( e \) and \( e' \). whose value is different from \( v \) and whose \( e_{\text{rep}} \leq e \). So only one witness is needed \((k = 1)\), the \text{prepareQC} and the first \text{commitQC} act as the irrefutable proof.

**Theorem 5.2.** With \( n = 3t + 1 \), when \( f > t \), if two honest replicas output conflicting values, HotStuff-hash provides \((2t + 1, t + 1, t + 1)\)-forensic support. Further \((2t + 1, n - f, d)\)-forensic support is impossible with \( d > 0 \).

**Proof.** We prove the forward part of the theorem below. The proof of converse (impossibility) is the same as \S C.1. Suppose two conflicting values \( v, v' \) are output in views \( e, e' \) respectively.

**Case** \( e = e' \). Same as Theorem 5.1.

**Case** \( e \neq e' \). Culpability. Same as Theorem 5.1.

Witnesses. Since \text{prepareQC} of HotStuff-hash only has the hash of highQC, the irrefutable proof contains the \text{NewView} message that includes the highQC and the corresponding \text{prepareQC} with the matching hash \( \text{Hash}(\text{highQC}) \). The client may need to store all \text{NewView} messages between \( e \) and \( e' \) whose value is different from \( v \) and the whose highQC is formed in \( e_{\text{rep}} \leq e \), until receiving a \text{prepareQC} for some \text{NewView} message with a matching hash. In the best case, some replica sends both the \text{NewView} message and the corresponding \text{prepareQC} so the client only needs to store \( k = 1 \) replica’s transcript. In the worst case, we can prove that any \( t + 1 \) messages of transcript are enough to get the proof. Consider the honest replicas who receive the first \text{prepareQC} and the \text{NewView} message. \( 2t + 1 \) replicas have access to the \text{prepareQC} and \( 2t + 1 \) replicas have access to the \text{NewView} message. Among them at least \( t + 1 \) replicas have access to both messages, and we assume they are all Byzantine. Then at least \( t \) honest replicas have the \text{prepareQC} and at least \( t \) honest replicas have the \text{NewView} message. The total number of honest replicas \( n - f \leq 2t \). Thus among any \( t + 1 \) honest replicas, at least one have \text{NewView} message and at least one have \text{prepareQC}. Therefore, \( t + 1 \) transcripts from honest replicas ensure the access of both \text{NewView} message and \text{prepareQC} and thus guarantee the irrefutable proof.

**Communication complexity.** In line 13 of Algorithm 2, the client needs to receive one message from \( k = 1 \) replica and the message size is \(|v| + |\text{sig}| \) where \(|v| \) and \(|\text{sig}| \) stand for the size of a value and an aggregate signature. Therefore the complexity of the client receiving messages is \( O(|v| + |\text{sig}|) \) for HotStuff-view. As for HotStuff-hash, theorem 5.2 shows that in the worst case, the client needs to receive messages from \( k = t + 1 \) replicas. Each of those replicas sends one message of size \( O(|v| + |\text{sig}| + |\text{hash}|) \) where \(|\text{hash}| \) stands for the size of a hash value. Therefore the complexity of the client receiving messages is \( O(n(|v| + |\text{sig}| + |\text{hash}|)) \) for HotStuff-hash.

### 5.4 Impossibility for HotStuff-null

Compared to the other two variants, in HotStuff-null, \text{prepare} message and \text{prepareQC} are not linked to the \text{NewView} message. We show that this lack of information is sufficient to guarantee impossibility of forensics.

**Intuition.** When \( f = t + 1 \), from the forensic protocols of HotStuff-view and HotStuff-hash, we know that given across-view \text{commitQC} \(_1\) and \text{commitQC} \(_2\) (ordered by view) and the first \text{prepareQC} higher than \text{commitQC} \(_1\), the intersection of \text{prepareQC} and \text{commitQC} \(_2\) contains at least \( d = t + 1 \) Byzantine replicas. The intersection argument remains true for HotStuff-null, however, it is impossible for a client to decide whether \text{prepareQC} is the first one only with the transcripts sent by \( 2t \) honest replicas (when \( f = t + 1 \)). In an execution where there are two \text{prepareQC} in view \( e' \) and \( e'' \) respectively \((e' \neq e'')\), the Byzantine replicas (say, set \( P \)) may not respond with the \text{prepareQC} in \( e' \). The lack of information disallows a client from separating this world from another world \( P \) is indeed honest and sharing all the information available to them. We formalize this intuition in the theorem below.

**Theorem 5.3.** With \( n = 3t + 1 \), when \( f > t \), if two honest replicas output conflicting values, \((t + 1, 2t, d)\)-forensic support is impossible with \( d > 1 \) for HotStuff-null. Further, \((t + 2, n - f, d)\)-forensic support is impossible with \( d > 0 \).

The theorem is proved in \S C.2.

### 6 FORENSIC SUPPORT FOR ALGORAND

Algorand [14] is a synchronous consensus protocol which tolerates up to one-third fraction of Byzantine users. At its core, it uses a BFT protocol from [13, 24]. However, Algorand runs the protocol by selecting a small set of replicas, referred to as the committee, thereby achieving consensus with sub-quadratic communication. The protocol is also player replaceable, i.e., different steps of the protocol have different committees, thus tolerating an adaptive adversary. Each replica uses cryptographic self-selection using a verifiable random function (VRF) [25] to privately decide whether it is selected in a committee. The VRF is known only to the replica until it has sent a message to other parties thus revealing and proving its selection in the committee. In this section, we present an overview of the BFT protocol and then show why it is impossible to achieve forensic support for this protocol.

#### 6.1 Overview

We start with an overview of the single-shot version of Algorand [14]. The protocol assumes synchronous communication, where messages are delivered within a known bounded time. The protocol proceeds in consecutive steps, each of which lasts for a fixed amount of time that guarantees message delivery. Each step has a self-selected committee, and a replica can compute its VRF, a value that decides whether it is selected in the committee, and it is known only by the replica itself until it has sent the value. All messages sent by a committee member is accompanied by a VRF thus allowing other replicas to verify its inclusion in the committee.
Parameters such as committee size $k$ are chosen such that the number of honest parties in the committee is greater than a threshold $t_H \geq 2k/3$ with overwhelming probability.

The BFT protocol is divided into two sub-protocols: Graded Consensus and BBA*. In Graded Consensus, which forms the first three steps of the protocol, each replica $r$ inputs its value $v_r$. Each replica $r$ outputs a tuple containing a value $v_{r}^{\text{out}}$ and a grade $g_r$. In an execution where all replicas start with the same input $v$, $v_{r}^{\text{out}} = v$ and $g_r = 2$ for all replicas $r$. The replicas then enter the next sub-protocol, denoted BBA*. If $g_r = 2$, replica $r$ inputs value $b_r = 0$, otherwise it inputs $b_r = 1$. At the end of BBA*, the replicas agree on the tuple $(0, v)$ or $(1, v_L)$.

The BBA* sub-protocol also uses a random coin; for simplicity, we assume access to an ideal global random coin. Our forensic analysis in the next section only depends on BBA* and thus, we only provide an overview for BBA* here.

The protocol proceeds in the following steps,

- **Steps 1-3** are Graded Consensus. At the end of Graded Consensus, each replica inputs a value $v_r$ and a binary value $b_r$ to BBA*.
- **Step 4.** Each replica in the committee broadcasts its vote for $(b_r, v_r)$ along with its VRF.
- **Step $s$ ($s \geq 5, s \equiv 2 \mod 3$) is the Coin-Fixed-To-0 step of BBA*.** In this step, a replica checks Ending Condition 0: if it has received $\geq t_H$ valid votes on $(0, v)$ from the previous step, where $b = 0$, it outputs $(b, v)$ and ends its execution. Otherwise, it updates $b_r$ as follows:
  
  $$b_r \left\{ \begin{array}{ll} 1, & \text{if } \geq t_H \text{ votes on } b = 1 \\ 0, & \text{otherwise} \end{array} \right.$$  
  
  If the replica is in the committee based on its VRF, it broadcasts its vote for $(b_r, v_r)$ along with its VRF.
- **Step $s$ ($s \geq 6, s \equiv 0 \mod 3$).** Symmetric to the previous step but for bit 1 instead of 0. Also, the votes need not be for the same $v$ in the ending condition.
- **Steps $s$ ($s \geq 7, s \equiv 1 \mod 3$) is the Coin-Genuinely-Flipped step of BBA*.** In this step, it updates its variable $b_r$ as follows:
  
  $$b_r \left\{ \begin{array}{ll} 0, & \text{if } \geq t_H \text{ votes on } b = 0 \\ 1, & \text{if } \geq t_H \text{ votes on } b = 1 \\ \text{random coin of step } s, & \text{otherwise} \end{array} \right.$$  
  
  If the replica is in the committee based on its VRF, it broadcasts its vote for $(b_r, v_r)$ along with the VRF.

**Safety of BBA* within a step.** If all honest replicas reach an agreement before any step, the agreement will hold after the step. If the agreement is on binary value 0 (1 resp.) then the opposite Ending Condition 1 (0 resp.) will not be satisfied during the step. This is because synchronous communication ensures the delivery of at least $t_H$ votes on the agreed value and there are not enough malicious votes on the other value.

**Safety of BBA* across steps.** For the step Coin-Fixed-To-0 (1 resp.), if any honest replica ends due to Ending Condition 0 (1 resp.), all honest replicas will agree on binary value 0 and value $v$ (1 and $v_L$ resp.) at the end of the step, because there could only be less than $t_H$ votes on binary value 1 (0 resp.). Hence, together with safety within a step, binary value 1 and value $v_L$ (0 and $v$ resp.) will never be output.

### 6.2 Impossibility of Forensics

When the Byzantine fraction in the system is greater than one-third, with constant probability, a randomly chosen committee of size $\kappa < n$ will have $t_H < 2k/3$. In such a situation, we can have a safety violation. Observe that since only $\kappa < n$ committee members send messages in a round, the number of culpable replicas may be bounded by $O(\kappa)$. However, we will show an execution where no Byzantine replica can be held culpable.

**Intuition.** The safety condition for BBA* relies on the following:

If some honest replica commits to a value $b$, say $b = 0$, in a step and terminates, then all honest replicas will set $b = 0$ as their local value. In all subsequent steps, there will be sufficient (> $2/3$ fraction) votes for $b = 0$ due to which replicas will never set their local value $b = 1$. Thus, independently of what Byzantine replicas send during the protocol execution, honest replicas will only commit on $b = 0$. On the other hand, if replicas do not receive $> 2/3$ fraction of votes for $b = 0$, they may switch their local value to $b = 1$ in the Coin-Fixed-To-1 or Coin-Genuinely-Flipped step. This can result in a safety violation. When the Byzantine fraction is greater than one-third, after some replicas have committed 0, the Byzantine replicas can achieve the above condition by selectively not sending votes to other replicas (say set $Q$), thereby making them switch their local value to $b = 1$. There is no way for an external client to distinguish this world from another world where the set Q is Byzantine and states that it did not receive these votes. We formalize this intuition in the theorem below. Observe that our arguments work for the BBA* protocol with or without player-replaceability.

**Theorem 6.1.** When the Byzantine fraction exceeds 1/3, if two honest replicas output conflicting values, (1+1, 2t, d)-forensic support is impossible with d > 0 for Algorand.

The theorem is proved in §C.3.

### 7 LIBRABFT AND DIEM

In this paper, we have focused on forensics for single-shot consensus. Chained BFT protocols are natural candidates for consensus on a sequence with applications to blockchains. LibraBFT is a chained version of Hotstuff and is the core consensus protocol in Diem, a new cryptocurrency supported by Facebook [5]. In this section, we show that LibraBFT has the strongest forensic support possible (as in Hotstuff-view). Further, we implement the corresponding forensic analysis protocol as a module on top of an open source Diem client. We highlight the system innovations of our implementation and associated forensic dashboard; this has served as a reference implementation presently under active consideration for deployment (due to anonymity imperatives we are unable to document this more concretely).

Diem blockchain. Diem Blockchain uses LibraBFT [30], a variant of HotStuff protocol for consensus. In LibraBFT, the replicas are called validators, who receive transactions from clients and propose blocks of transactions in a sequence of rounds.
LibraBFT forensics. The culpability analysis for LibraBFT is similar to Theorem 5.2. However, for the witnesses, the blockchain property of LibraBFT makes sure that any replica (validator) has access to the full blockchain and thus provides (n − 2, 1, t + 1)-forensic support. The formal result is below (proof in §C).

**Theorem 7.1.** For n = 3t + 1, when f > t, if two honest replicas output conflicting blocks, LibraBFT provides (n − 2, 1, t + 1)-forensic support.

The aforementioned three variants of HotStuff in §5 are described under the VBA setting to reach consensus on a single value, and the value can be directly included in the vote message (cf. Table 4, ν is contained in the Prepare message). In this setting, once a replica receives the commitQC for the value, it will output the value and send a reply to the client, even if the commitQC is the only message it receives in the current view so far. So when two honest replicas output conflicting values, it is possible that the client receives only the commit messages and extra communication is needed. And when m > 2t, Byzantine replicas are able to form QCs by themselves so that no other honest replicas can get access to the first prepareQC. Thus the bound on m for HotStuff-view and HotStuff-hash is 2t.

However, the setting is slightly different in practice, when the value ν is no longer a single value, but actually a block with more fields and a list of transactions/commands, as in LibraBFT. In single-shot consensus, a block includes the transactions (value) and the highQC. In this case, the block is too heavy to include directly in a vote message, so the replicas add the hash of the block to the vote message (see Table 4, Hash(b = (ν, highQC)) is contained in the Prepare message). And since only the NewView message has the block’s preimage, replicas cannot vote/output until receiving the original blocks. Thus when two honest replicas output conflicting values, the client can obtain the full blockchain from one of them (k = 1) and all prepareQC are part of the blocks. In this case, even if m > 2t the client can still enjoy non-trivial forensic support.

**Forensic module.** Our prototype consists of two components, a database Forensic Storage used to store quorum certificates received by validators, which can be accessed by clients through JSON-RPC requests or consensus API; and an independent Detector run by clients to analyze the forensic information.

- **Forensic Storage** maintains a map from the view number to quorum certificates and its persistent storage. It is responsible for storing forensic information and allows access by other components, including clients (via JSON-RPC requests or consensus API).

- **Detector** is run by clients manually to send requests periodically to connected validators. It collates information received from validators, using it as the input to the forensic analysis protocol.

**Testing using Twins [6].** To test the correctness of forensic protocols, we build a testbed to simulate Byzantine attacks and construct different types of safety violations. Ideally, for modularity purposes, our testbed should not require us to modify the underlying consensus protocol to obtain Byzantine behavior. We leverage Twins [6], an approach to emulate Byzantine behaviors by running two instances of a node (i.e. replica) with the same identity. Consider a simple example setting with four nodes (denoted by node0 ~ 3), where node0 and node1 are Byzantine so they have twins called twin0 and twin1. The network is split into two partitions, the first partition P1 includes nodes {node0, node1, node2} and the second partition P2 includes nodes {twin0, twin1, node3}. Nodes in one partition can only receive the messages sent from the same partition. The double voting attack can be simulated when Byzantine leader proposes different valid blocks in the same view, and within each partition, all nodes will vote for the proposed block. The network partition is used to drop all messages sent from a set of nodes. However, it can only help construct the safety violation within the view. To construct more complicated attacks, we further improve the framework and introduce another operation called “detailed drop”, which drops selected messages with specific types.

**Visualization.** The Detector accepts the registration of different views to get notified once the data is updated. We built a dashboard to display the information received by the detector and the analysis results output by the forensic protocol. Figure 4 shows a snapshot of the dashboard which displays information about the network topology, hashes of latest blocks received at different validators, conflicting blocks, detected culprit keys and raw logs. Interactions with end-users, including Diem core-devs, has guided our design of the dashboard.

**8 CONCLUSION**

In this paper, we have embarked on a systematic study of the forensic properties of BFT protocols, focusing on 4 canonical examples: PBFT (classical), Hotstuff and VABA (state-of-the-art protocols on partially synchronous and asynchronous network settings) and Algorand (popular protocol that is adaptable to proof of stake...
blockchains). Our results show that minor variations in the BFT protocols can have outsized impact on their forensic support.

We exactly characterize the forensic support of each protocol, parameterized by the triplet \((m, k, d)\). The forensic support characterizations are remarkably similar across the protocols: if any non-trivial support is possible (i.e., at least one culpable replica can be implicated; \(d > 0\)), then the largest possible forensic support, \((2t, 1, t + 1)\), is also possible; the one exception to this result is the Hotstuff-hash variant. Although the proof of forensic support is conducted for each protocol and its variant individually, we observe common trends:

- For each of the protocols with strong forensic support, as a part of the protocol execution, there exist witnesses who hold signed messages from Byzantine parties indicating that they have not followed some rule in the protocol.
- On the other hand, for protocols with no forensic support, the Byzantine parties are able to break safety without leaving any evidence, although the mechanism to achieve this is different for each of PBFT-MAC, Algorand, and HotStuff-null. With PBFT-MAC, Byzantine parties are able to construct arbitrary transcripts due to the absence of signatures. Hence, message transcripts cannot be used as evidence. With Algorand, they can utilize a rule which relies on the absence of messages (under synchrony) to set an incorrect protocol state without leaving a trail. With HotStuff-null, due to the lack of links between messages across views, Byzantine parties can present fake message transcripts and thus, pretend to be honest.

Conceptually, the burning question is whether these common ingredients can be stitched together to form an overarching theory of forensic support for abstract families of secure BFT protocols: First, from an impossibility standpoint, is there a relationship between the need to use synchrony or the absence of PKI in a protocol and absence of forensic support? Second, for the positive results, can one argue strong forensic support for an “information-complete” variant of any BFT protocol? This is an active area of research.

From a practical standpoint, forensic analysis for existing blockchain protocols is of great interest; our forensic protocol for LibraBFT and its reference implementation has made strong inroads towards practical deployment. However, one shortcoming of the approach in this paper is that forensic analysis is conducted only upon fatal safety breaches. It is of great interest to conduct forensics with other forms of attacks: liveness attacks, censorship, small number of misbehaving replicas that impact performance. We note that liveness attacks do not afford, at the outset, undeniable complicity of malicious replicas and an important research direction is formalizing weaker notions of culpability proofs (perhaps including assumptions involving external forms of trust). This is an active area of research.

9 ACKNOWLEDGEMENT

This research was partly supported by US Army Research Office Grant W911NF-18-1-0332, National Science Foundation CCF-1705007 and the XDC network. Kartik Nayak was supported in part by Novi and VMware gift research grant.

We thank the Diem team for implementation advice and Jovan Komatovic for helpful discussions about a forensic attack on HotStuff.

REFERENCES

[1] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. 2019. Sync hotstuff: Simple and practical synchronous state machine replication. IACR Cryptology ePrint Archive 2019 (2019), 270.

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically optimal validated asynchronous byzantine agreement. In Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. 337–346.
A VABA HAS STRONG FORENSIC SUPPORT

Validated Asynchronous Byzantine Agreement (VABA) is a state-of-the-art protocol [2] in the asynchronous setting with asymptotically optimal $O(n^2)$ communication complexity and expected $O(1)$ latency for $n \geq 3t + 1$.

A.1 Overview

At a high-level, the VABA protocol adapts HotStuff to the asynchronous setting. There are three phases in the protocol:

- **Proposal promotion.** In this stage, each of the $n$ replicas run $n$ parallel HotStuff-like instances, where replica $i$ acts as the leader within instance $i$.

Algorithm 4 Forensic protocol for VABA

1. as a replica running VABA
2. for $e \geq 1$ initialize:
3. for $j \in [n]$ do
4. ledger[e][j] $\leftarrow \{\}$
5. coin[e] $\leftarrow \{\}$
6. upon receiving $(i, \text{NEWVIEW}, e, v, L)$ in view $e$ in Proposal-Promotion instance $i$ do
7. $(e', v', \sigma, e_qc) \leftarrow L$ Note that $L$ has selectors $e, v, \sigma, e_qc$
8. ledger[e'][i] $\leftarrow$ ledger[e][i] $\cup \{(o, \sigma, e_qc)\}$
9. upon receiving $(i, \text{PREPARE}, e, v, \sigma, e_qc)$ in view $e$ in Proposal-Promotion instance $i$ do
10. ledger[e][i] $\leftarrow$ ledger[e][i] $\cup \{(o, \sigma, e_qc)\}$
11. if Leader[e] is elected then
12. discard ledger[e][i] for $j \neq \text{Leader}[e]$
13. coin[e] $\leftarrow$ inputs to threshold-coin for electing Leader[e]
14. upon receiving $(\text{VIEWCHANGE}, e, \text{prepareQC}, \text{precommitQC}, \text{commitQC})$ in view $e$ do
15. $(e', v', \sigma, e_qc) \leftarrow \text{prepareQC} >$ Note that $\text{prepareQC}$ has selectors $e, v, \sigma, e_qc$
16. ledger[e'][i][\text{Leader}[e']][i] $\leftarrow$ ledger[e][i][\text{Leader}[e']][i] $\cup \{(o', \sigma, e_qc)\}$
17. upon receiving $(\text{REQUEST-PROOF-OF-LEADER}, e, e')$ from a client do
18. for all $e \leq e' \leq e'$ do
19. send $(\text{PROOF-OF-LEADER}, e, e', \text{Leader}[e'], \text{coin}[e'])$
20. to client
21. upon receiving $(\text{REQUEST-PROOF-OF-LEADER}, e, v, e')$ with a collection of LeaderMsg from a client do
22. for all $e < e' \leq e'$ do
23. if Leader[e'] is not elected yet then
24. $(\text{PROOF-OF-LEADER}, e, e', \text{leader}, \text{coin}) \leftarrow$ LeaderMsg of view $e'$
25. check leader is the leader generated by coin in view $e'$(otherwise don’t reply to the client)
26. Leader[e'] $\leftarrow$ leader
27. for $qc \in \text{ledger}[e'][\text{Leader}[e']][i]$ do
28. if $(qc, v \neq o) \land (qc, e_qc \leq e)$ then
29. send $(\text{PROOF-Across-VIEW}, e, e', \text{Leader}[e'], qc)$ to the client
Algorithm 4 Forensic protocol for VABA

as a client
upon receiving two conflicting REPLY messages do
  $e \leftarrow$ the view number of REPLY from lower view
  $e' \leftarrow$ the view number of REPLY from higher view
for all $e \leq e' \leq e'$ initialize:
  Leader[$e'$] $\leftarrow \emptyset$
  LeaderMsg[$e'$] $\leftarrow \emptyset$
send (REQUEST-Proof-of-Leader, $e, e'$) to the replica of REPLY message from higher view
for all $e \leq e' \leq e'$ do
  wait for (PROOF-OF-LEADER, $e', $leader, coin) s.t.
  leader is the leader generated by coin in view $e'$ (otherwise the REPLY message is not considered valid)
  Leader[$e'$] $\leftarrow$ leader
  LeaderMsg[$e'$] $\leftarrow$
  (PROOF-OF-LEADER, $e'$, leader, coin)
if the two REPLY messages are from different views
  ($i, \text{REPLY}, e, v, \sigma$) $\leftarrow$ the message from lower view
  ($i', \text{REPLY}, e', v', \sigma'$) $\leftarrow$ the message from higher view
  check $i = \text{Leader}[e]$ and $i' = \text{Leader}[e']$ (otherwise the REPLY message is not considered valid)
  broadcast (REQUEST-PROOF, $e, v, \sigma, e'$) with LeaderMsg[$e'$] for all $e < e' \leq e'$
for (PROOF-ACROSS-VIEW, $e^0$, leader, qc) s.t.
  (1) $e < e^0 \leq e'$, and
  (2) $(qc.v \neq v) \land (qc.v.qc \leq e)$, and
  (3) $\text{leader} = \text{Leader}[e^0]$
output $qc.\sigma \cap \sigma$
else
  ($i', \text{REPLY}, e, v, \sigma$) $\leftarrow$ first REPLY message
  ($i', \text{REPLY}, e', v', \sigma'$) $\leftarrow$ second REPLY message
  check $i' = i' = \text{Leader}[e]$ (otherwise the REPLY message is not considered valid)
output $\sigma \cap \sigma'$

- Leader election. After finishing the previous stage, replicas run a leader election protocol using a threshold-coin primitive [9] to randomly elect the leader of this view, denoted as Leader[$e$] where $e$ is a view number. At the end of the view election phase, replicas adopt the “progress” from Leader[$e$]’s proposal promotion instance, and discard values from other instances.

- View change. Replicas broadcast quorum certificates from Leader[$e$]’s proposal promotion instance and update local variables and/or output value accordingly.

Within a proposal promotion stage, the guarantees provided are the same as that of HotStuff, and hence we do not repeat it here. The leader election phase elects a unique leader at random – this stage guarantees (i) with $\geq 2/3$ probability, an honest leader is elected, and (ii) an adaptive adversary cannot stall progress (since a leader is elected in hindsight). Finally, in the view-change phase, every replica broadcasts the elected leader’s quorum certificates to all replicas.

A.2 Forensic Support for VABA

The key difference between forensic support for HotStuff and VABA is the presence of the leader election stage – every replica/client needs to know which replica was elected as the leader in each view. Importantly, the threshold-coin primitive ensures that there is a unique leader elected for each view. Thus, the forensic analysis boils down to performing an analysis similar to the HotStuff protocol, except that the leader of a view is described by the leader election phase.

We present the full forensic protocol in Algorithm 4 for completeness. We make the following changes to VABA:

- Storing information for forensics. Each replica maintains a list of leaders for all instances, containing all received prepareQC from PREPARE messages, VIEWCHANGE messages, and VIEWCHANGE messages (lines 8,10,16). When the leader of a view is elected, a replica keeps the leader from the leader’s instance and discards others (line 12). A replica also stores the random coins from the leader election phase for client verification (line 13).

- Bringing proposal promotion closer to HotStuff-view. There are minor differences in the proposal promotion phase of VABA [2] to the description in HotStuff (§5). We make this phase similar to that in the description of our HotStuff protocol with forensic support. In particular: (i) the LOCK variable stores both the view number and the value (denoted by LOCK.$v$ and LOCK.$v,qc$), (ii) the voting rule in a proposal promotion phase is: vote if KEY has view and value equal to LOCK, except when KEY’s view is strictly higher than LOCK.$v$. (iii) assume a replica’s own VIEWCHANGE message arrives first so that others’ VIEWCHANGE messages do not overwrite local variables KEY and LOCK, and (iv) add $\epsilon_qc$ into PREPARE.

A client first verifies leader election (lines 36-40). Then, it follows steps similar to the HotStuff forensic protocol (lines 41-52) except that there are added checks pertaining to leader elections (lines 44,46,51).

We prove the forensic support in Theorem A.1.

Theorem A.1. For $n = 3t + 1$, when $f > t$, if two honest replicas output conflicting values, VABA protocol provides $(2t + 1, t + 1)$-forensic support. Further $(2t + 1, n - f, d)$-forensic support is impossible with $d > 0$.

Proof. We prove the forward part of the theorem below. The proof of converse (impossibility) is the same as §C.1.

The leader of a view is determined by the threshold coin-tossing primitive threshold-coin and Byzantine replicas cannot forge the result of a leader election by the robustness property of the threshold coin. Suppose two conflicting outputs happen in view $e, e'$ with $e \leq e'$. The replica who outputs in view $e'$ has access to the proof of leader election of all views $\leq e'$. Therefore, a client can verify the leader election when it receives messages from this replica. Even if other replicas have not received messages corresponding to the elections in views $\leq e'$, the client can send the proof of leader to them. The remaining forensic support proof follows from Theorem 5.1 in a straightforward manner, where any witness will
receive the proof of leader from the client (if leader is not elected) and send the proof of culprits to the client. □

Communication complexity. The client needs to first receive all leader election results from view to view $e'$, and each result is of size $|\text{coin}|$ (the size of the coin in the threshold-coin primitive). Then, the client shares leader election results with all replicas. This step incurs receiving message complexity $O(l|\text{coin}|)$ where $l = e' - e$. Next, the client needs to receive one message from $k = 1$ replica and the message size is $(|v| + |\text{sig}|)$. Therefore the complexity for the client receiving messages is $O(|v| + |\text{sig}| + l|\text{coin}|)$. However, the procedure of sharing leader election is irrelevant to forensic support, and we could assign it to replicas. (This procedure is included in the forensic protocol because we do not want to change the consensus protocol itself.) In that case, the client needs to receive just one leader election result, so the receiving message complexity is $O(|v| + |\text{sig}| + |\text{coin}|)$.

## B IMPOSSIBILITY OF FORENSIC SUPPORT FOR $n = 2t + 1$

A validated Byzantine agreement protocol allows replicas to obtain agreement, validity, and termination so far as the actual number of faults $f \leq t$ where $t$ is a Byzantine threshold set by the consensus protocol. A protocol that also provides forensic support with parameters $m$ and $d$ allows the detection of $d$ Byzantine replicas when $m$ out of $n$ replicas are Byzantine faulty. In particular, in §4 and §5, we observed that when $t = [n/3]$, $m = 2t$, and $k = 1$, we can obtain $(2t, 1, d)$-forensic support for $d = t + 1$. This section presents the limits on the number of Byzantine replicas detected ($d$), given the total number of Byzantine faulty replicas available in the system ($m$). In particular, we show that if the total number of Byzantine faulty replicas are too high, in case of a disagreement, the number of corrupt (Byzantine) replicas that can be deemed undeniably culpable will be too few.

**Intuition.** To gain intuition, let us consider a specific setting with $n = 2t + 1$, $m = n - t = t + 1$, and $d > 1$. Thus, such a protocol provides us with agreement, validity, and termination if the Byzantine replicas are in a minority. If they are in the majority, the protocol transcript provides undeniable guilt of more than one Byzantine fault. We show that such a protocol does not exist. Why? Suppose we split the replicas into three groups $P$, $Q$, and $R$ of sizes $t$, $t$, and $1$ respectively. First, observe that any protocol cannot expect Byzantine replicas to participate in satisfying agreement, validity, and termination. Hence, if the replicas in $Q$ are Byzantine, replicas in $P \cup R$ may not receive any messages from $Q$. However, if, in addition, the replica $R$ is also corrupt, then $R \cup Q$ can separately simulate another world where $P$ are Byzantine and not sending messages, and $Q \cup R$ output a different value. Even if an external client obtains a transcript of the entire protocol execution (i.e., transcripts of $k = n - f$ honest replicas and $f$ Byzantine replicas), the only replica that is undeniably culpable is $R$ since it participated in both worlds. For all other replicas, neither $P$ nor $Q$ have sufficient information to prove the other set’s culpability. Thus, an external client will not be able to detect more than one Byzantine fault correctly. Our lower bound generalizes this intuition to hold for $n > 2t$, $m = n - t$, $k = n - f$, and $d > n - 2t$.

**Theorem B.1.** For any validated Byzantine agreement protocol with $t < n/2$, when $f > t$, if two honest replicas output conflicting values, $(n - t, n - f, d)$-forensic support is impossible with $d > n - 2t$.

**Proof.** Suppose there exists a protocol that achieves agreement, validity, termination, and forensic support with parameters $n$, $t < n/2$, $m = n - t$, $k = n - f$ and $d > n - 2t$. Through a sequence of worlds, and through an indistinguishability argument we will show the existence of a world where a client incorrectly holds at least one honest replica as culpable. Consider the replicas to be split into three groups $P$, $Q$, and $R$ with $t$, $t$, and $n - 2t$ replicas respectively. We consider the following sequence of worlds:

**World 1.** $[t$ Byzantine faults, satisfying agreement, validity, and termination]

Setup. Replicas in $P$ and $R$ are honest while replicas in $Q$ have crashed. $P$ and $R$ start with a single externally valid input $v_1$. All messages between honest replicas arrive instantaneously.

Output. Since there are $|Q| = t$ faults, due to agreement, validity and termination properties, replicas in $P$ and $R$ output $v_1$. Suppose replicas in $P$ and $R$ together produce a transcript $T_1$ of all the messages they have received.

**World 2.** $[t$ Byzantine faults, satisfying agreement, validity, and termination]

Setup. Replicas in $Q$ and $R$ are honest while replicas in $P$ have crashed. $Q$ and $R$ start with an externally valid input $v_2$. All messages between honest replicas arrive instantaneously.

Output. Since $t$ replicas are Byzantine faulty, due to agreement, validity and termination properties, replicas in $Q$ and $R$ output $v_2$. Suppose replicas in $Q$ and $R$ together produce a transcript $T_2$ of all the messages they have received.

**World 3.** $[n - t$ Byzantine faults satisfying validity, termination, forensic support]

Setup. Replicas in $P$ are honest while replicas in $Q$ and $R$ are Byzantine. Replicas in $P$ start with input $v_1$. Replicas in $Q$ and $R$ have access to both inputs $v_1$ and $v_2$. $Q$ behaves as if it starts with input $v_2$ whereas $R$ will use both inputs $v_1$ and $v_2$. Replicas in $Q$ and $R$ behave with $P$ exactly like in World 1. In particular, replicas in $Q$ do not send any message to any replica in $P$. Replicas in $R$ perform a split-brain attack where one brain interacts with $P$ as if the input is $v_1$ and it is not receiving any message from $Q$. Also, separately, replicas in $Q$ and the other brain of $R$ start with input $v_2$ and communicate with each other exactly like in World 2. They ignore messages arriving from $P$.

Output. For replicas in $P$, this world is indistinguishable from that of World 1. Hence, they output $v_1$. Replicas in $P$ and the first brain of $R$ output transcript $T_1$ corresponding to the output. Replicas in $Q$ and the other brain of $R$ behave exactly like in World 2. Hence, they can output transcript $T_2$. Since the protocol provides $(n - t, n - f, d)$-forensic support for $d > n - 2t$, the transcript of messages should hold $d > n - 2t$ Byzantine replicas undeniably corrupt. Suppose the client can find the culpability of $> n - 2t$ replicas from $Q \cup R$, i.e., $\geq 1$ replica from $Q$.

**World 4.** $[n - t$ Byzantine faults satisfying validity, termination, forensic support]
Setup. Replicas in $Q$ are honest while replicas in $P$ and $R$ are Byzantine. Replicas in $Q$ start with input $v_2$. Replicas in $P$ and $R$ have access to both inputs $v_1$ and $v_2$. $P$ behaves as if it starts with input $v_1$ whereas replicas in $R$ use both $v_1$ and $v_2$. Replicas in $P$ and $R$ behave with $Q$ exactly like in World 2. In particular, replicas in $P$ do not send any message to any replica in $Q$. Replicas in $R$ perform a split-brain attack where one brain interacts with $Q$ as if the input is $v_2$ and it is not receiving any message from $P$. Also, separately, replicas in $P$ and the other brain of $R$ start with input $v_1$ and communicate with each other exactly like in World 1. They ignore messages arriving from $Q$.

Output. For replicas in $Q$, this world is indistinguishable from that of World 2. Hence, they output $v_2$. Replicas in $Q$ and the first brain of $R$ output transcript $T_2$ corresponding to the output. Replicas in $P$ and the other brain of $R$ behave exactly like in World 1. Hence, they can output transcript $T_1$.

Observe that the transcript and outputs produced by replicas in $P$, $Q$, and $R$ are exactly the same as in World 3. Hence, the client will hold $n > 2t$ replicas from $Q \cup R$, i.e., $\geq 1$ replica from $Q$ as culpable. However, all replicas in $Q$ are honest in this world. This is a contradiction.

\[\square\]

\section{Proof of Theorems}

\subsection{Proof of Theorem 4.1}

Proof. Suppose there are $f = 2t + 1$ Byzantine replicas, and let there be three replica partitions $P, Q, R$, $|P| = |Q| = t, |R| = t + 1$. To prove the result, suppose the protocol has forensic support for $d > 0$, we construct two worlds where a different set of replicas are Byzantine in each world.

\textbf{World 1.} Let $R, Q$ be Byzantine replicas in this world. During the protocol, replicas in $Q$ behave like honest parties. Suppose in view $e, e'$ ($e < e'$), two honest replicas $p_1, p_2 \in P$ output two conflicting values $v, v'$ after receiving two commitQC. The commitQC for $v$ contains the COMMIT messages from $P$ and $R$, the commitQC for $v'$ contains the COMMIT messages from $R$ and $Q$. All the other messages never reach $P$. During the forensic protocol, replicas in $P$ send their transcripts to the client. Since the protocol has forensic support for $d > 0$, using these transcripts (two commitQC), the forensic protocol determines some subset of $R$ are culpable (since $Q$ behave like honest).

\textbf{World 2.} Let $P, Q$ and some replica $r \in R$ are Byzantine replicas and replicas in $r$ behave honestly. Again, in view $e, e'$ ($e < e'$), two replicas $p_1, p_2 \in P$ output two conflicting values $v, v'$ after receiving two commitQC. The commitQC for $v$ contains the COMMIT messages from $P$ and $R$, the commitQC for $v'$ contains the COMMIT messages from $R$ and $Q$. Replicas in $R$ unlock themselves due to receiving a higher prepareQC formed in $e'$ ($e < e' < e'$). During the forensic protocol, replicas in $P$ send the same transcripts as of World 1 to the client (only two commitQC). Thus, the forensic protocol outputs some subset of $R$ as culpable replicas. However, this is incorrect since replicas in $R$ are honest ($r$ is indistinguishable with replicas in $R \setminus \{r\}$). This complete the proof.

\[\square\]
During the forensic protocol, all honest replicas in $P$ and $Q$ send their transcripts. Byzantine $R$ and $x$ do not provide any information. Since the protocol has forensic support, the forensic protocol can output $d > 1$ replicas in $R$ and $x$ as culprits.

**World 2** is presented in Figure 6. Let $P$ and $x$ be Byzantine replicas in this world. Here, in view $v' > e$, $P$ and $x$, together with $Q$ sign $\text{prepareQC}$ on $v'$. In this world, $v'$ is the first view where a $\text{prepareQC}$ for $v'$ is formed. View $v' > e'$ is similar to that of World 1 except that honest $R$ receives a $\text{NewView}$ message with $\text{prepareQC}^r$ (rather than $\text{prepareQC}_{\text{old}}$).

During the forensic protocol, $Q$ sends their transcripts, which are identical to those in World 1. Byzantine $P$ can provide the same transcripts as those in World 1. Observe that the transcripts from $P$ and $Q$ presented to the forensic protocol are identical to those in World 1. Thus, the forensic protocol can also outputs $d > 1$ replicas in $R$ and $x$ as culpable. In World 2, this is incorrect since replicas in $R$ are honest.

Based on $2t$ transcripts, World 1 and World 2 are indistinguishable. To obtain an irrefutable proof of $d > 1$ culprits, the client needs to collect more than $2t$ transcripts, more than the number of honest parties available. This completes the proof. □

**Remark.** The above proof can be easily modified to work with parameters $d > 0$ when $m = t + 2$.

### C.3 Proof of Theorem 6.1

**Proof.** We construct two worlds where a different set of replicas are Byzantine in each world. Let replicas be split into three partitions, $P$, $Q$, and $R$, and $|P| = (n - 2\varepsilon)/3$, $|Q| = |R| = (n + \varepsilon)/3$ and $\varepsilon > 0$ is a small constant. Denote the numbers of replicas from $P$, $Q$, $R$ in a committee by $p$, $q$, $r$. Let $\kappa$ denote the expected committee size; $t_H = 2\kappa/3$. With constant probability, we will have $p < \kappa/3$, $q > \kappa/3$ and $r > \kappa/3$ and $p + q < 2\kappa/3$ in steps 4 to 8.

**World 1.** Replicas in $R$ are Byzantine in this world. We have $p + q < t_H$ and $q + r > t_H$. The Byzantine parties follow the protocol in Graded Consensus. Thus, all replicas in step 4 hold the same tuple of $(b = 0$ and $v (v \neq v_2))$. Then, the following steps are executed.

Step 4 Honest committee members that belong to $P$ and $Q$ broadcast their votes on $(b = 0, e)$ whereas Byzantine committee members that belong to $R$ send votes to replicas in $P$ and not $Q$.

Step 5 Replicas in $P$ satisfy Ending Condition 0, and output $b = 0$ and the value $v$. Replicas in $Q$ do not receive votes from committee members in $R$, so they update $b = 0$ and broadcast their votes on $(b = 0, v)$. Byzantine committee members that belong to $R$ pretend not to receive votes from committee members in $Q$, and also update $b = 0$. And they send votes to replicas in $P$ and not $Q$.

Step 6 Replicas in $Q$ update $b = 1$ since they receive $p + q < t_H$ votes. Replicas in $R$ pretend not to receive votes from committee members in $Q$, and also update $b = 1$. Committee members in $Q$ and $R$ broadcast their votes.

Steps 7-8 Committee members that belong to $Q$ and $R$ receive $q + r > t_H$ votes, so they update $b = 1$ and broadcast their votes.

Step 9 Replicas in $Q$ and $R$ satisfy Ending Condition 1, and output $b = 1$ and $v_+$, a disagreement with replicas in $P$.

**World 2.** This world is identical to World 1 except (i) Replicas in $Q$ are Byzantine and replicas in $R$ are honest, and (ii) the Byzantine set $Q$ behaves exactly like set $R$ in World 1, i.e., replicas in $Q$ do not send any votes to $R$ in steps 4 and 5 and ignore their votes. During the forensic protocol, $P$ send their transcripts and state that they have output $b = 0$, $Q$ and $R$ send their transcripts claiming in steps 4 and 5 they do not hear from the other partition, and they state that output $b = 1$.

If this protocol has any forensic support, then it should be able to detect some replica in $R$ as Byzantine.

**Remark.** The commitQC of $b$ (the QC in $e + 3$) and commitQC of $b''$ intersect in $t + 1$ replicas. These $t + 1$ replicas should be Byzantine since the protocol requires a replica to vote for at most one value in a view.

Witneses. Client can get the proof based on the two blocks in $e + 3$, so additional witnesses are not necessary in this case.

**Remark.** The view $e$ is larger than their locks because the block is on another chain. Such a view must exist since $b''$ is output in view $e' > e$ and a lock will be formed in at least view $e'$. For a lock to be formed, a higher $\text{prepareQC}$ must be formed too.

Consider the first view $e < e^* \leq e'$ in which a $\text{prepareQC}$ in chain of $b''$ is formed. The leader in $e''$ broadcasts the block containing a $\text{highQC}$ on $(b''', e''')$. Since this is the first time a higher $\text{prepareQC}$ is formed and there is no $\text{prepareQC}$ for chain of $b''$ formed between view $e$ and $e''$, we have $e''' \leq e'$. The formation of the higher $\text{prepareQC}$ indicates that $2t + 1$ replicas received the block extending $b''$ with $\text{highQC}$ on $(b''', e''')$ and consider it a valid proposal, i.e., the view number $e'''$ is larger than their locks because the block is on another chain.

Recall that the output block $b$ indicates $2t + 1$ replicas are locked on $(b, e)$ at the end of view $e$. In this case, the $2t + 1$ votes in $\text{prepareQC}$ in view $e''$ intersect with the $2t + 1$ votes in $\text{commitQC}$ in view $e$ at $t + 1$ Byzantine replicas. These replicas should be Byzantine because they were locked on the block $b$ in view $e$ and vote for a conflicting block in a higher view $e'$ whose $\text{highQC}$ is from a view $e''' \leq e'$. Thus, they have violated the voting rule.

Witnesses. Client can get the proof by storing a $\text{prepareQC}$ formed in $e''$ between $e$ and $e'$ in a different chain from $b$. The $\text{prepareQC}$

During the forensic protocol, replicas in $P$ send their transcripts and state that they have output $b = 0$, $Q$ and $R$ send their transcripts claiming in steps 4 and 5 they do not hear from the other partition, and they state that output $b = 1$.

If this protocol has any forensic support, then it should be able to detect some replica in $R$ as Byzantine.

**Remark.** The commitQC of $b$ (the QC in $e + 3$) and commitQC of $b''$ intersect in $t + 1$ replicas. These $t + 1$ replicas should be Byzantine since the protocol requires a replica to vote for at most one value in a view.

Witnesses. Client can get the proof based on the two blocks in $e + 3$, so additional witnesses are not necessary in this case.

**Remark.** The view $e$ is larger than their locks because the block is on another chain. Such a view must exist since $b''$ is output in view $e' > e$ and a lock will be formed in at least view $e'$. For a lock to be formed, a higher $\text{prepareQC}$ must be formed too.

Consider the first view $e < e^* \leq e'$ in which a $\text{prepareQC}$ in chain of $b''$ is formed. The leader in $e''$ broadcasts the block containing a $\text{highQC}$ on $(b''', e''')$. Since this is the first time a higher $\text{prepareQC}$ is formed and there is no $\text{prepareQC}$ for chain of $b''$ formed between view $e$ and $e''$, we have $e''' \leq e'$. The formation of the higher $\text{prepareQC}$ indicates that $2t + 1$ replicas received the block extending $b''$ with $\text{highQC}$ on $(b''', e''')$ and consider it a valid proposal, i.e., the view number $e'''$ is larger than their locks because the block is on another chain.

Recall that the output block $b$ indicates $2t + 1$ replicas are locked on $(b, e)$ at the end of view $e$. In this case, the $2t + 1$ votes in $\text{prepareQC}$ in view $e''$ intersect with the $2t + 1$ votes in $\text{commitQC}$ in view $e$ at $t + 1$ Byzantine replicas. These replicas should be Byzantine because they were locked on the block $b$ in view $e$ and vote for a conflicting block in a higher view $e'$ whose $\text{highQC}$ is from a view $e''' \leq e'$. Thus, they have violated the voting rule.

Witnesses. Client can get the proof by storing a $\text{prepareQC}$ formed in $e''$ between $e$ and $e'$ in a different chain from $b$. The $\text{prepareQC}
is for the previous block in $e^i$ whose $highQC$ is formed in a view $e'' < e$. For the replicas who have access to the $prepareQC$, they must have access to all blocks in the same blockchain. Thus only one witness is needed ($k = 1$) to provide the $prepareQC$ and its previous block containing the $highQC$ on $(b'', e'')$, the $prepareQC$, $highQC$ and the first $commitQC$ act as the irrefutable proof. □

D DESCRIPTION OF BFT PROTOCOLS

In the protocols in this paper, we assume that replicas and clients ignore messages with invalid signatures and messages containing external invalid values. When searching for an entity (e.g. lock or $prepareQC$) with the highest view, break ties by alphabetic order of the value. Notice that ties only occur when $f > t$ and Byzantine replicas deliberately construct conflicting quorum certificates in a view.

With $n = 3t + 1$, the descriptions of the PBFT protocol and HotStuff protocol are presented in Algorithm 5 and 6.

D.1 A Forensic Attack on HotStuff-view

Compared to HotStuff [32, Algorithm 2], Algorithm 6 highlights a slightly different voting rule in line 6. In addition to check whether $(LOCK.e < highQC.e) \lor (LOCK.v = v)$ holds as in HotStuff [32, Algorithm 2], when the value in $NewView$ is the same as the value in lock, our voting rule requires $LOCK.e = highQC.e$.

We argue that the lack of this additional check on the view number will not affect the safety and liveness for HotStuff, but pose a threat for forensics. In the following, we exhibit a forensic attack on HotStuff-view protocol with the original voting rule.

- $e = i > 0$ : An honest replica $R$ receives a $(Commit, i, v, \sigma)$ from the leader and updates its lock to be $(i, v, \sigma)$. $R$ sends $(Commit, i, v)$ to leader, which is contained in a $commitQC$ denoted as $qc_1$. $v$ is output in this view.
- $e = i + 1$ : $R$ receives $(Commit, i + 1, v', \sigma')$ and updates its lock to be $(i + 1, v', \sigma')$.
- $e = i + 2$ : A leader broadcasts $(NewView, i + 2, v', highQC)$, where $highQC$ is a QC from $i - 1$. Replica $R$ receives the message and sends $(Prepare, i + 2, v', i - 1)$, because $LOCK.v = v'$ by checking the original voting rule. This message is contained into a $prepareQC$ denoted as $qc_2$. Further, $v'$ is output in this view.

In this execution, replica $R$ follows the protocol, however, it will be mistakenly blamed by Algorithm 2 if the client receives the $qc_1$ for $v$ and the $qc_2$ for $v'$. Since $qc_2.v \neq qc_1.v$ and $qc_2.e_{qc} = i - 1 \leq qc_1.e = i$ according to line 10.

While the actual $prepareQC$ whose intersection with $qc_1$ should be blamed is generated in $e = i + 1$, it is possible that some honest replicas who have the same transcripts as $R$ will be improperly held culpable in this case. By adding the condition to check $LOCK.e = highQC.e$, honest replicas will not vote for a $NewView$ with stale $highQC$, which prevents them from the attack described above.
Algorithm 5 PBFT-PK protocol: replica’s initial value $v_i$

1: $LOCK \leftarrow (0, v_L, \sigma_L)$ with selectors $e, v, \sigma$ \quad \triangleright \ 0, v_L, \sigma_L$: default view, value, and signature
2: $e \leftarrow 1$
3: \textbf{while} true \textbf{do} \quad \triangleright \textbf{Pre-prepare and Prepare Phase}
4: \hspace{1em} \textbf{as a leader}
5: \hspace{2em} collect $\langle \text{ViewChange}, e - 1, \cdot \rangle$ from $2t + 1$ distinct replicas as status certificate $M$ \quad \triangleright \text{Assume special ViewChange messages from view 0}
6: \hspace{2em} $v \leftarrow$ the locked value with the highest view number in $M$
7: \hspace{2em} \textbf{if} $v = v_L$ \textbf{then}
8: \hspace{3em} $v \leftarrow v_i$
9: \hspace{3em} broadcast $\langle \text{NewView}, e, v, M \rangle$
10: \hspace{2em} \textbf{as a replica}
11: \hspace{3em} wait for valid $\langle \text{NewView}, e, v, M \rangle$ from leader \quad \triangleright \text{Use function VALID(\langle NewView, e, v, M \rangle)}
12: \hspace{3em} send $\langle \text{Prepare}, e, v \rangle$ to leader
13: \hspace{1em} \textbf{Commit Phase}
14: \hspace{2em} \textbf{as a leader}
15: \hspace{3em} collect $\langle \text{Prepare}, e, v \rangle$ from $2t + 1$ distinct replicas, denote the collection as $\Sigma$
16: \hspace{3em} $\sigma \leftarrow \text{aggregate-sign}(\Sigma)$
17: \hspace{3em} broadcast $\langle \text{Commit}, e, v, \sigma \rangle$
18: \hspace{2em} \textbf{as a replica}
19: \hspace{3em} wait for $\langle \text{Commit}, e, v, \sigma \rangle$ from leader \quad \triangleright \text{prepareQC}
20: \hspace{3em} $LOCK \leftarrow (e, v, \sigma)$
21: \hspace{2em} \textbf{Reply Phase}
22: \hspace{2em} \textbf{as a leader}
23: \hspace{3em} collect $\langle \text{Commit}, e, v \rangle$ from $2t + 1$ distinct replicas, denote the collection as $\Sigma$
24: \hspace{3em} $\sigma \leftarrow \text{aggregate-sign}(\Sigma)$
25: \hspace{3em} broadcast $\langle \text{Reply}, e, v, \sigma \rangle$
26: \hspace{2em} \textbf{as a replica}
27: \hspace{3em} wait for $\langle \text{Reply}, e, v, \sigma \rangle$ from leader \quad \triangleright \text{commitQC}
28: \hspace{3em} output $v$ and send $\langle \text{Reply}, e, v, \sigma \rangle$ to the client
29: \hspace{2em} call procedure ViewChange()
30: \hspace{1em} \textbf{if} a replica encounters timeout in any “wait for”, call procedure ViewChange()
31: \hspace{1em} \textbf{procedure ViewChange()}
32: \hspace{2em} broadcast $\langle \text{Blame}, e \rangle$
33: \hspace{2em} collect $\langle \text{Blame}, e \rangle$ from $t + 1$ distinct replicas, broadcast them
34: \hspace{2em} quit this view
35: \hspace{2em} enter the next view, $e \leftarrow e + 1$
36: \hspace{1em} \textbf{function VALID(\langle NewView, e, v, M \rangle)}
37: \hspace{2em} $v^* \leftarrow$ the locked value with the highest view number in $M$
38: \hspace{2em} \textbf{if} $(v^* = v \lor v^* = v_L) \land (M \text{ contains locks from } 2t + 1 \text{ distinct replicas})$ \textbf{then}
39: \hspace{3em} return $true$
40: \hspace{2em} else
41: \hspace{3em} return $false$
Algorithm 6 General HotStuff protocol: replica's initial value \( v_i \), protocol variant indicator \( v_{\text{var}} \in \{\text{HotStuff-view'}, \text{HotStuff-hash'}, \text{HotStuff-null'}\) } 

1: \( \text{prepareQC} \leftarrow (0, v_{\bot}, \sigma_{\bot}, \text{Info}_{\bot}) \) with selectors \( e, v, \sigma, \text{Info} \)

2: \( \text{LOCK} \leftarrow (0, v_{\bot}) \) with selectors \( e, v \) \( \rightarrow \) \( 0, v_{\bot}, \sigma_{\bot}, \text{Info}_{\bot} \): default view, value, signature, and info

3: \( e \leftarrow 1 \)

4: while true do

\( \triangleright \quad \text{Pre-prepare and Prepare Phase} \)

5: as a leader

6: collect \( \langle \text{ViewChange}, e - 1, \cdot \rangle \) from \( 2t + 1 \) distinct replicas as \( M \) \( \triangleright \) Assume special ViewChange messages from view 0

7: \( \text{highQC} \leftarrow \) the highest QC in \( M \)

8: \( v \leftarrow \text{highQC}.v \)

9: if \( v = v_{\bot} \) then

10: \( v \leftarrow v_i \)

11: broadcast \( \langle \text{NewView}, e, v, \text{highQC} \rangle \)

12: as a replica

13: wait for \( \langle \text{NewView}, e, v, \text{highQC} \rangle \) from leader s.t. \( \text{highQC}.v = v \lor \text{highQC}.v = v_{\bot} \) \( \triangleright \) Validate \( v \)

14: if \( (\text{LOCK}.e < \text{highQC}.e) \lor (\text{LOCK}.e = v_{\bot} \land \text{LOCK}.e = \text{highQC}.e) \) then

15: send \( \langle \text{Prepare}, e, v, \text{Info}(v_{\text{var}}, \text{highQC}) \rangle \) to leader \( \triangleright \) Voting rule, see Appendix D.1

\( \triangleright \quad \text{Precommit Phase} \)

16: as a leader

17: collect \( \langle \text{Prepare}, e, v, \text{Info} \rangle \) from \( 2t + 1 \) distinct replicas, denote the collection as \( \Sigma \)

18: \( \sigma \leftarrow \text{aggregate-sign}(\Sigma) \)

19: broadcast \( \langle \text{Precommit}, e, v, \sigma, \text{Info} \rangle \)

20: as a replica

21: wait for \( \langle \text{Precommit}, e, v, \sigma, \text{Info} \rangle \) from leader \( \triangleright \) prepareQC

22: \( \text{prepareQC} \leftarrow (e, v, \sigma, \text{Info}) \)

23: send \( \langle \text{Precommit}, e, v \rangle \) to leader

\( \triangleright \quad \text{Commit Phase} \)

24: as a leader

25: collect \( \langle \text{Precommit}, e, v \rangle \) from \( 2t + 1 \) distinct replicas, denote the collection as \( \Sigma \)

26: \( \sigma \leftarrow \text{aggregate-sign}(\Sigma) \)

27: broadcast \( \langle \text{Commit}, e, v, \sigma \rangle \)

28: as a replica

29: wait for \( \langle \text{Commit}, e, v, \sigma \rangle \) from leader \( \triangleright \) precommitQC

30: \( \text{LOCK} \leftarrow (e, v) \)

31: send \( \langle \text{Commit}, e, v \rangle \) to leader

\( \triangleright \quad \text{Reply Phase} \)

32: as a leader

33: collect \( \langle \text{Commit}, e, v \rangle \) from \( 2t + 1 \) distinct replicas, denote the collection as \( \Sigma \)

34: \( \sigma \leftarrow \text{aggregate-sign}(\Sigma) \)

35: broadcast \( \langle \text{Reply}, e, v, \sigma \rangle \)

36: as a replica

37: wait for \( \langle \text{Reply}, e, v, \sigma \rangle \) from leader \( \triangleright \) commitQC

38: output \( v \) and send \( \langle \text{Reply}, e, v, \sigma \rangle \) to the client

39: call procedure ViewChange()

40: if a replica encounters timeout in any “wait for”, call procedure ViewChange()
procedure ViewChange()
  broadcast ⟨Blame, e⟩
  collect ⟨Blame, e⟩ from t + 1 distinct replicas, broadcast them
  quit this view
  send ⟨ViewChange, e, prepareQC⟩ to the next leader
  enter the next view, e ← e + 1

function Info(var, highQC)
  if var = ’HotStuff-view’ then
    return highQC.𝑒
  if var = ’HotStuff-hash’ then
    return Hash(highQC)
  if var = ’HotStuff-null’ then
    return ∅