



Journal Website:
<http://sciencebring.com/index.php/ijasr>

Copyright: Original content from this work may be used under the terms of the creative commons attributes 4.0 licence.

Research Article

Deterministic Evidence Validation in A Byzantine Fault Tolerant File Registry

Submission Date: February 12, 2026, **Accepted Date:** March 10, 2026,

Published Date: March 31, 2026

Crossref doi: <https://doi.org/10.37547/ijasr-06-03-09>

B. B. Turdibekov

Tashkent University of Information Technologies named after Muhammad al-Khwarizmi, Uzbekistan

ABSTRACT

A permissioned file registry needs both agreement among validators and a defensible rule for accepting evidence about a file. Consensus can order records consistently, yet it cannot establish that an asserted digest belongs to the intended document. This article develops a validation boundary for file hash, CID, identity and storage availability checks in a Byzantine fault tolerant registry. The method separates checks computable from a proposal from checks that depend on mutable external state. A transaction schema, acceptance states and registration strategies are derived and examined through logical failure cases. The analysis shows when a confirmed entry supports a byte binding claim and when it merely records a submitted assertion. It gives evaluation criteria for PBFT, BFT-SMaRt and HotStuff based deployments with an explicit definition of the work performed before and after confirmation.

KEYWORDS

Byzantine fault tolerance, permissioned registry, deterministic validation, content identifier, application validity.

1. INTRODUCTION

A registry for digital files may record a file digest, a content identifier, submitter identity and transaction

history. Validators must reach agreement on which records become part of the shared state despite faulty

participants. Byzantine fault tolerant state machine replication supplies one family of approaches to this problem [1]. PBFT, BFT-SMaRt and HotStuff offer influential designs with different interfaces and communication patterns [2][3][4]. Their shared purpose is to coordinate a replicated decision under defined assumptions.

A separate question precedes the vote: what exactly is a valid file record? A block may contain a syntactically correct digest that was computed from the wrong file, a CID whose content is unavailable, or a valid signature from a user who lacks permission for that document. Agreement on the block does not repair these application errors. Conversely, a useful application check can create inconsistent votes if it depends on an external service that each validator observes at a different moment.

This article asks where the boundary should lie between deterministic proposal validation, storage checks and subsequent audit. Its result is a claim

$$P = (\text{record ID, version ID, file digest } H, \text{CID, submitter, signature, policy ID}).$$

To avoid binding a digest to the wrong logical document, the record also needs a version reference and a controlled submitter rule. A time stamp can provide an additional assertion that data existed before a stated time when issued under the conditions of a time-stamping authority, but it is not a substitute for file approval or source identity [5]. A time field supplied by the client alone has a different trust basis. The represented file F is stored outside the ledger. A complete claim that the registry binds P to F needs both an accepted record and a controlled account of how H and CID were produced from the same F . A validator can verify only the information available to it under the specified transaction design.

aware transaction design. The analysis is directed to a permissioned registry that stores concise evidence rather than full documents on the ledger. The transaction design makes the validation context part of the accepted evidence.

2. METHODS

2.1 Fault and application assumptions

The comparison uses the conventional bounded Byzantine fault setting in which $n = 3f + 1$ validators can tolerate at most f faulty members under the relevant protocol assumptions [2]. A quorum of $2f + 1$ intersects another such quorum in at least $f + 1$ members. This overlap is an arithmetic safety ingredient, but protocol safety also depends on the rules for votes, certificates, locking and changes of leader. Membership must be known consistently during a decision. Voting and reconfiguration rules remain part of the full safety argument.

The registry application receives a proposal

2.2 Classification of checks

Checks are classified by whether every honest validator can compute the same result from agreed inputs. Syntax, field ranges, signature verification and permission rules based on a common policy snapshot are proposal local. Recomputing H is also deterministic when the relevant bytes are included in the proposal or supplied through an authenticated common input. Fetching a CID from a live IPFS network is different: one validator may retrieve content while another encounters a temporary outage. This availability observation must be kept distinct from a claim of byte integrity.

The decision function is written $V(P,S)$ where S is the agreed application state. Determinism requires that any two honest validators using the same proposal and committed policy snapshot obtain the same application result. Equation 1 gives the invariant required at the application boundary.

$$\forall i, j \in \text{Honest}: V_i(P, S) = V_j(P, S) \quad (1)$$

For the conventional threshold $n = 3f + 1$ and $q = 2f + 1$, the quorum intersection bound is $|Q_1 \cap Q_2| \geq 2q - n = f + 1$. The bound explains why

two threshold certificates share at least one honest member under the fault limit, but it does not replace the locking and view-change rules of PBFT or HotStuff [2][4]. V returns accept or reject when all required inputs are deterministic. For an external check $X(P, t, \text{node})$, the result may vary with time and node. The registry must either make X an input to a separately specified protocol with consistent evidence, or remove X from the deterministic acceptance condition and handle it through later audit. Merely adding a common timeout does not guarantee equal observations of the storage network.

Table 1.

Validation location and supported conclusion

Check	Input needed	Acceptance role	Claim supported
Signature	Proposal and authorized key	Deterministic gate	Submitter key signed record
Permission	Proposal and agreed policy state	Deterministic gate	Submitter may request this operation
Digest format	Proposal fields	Deterministic gate	Field is well formed only
Digest of supplied bytes	Shared file bytes	Deterministic gate if bytes shared	Supplied bytes match H
CID retrieval	External storage network	Audit or specified evidence protocol	Content was obtainable at check time
Historical source approval	Workflow record	Separate governance check	Document was approved in context

2.3 Two registration strategies

Strategy A places file bytes, or a cryptographically authenticated common representation of them, within the validator input. Honest validators can then

compute the digest and check the link to P . The ledger may retain only the compact record after acceptance. This strengthens the meaning of a confirmed transaction but increases the input and processing

burden. For large or sensitive files, the application must specify confidentiality and transfer rules.

Strategy B sends only the digest, CID and authorization evidence to validators. The proposal local checks establish that a permitted submitter recorded a well formed assertion. A separate storage service retains the file, and a later verifier retrieves content and compares its digest with H. This lowers consensus input volume, but the accepted record cannot by itself prove that validators saw or hashed the file. The claim must be worded accordingly. Both strategies require controlled handling of file version and submitter authority.

2.4 Logical analysis procedure

Each strategy was examined against four situations: correct submission, incorrect digest, temporary unavailability of IPFS content and disagreement about

a permission rule. The analysis follows which information is common to validators, whether votes can diverge and what a user may conclude from an accepted record. These are logical cases, not observations from an implemented deployment.

3. RESULTS

3.1 Acceptance boundary

The resulting architecture separates application admission, consensus and post acceptance verification. Figure 1 depicts the boundary. The record is first checked against rules that can be reproduced from common inputs. Validators then agree on its placement. Storage retrieval and later rehashing are represented as distinct events with their own timestamps and outcomes.

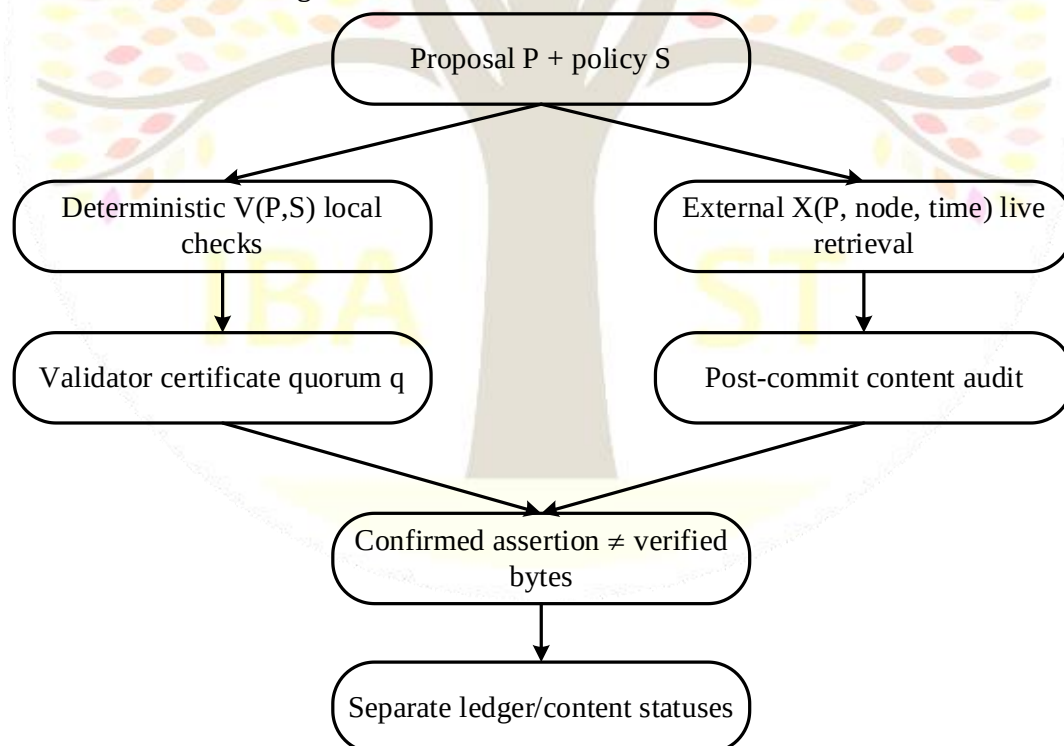


Figure 1. Application validation and validator agreement for a file record



Under Strategy A, a mismatch between supplied bytes and H is a deterministic reason to reject when all honest validators receive the same authenticated bytes. Under Strategy B, the same mismatch cannot be detected solely from the proposed digest and CID fields. The accepted ledger entry instead preserves who asserted those values and when the registry accepted the assertion. Later rehashing can contradict it. This difference determines the exact wording of a confirmation response.

3.2 State semantics

The design uses separate states for rejected proposal, pending agreement, confirmed assertion, verified content and unavailable content. “Confirmed assertion” means the transaction passed proposal local rules and was ordered by the validator set. “Verified content” additionally requires a successful retrieval and a digest comparison. “Unavailable content” means the comparison could not be completed; it is neither a match nor a mismatch. These distinctions are summarized in Table 2.

Table 2.

Meaning of registry and storage states

State	Established condition	Valid user statement	Open issue
Rejected	Proposal local rule failed	No accepted record	Reason for rejection
Pending	No final certificate yet	Acceptance unresolved	Agreement progress
Confirmed assertion	Record accepted by validators	Assertion was registered	Whether content matches
Verified content	Retrieved bytes hash to H	Retrieved version matches record	Original source approval
Unavailable content	Retrieval failed	Stored bytes not checked	Availability and retention

A confirmation identifier should therefore identify a specific version and transaction. A generic statement that the “file is verified” collapses at least two different states. The narrower message permits later storage audit without rewriting what the original consensus result meant.

3.3 Failure cases

If live retrieval is placed directly inside the voting rule, temporary storage differences may cause honest validators to vote differently. This can delay liveness even though the proposal fields are identical. If availability is audited after confirmation, votes remain based on common inputs, while the audit reports a separate inability to retrieve content. The tradeoff is

visible in the claim: acceptance of the assertion precedes independent content checking.

An inconsistent permission snapshot is a different source of divergence. The validator set must agree on which policy version applies to a proposal. A policy ID in P is useful only if the referenced rules are available in agreed state and take effect at a specified point. Likewise, membership changes require a defined transition; a threshold without a shared member list does not define a quorum. These requirements belong to the application and governance layer around any selected BFT protocol.

3.4 Evaluation matrix

A meaningful comparison of PBFT, BFT-SMaRt and HotStuff deployments requires the same application workload, validator count, transaction payload, network conditions and persistence policy. Performance surveys caution that throughput figures depend on measurement boundaries [6]. For a file registry, results should separately report consensus confirmation time, file transfer and hashing time, storage retrieval time and end to end user response. Without these separations, the effect of a consensus choice cannot be isolated from application work.

3.5 Binding the certificate to a validation context

A proposal should identify not only its payload but the context under which it was checked. Let e be the membership epoch, p the policy version and v the application validation version. Define $d_P = H(\text{Encode}(P, e, p, v))$, where Encode has an unambiguous field layout. A vote is bound to d_P together with the round and protocol phase required by the selected consensus protocol. If the certificate records only a file hash while leaving policy and epoch

implicit, later verification cannot establish which admission rule the validators applied.

$$d_P = H(\text{Encode}(P, e, p, v)) \quad (2)$$

A verifier first resolves the authorized validator set for e , checks certificate signatures and the threshold, then evaluates the application state transition under p and v . This order prevents a valid certificate from an earlier policy from being silently interpreted under new permissions. The context fields are application data included in the proposal and covered by its certificate.

3.6 Content assertions and deterministic receipts

Strategy B can incorporate a signed registration receipt from a designated intake service. Such a receipt can bind a file digest, CID, import profile, submitter and request ID. Validators can deterministically verify its signature and issuer permission from agreed state. This narrows the claim: the ledger records that an authorized intake service asserted the binding. It does not imply that every validator independently fetched the content. The service becomes an explicit trust dependency instead of an undocumented part of file registration.

A compromised intake service could sign a consistent-looking false binding. Increasing the number of validators does not remove that single source dependency if every validator merely verifies the same receipt. Where independent byte verification is required, the design must supply the bytes or a separately specified verification mechanism. This distinction reveals a common architectural error: replication of an assertion is treated as diversification of evidence even though all replicas rely on the same origin.



Table 3.

Trust placement in registration strategies

Strategy	Common validator input	Main trust dependency	Confirmed statement
Full bytes	File and signed proposal	Admission and consensus rules	Supplied bytes match registered digest
Signed receipt	Receipt and policy state	Authorized intake service	Service asserted this binding
Bare assertion	Digest, CID and submitter signature	Submitter and later audit	Submitter registered these fields

3.7 Logical failure trace

Consider a request for which intake returns a receipt, consensus commits the record, but the response to the client is lost. The client retries using the same request ID. Deterministic admission should return the committed result rather than create a second logical action. The request ID and its associated payload digest must be stored in agreed state. If the same ID arrives with a different digest, the proposal is rejected as a conflict. This rule is reproducible because its inputs are already replicated.

Now consider policy p allowing the submitter at proposal time, followed by $p + 1$ revoking that right. The protocol must specify which committed state precedes the transaction in the agreed order. Validation against that state produces a single result; independently reading a live authorization service can produce different votes. The trace shows that determinism concerns both file evidence and governance data.

3.8 Work distribution and resource consequences

For n validators independently processing B file bytes, aggregate byte hashing work is nB , while a signed receipt strategy requires approximately n receipt verifications plus one intake pass over B . The reduction in duplicated work comes with the changed trust dependency described above. These expressions measure work allocation and do not turn one strategy into a universal preference. Large files and restrictive confidentiality may favor receipt admission; independent content validation may justify distributing bytes.

The ledger should expose the strategy identifier with the transaction so that clients can select the assurance they require. A later audit can add a content match event to a previously accepted assertion, but it must not rewrite the historical statement that the original validators made. This yields an append-only progression of evidence: accepted assertion, retrieved content and verified digest remain distinguishable events tied to the same transaction.

4. DISCUSSION

The central design choice is not merely which BFT protocol processes more transactions. It is what each transaction claims. A registry that accepts a hash without seeing bytes can still be valuable: it provides an ordered record of submitted assertions, permissions and later checks. Its value is diminished if the interface describes that record as proof that validators independently examined a document. Claim precision is therefore an architectural property, not an editorial afterthought.

Strategy A and Strategy B can coexist by file class. A small public document may be supplied to validators, while a large confidential file may be handled through a compact assertion plus controlled external verification. The selected strategy must be recorded per transaction so that later readers understand what “confirmed” meant at the time. Mixing strategies under one unlabeled status would create an audit ambiguity.

The CID and full file digest need separate treatment. A CID is a content address shaped by IPFS encoding and representation choices [7]; it is not always a simple textual form of the digest over the original file bytes. The application should register both when both are needed and establish that they were derived from the same accepted input. A protected registration process matters because an internally consistent pair can still refer to the wrong document if the initial submission was misidentified.

Storage availability is operational and time dependent. A blockchain record provides a history of recorded transactions, not the missing file bytes [8]. Even a content addressed network does not guarantee that a particular object remains retrievable indefinitely. Pinning, controlled replicas and recovery

procedures support availability. Later audit should state whether the object was retrieved, when it was retrieved and which bytes were compared. A failure to retrieve is not evidence that the hash changed. A successful retrieval and matching hash does not retrospectively establish that the submitter had authority to register the source file.

5. CONCLUSION

A Byzantine fault tolerant registry can guarantee an ordered shared decision under its protocol assumptions, but the meaning of the registered file evidence depends on application validation. The proposed boundary distinguishes reproducible proposal checks from live storage observations and assigns separate states to confirmed assertions, verified content and unavailable content. This makes the confirmation message auditable and prevents consensus agreement from being described as a check that validators did not perform. It also provides a common workload definition for any future controlled comparison of permissioned registry designs.

REFERENCES

1. L. Lamport, R. Shostak and M. Pease, “The Byzantine Generals Problem,” ACM Transactions on Programming Languages and Systems, vol. 4, no. 3, pp. 382–401, 1982. DOI: 10.1145/357172.357176.
2. M. Castro and B. Liskov, “Practical Byzantine Fault Tolerance,” Proc. OSDI, pp. 173–186, 1999.
3. A. Bessani, J. Sousa and E. E. P. Alchieri, “State Machine Replication for the Masses with BFT-SMaRt,” Proc. DSN, pp. 355–362, 2014. DOI: 10.1109/DSN.2014.43.
4. M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta and I. Abraham, “HotStuff: BFT Consensus with

- Linearity and Responsiveness,” Proc. PODC, pp. 347–356, 2019. DOI: 10.1145/3293611.3331591.
5. C. Adams et al., Internet X.509 Public Key Infrastructure Time-Stamp Protocol (TSP), RFC 3161, 2001. <https://www.rfc-editor.org/info/rfc3161/>.
 6. C. Fan, S. Ghaemi, H. Khazaei and P. Musilek, “Performance evaluation of blockchain systems: A systematic survey,” IEEE Access, vol. 8, pp. 126927–126950, 2020. DOI: 10.1109/ACCESS.2020.3006078.
 7. IPFS, CID (Content IDentifier) Specification. <https://specs.ipfs.tech/cid/>. Accessed September 26, 2026.
 8. D. Yaga, P. Mell, N. Roby and K. Scarfone, Blockchain Technology Overview, NISTIR 8202, 2018. DOI: 10.6028/NIST.IR.8202.

