DRNEALAGGARWAL.INFO · NAIROBI · 2026

Twenty Generals, One Ledger

Why MedLattice Runs on Hyperledger Besu and QBFT

Working through the consensus layer underneath the patient record I am building — from the Byzantine Generals Problem to the three smart contracts actually running on the chain — in enough depth to defend the choice, and enough plain English to explain it.

Dr Neal Aggarwal

Medicine · Surgery · Engineering · Information Technology · Artificial Intelligence

18 SEPTEMBER 2026

What this covers. MedLattice's specification states its topology in one line — "permissioned EVM consortium (Besu/QBFT)" — and moves on. This is the argument behind that line: what a blockchain is actually doing underneath the word, why a permissioned chain and not a public one, what the Byzantine Generals Problem is and why every consensus protocol is answering it, how Practical Byzantine Fault Tolerance became Istanbul BFT became QBFT, why Besu specifically rather than Fabric or Corda, and how MedLattice's own three contracts — IdentityRegistry, ConsentCapabilityManager, RecordAnchorRegistry — sit on top of all of it. A companion piece to *Not Shown Is Not Locked*, which covers the cryptography rather than the ledger.

Contents

This is the PDF companion to the post. The full piece — with live hyperlinks throughout — is at drnealaggarwal.info/post/2026-09-18-twenty-generals-one-ledger. This PDF is for printing or reading offline; the diagrams and references are identical. An audio deep dive specific to this decision (rather than the wider Hyperledger family) is in progress and will be linked from the post once it exists.

1. What a blockchain is actually doing

Strip away the decade of hype and a blockchain is three unremarkable ideas, stacked.

The first is a **ledger**: an ordered list of entries, each one referring back to the one before it — usually by including a cryptographic hash of the previous entry inside the new one, so that changing anything upstream changes every hash downstream of it and becomes instantly detectable. Nothing new here; a paper accounts book has the same property if you refuse to use an eraser.

The second is **replication**: instead of one party holding the ledger, every participating organisation holds an identical copy, and they all update in lockstep. This is the part that actually does the work. A ledger held by one party is only as honest as that party. A ledger held identically by several mutually distrusting parties can only be altered if enough of them collude, which is a much harder thing to arrange quietly.

The third is a **smart contract**: a small program, stored on the ledger itself, that every participant runs an identical copy of. When a transaction calls it, every node executes the same code on the same inputs and arrives at the same output, which is then what gets written to the ledger. Nobody can quietly patch the logic, because everybody is running it, and a change to it is itself a transaction everybody can see.

None of that is where the hard engineering problem lives. The hard problem is the fourth thing, the one that makes the other three trustworthy rather than merely well-intentioned: getting every copy of the ledger to **agree**, entry by entry, even though the parties involved do not fully trust each other and some of them might be faulty, offline, or actively lying. That agreement problem is called **consensus**, and it is the entire subject of this note. Everything downstream — whether MedLattice's record of who accessed what can be trusted, whether a hospital could quietly rewrite its own history, whether the whole thing stays up when one member's server falls over — is a consequence of which consensus protocol was chosen and how it behaves.

The one sentence worth keeping

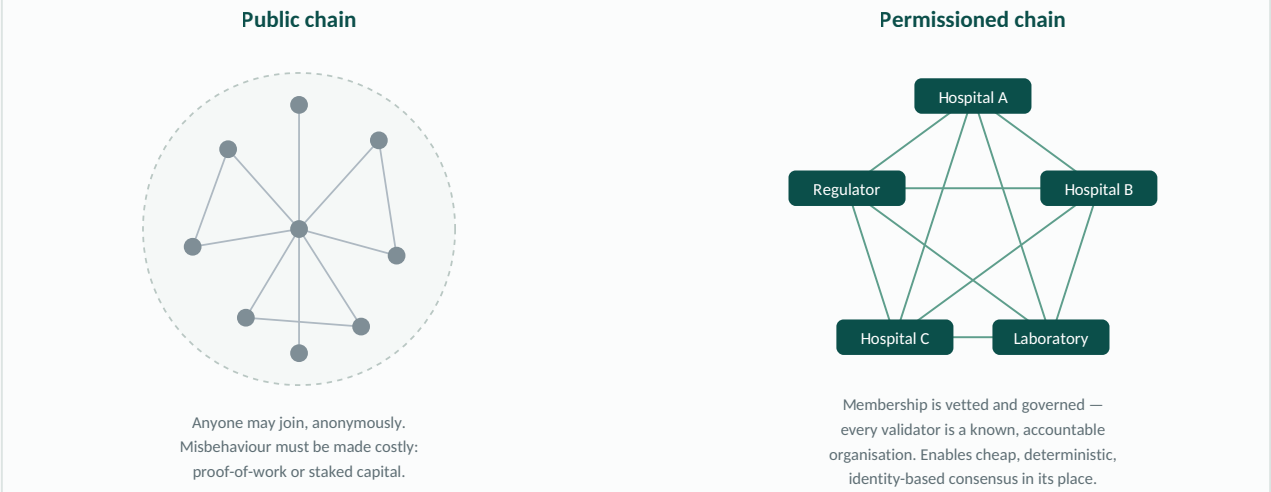
A blockchain is a replicated ledger whose participants use a consensus protocol to agree on what gets appended to it, plus smart contracts that let the ledger enforce rules rather than merely record facts. Everything else — the cryptocurrency, the mining, the public speculation — is one particular application of that machinery, not a required part of it.

2. The fork in the road: public or permissioned

The first real design decision in any blockchain project is not which protocol to use. It is **who is allowed to run a node** — and that decision determines almost everything downstream of it, including which consensus protocols are even available to choose from.

On a **public** chain — Bitcoin, public Ethereum — anybody can join as a participant with no vetting at all. Nobody needs your permission to run a node, hold funds, or, on Ethereum, deploy a contract. That openness is the entire point of those systems, and it comes at a specific cost: because anyone can join, and anyone who joins might be an attacker, the consensus protocol has to defend against an unbounded, anonymous, potentially enormous set of participants. That is why Bitcoin spends real electricity on proof-of-work and why even Ethereum's proof-of-stake needs an economic penalty — slashing a validator's staked funds — to make misbehaviour costly. Openness has to be paid for somewhere, and on a public chain it is paid for in energy, capital lockup, or both.

On a **permissioned** chain, only vetted, known organisations may run a node at all. Membership is itself governed — typically by a vote among existing members, recorded on the chain, exactly like everything else. Because the validator set is small, known, and legally accountable (these are real organisations with real names, not anonymous addresses), a permissioned chain can use consensus protocols that would be hopeless at public-internet scale: ones that need every validator to know every other validator's identity, that scale quadratically with the number of validators, and that assume a validator who misbehaves can, in principle, be identified and removed by the others. That trade — give up permissionless openness, get deterministic finality and much cheaper agreement — is the trade MedLattice makes, and it was the right one for a system holding protected health information, for a reason that has nothing to do with performance.



The decision that precedes every other decision. A public chain has to defend an open, anonymous validator set with economic cost — proof-of-work or staked capital. A permissioned chain trades that openness for a small, vetted, accountable validator set, which is what makes the consensus protocols in the rest of this note possible at all.

The reason is the one already argued at length in Not Shown Is Not Locked: MedLattice's cryptography keeps patient-facing content off the ledger entirely — the compartments, the envelopes, the key tree, all of it operate independently of which chain the shared logbook happens to run on. So the choice of public versus permissioned was never really a cryptography question. It was a governance and a risk-surface question. A public chain has no membership to vet, which means anybody in the world can submit a transaction to a contract sitting on the same infrastructure my consortium's contracts sit on, and the entire attack surface of a global, permissionless network becomes MedLattice's attack surface by association — whether or not a single byte of patient data is exposed to it. A permissioned consortium chain has none of that surface. The only entities that can ever submit a transaction, propose a block, or vote on one are the hospitals, laboratories and regulators who were vetted onto the network in the first place. That is a smaller, more boring, much more defensible perimeter, and boring is exactly what I want the infrastructure underneath a patient record to be.

3. The problem underneath consensus — the Byzantine Generals

Every consensus protocol used in blockchains today, permissioned or public, is ultimately answering a single question first posed precisely in 1982, by Leslie Lamport, Robert Shostak and Marshall Pease, in a paper that gave the problem its name and its enduring illustration.[1]

Picture several divisions of the Byzantine army, each commanded by a different general, surrounding an enemy city. The generals can only communicate by messenger. They must collectively decide, and act on, the same plan — attack, or retreat — because a divided army that half-attacks and half-retreats loses regardless of which choice was correct. The complication is that some of the generals might be traitors, who will send different, contradictory messages to different loyal generals specifically to cause that split. The question Lamport, Shostak and Pease answered precisely is: how many loyal generals are needed, and what protocol must they follow, so that all the loyal generals still agree on the same plan no matter what the traitors do?

Their answer, translated out of the military framing: with n generals in total, of whom up to f may be traitors, the loyal generals can always reach agreement if and only if $\mathbf{n \geq 3f + 1}$ — that is, traitors must number strictly fewer than one third of the total. Below that ratio, no protocol, however clever, can guarantee agreement, because a sufficiently large minority of liars can always construct a scenario indistinguishable, from any single loyal general's point of view, from the opposite conclusion being true.

> **Why this is not an abstraction**
>
> Replace “general” with “validator node” and “traitor” with “a node that is compromised, malfunctioning, or lying” and you have exactly the problem a permissioned blockchain's validators face on every single block: agree on the same next entry in the ledger, even though some validators might be faulty or actively malicious, and even though messages between them can be delayed, lost, or arrive out of order. The one-third bound is not a design choice any particular protocol made. It is a mathematical ceiling that *no* protocol can beat, proven in the same 1982 paper. Every practical Byzantine consensus protocol built since — including the one MedLattice runs — is an engineering answer to a question whose theoretical limit was already settled four decades earlier.

For seventeen years after that paper, the theory stayed mostly in the theory. Protocols that tolerated Byzantine faults existed, but they were impractically slow — some required an exponential amount of message-passing in the number of participants, fine for a handful of generals, useless for a real computer system. The paper that changed

that arrived in 1999, and it is the direct intellectual ancestor of everything MedLattice's chain does today.

4. From PBFT to Istanbul BFT to QBFT

Practical Byzantine Fault Tolerance, 1999

Miguel Castro and Barbara Liskov's *Practical Byzantine Fault Tolerance*, presented at OSDI in 1999, is the paper that took the Byzantine Generals result out of pure theory and made it fast enough to run a real, replicated service — their own demonstration was a Byzantine-fault-tolerant network file system, with overhead low enough to be usable in production, not merely correct on paper.[2] PBFT keeps the same one-third bound Lamport's paper proved was unavoidable — it tolerates up to f faulty nodes out of $n \geq 3f + 1$ total — but gets there through a practical three-phase voting protocol among a fixed, known set of replicas: a **pre-prepare** phase where a designated leader (the "primary") proposes an operation and its position in the sequence, a **prepare** phase where replicas broadcast their agreement with each other so that any replica can independently detect if the primary sent different proposals to different replicas, and a **commit** phase where replicas confirm that enough of their peers prepared the same thing, at which point the operation is safely and permanently ordered. Once two-thirds of the replicas have committed, the result is **final** — not probably final, not final after enough confirmations have piled up on top of it the way a Bitcoin transaction becomes safer with each additional block. Final, immediately, the moment enough signatures exist.

That immediate finality is the property every blockchain BFT protocol since has inherited, and it is worth pausing on why it matters for a system like MedLattice rather than treating it as a performance detail. A disclosure anchored to the shared logbook needs to be an unambiguous fact the instant it happens — "this access was granted, recorded, at this position in the history, permanently" — because crypto-shredding, legal-hold enforcement and the 48/72-hour regulatory breach-notification clocks all depend on that fact never later becoming untrue. A consensus protocol with only probabilistic finality would mean the shared logbook's own history could, in principle if not in practice, still be reorganised after the fact — precisely the property the whole design exists to rule out.

Istanbul BFT and IBFT 2.0

PBFT was built for a generic replicated service, not for a blockchain, and adapting it to a chain of blocks — where you specifically want one agreed block per round rather than an arbitrary stream of operations, and where you want the validator set itself to be changeable over time by an on-chain vote — took further work. **Istanbul BFT** (IBFT) was that adaptation, developed within the Ethereum enterprise ecosystem specifically

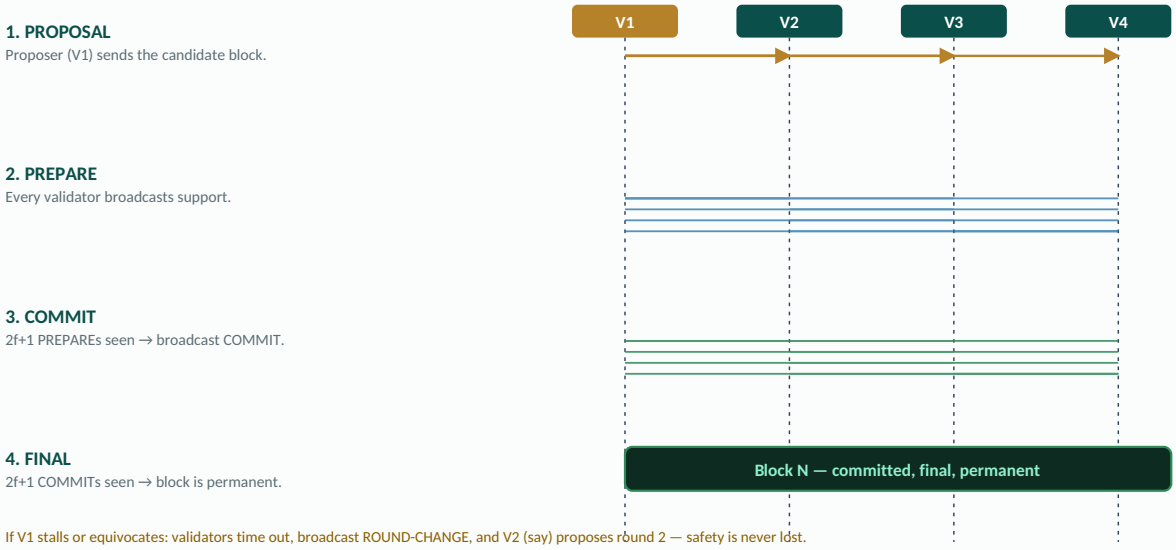
to give a PBFT-style guarantee to an Ethereum-compatible chain: a round-robin proposer, the same prepare/commit voting structure, and validator-set changes handled as ordinary transactions rather than a separate out-of-band process.

The original IBFT had subtle liveness bugs — scenarios, found by later formal analysis, where the protocol could stall rather than either committing a block or cleanly moving to the next round. **IBFT 2.0**, formally specified by Roberto Saltini and David Hyland-Wood and published in 2019, is the corrected version: a proof-carrying redesign that explicitly separates *safety* (loyal validators never agree on two different blocks for the same position) from *liveness* (the network keeps making progress even under partial synchrony, meaning messages can be delayed but not lost forever), and closes the gaps the original protocol left open.[3] It adds a fourth message type beyond PBFT's three — **round-change** — specifically for the case where a proposer is faulty or simply offline: validators that time out waiting for a valid proposal broadcast a round-change vote, and once enough of them agree, the protocol advances to the next round with a new proposer, without ever compromising the one-third safety bound.

QBFT: the version Besu actually runs today

QBFT is the protocol Besu's own documentation now recommends for every new permissioned network, with IBFT 2.0 kept available only for networks that adopted it before QBFT existed.[4] QBFT is formally specified by the Enterprise Ethereum Alliance, published 17 January 2023 as EEA Specification v1, and is described in its own text as based directly on the Istanbul BFT agreement protocol — it is best understood as a cleanup and hardening of IBFT 2.0 rather than a different algorithm: a more precisely specified message encoding chosen specifically so that independently written clients (Besu and GoQuorum, for instance) agree byte-for-byte on what a valid message looks like, which closes a class of cross-client interoperability bugs IBFT 2.0 left as an implementation detail.[5]

Mechanically, a QBFT round looks like this. For each new block, a proposer is selected deterministically from the current validator set (round-robin, or by another agreed rule). The proposer broadcasts a **PROPOSAL** containing the candidate block. Every validator that receives a valid proposal broadcasts a **PREPARE** vote for it. Once a validator has collected PREPARE votes from at least **2f + 1** of the *n* validators (itself included) — the same one-third-tolerance threshold Lamport's paper proved was the ceiling — it broadcasts a **COMMIT**. Once a validator collects 2f + 1 COMMIT votes, the block is final: appended to the chain, permanently, with no further confirmations needed or meaningful. If a proposer stalls or sends conflicting proposals, validators time out and exchange **ROUND-CHANGE** votes, and the process restarts at the next round with the next proposer — without ever losing the guarantee that two honest validators cannot finalise two different blocks at the same height.



One QBFT round among four validators (n = 4, tolerating f = 1 faulty validator, so 2f + 1 = 3 votes are enough to finalise). Propose, prepare, commit — the same three phases Castro and Liskov specified in 1999, with round-change added by IBFT 2.0 to handle a faulty or absent proposer without ever risking two different blocks being finalised at the same height. Message overhead grows with the square of the validator count, which is exactly why this protocol belongs on a consortium of tens of known validators and not on a public network of unknown size.

The cost hiding in that diagram

Every validator broadcasts to every other validator, twice (once for PREPARE, once for COMMIT), so message volume grows as $O(n^2)$ in the number of validators. For MedLattice's consortium — a handful of hospitals, a laboratory network, a regulator, likely low tens of validators even at full national scale — that is a non-issue; QBFT networks are used in production with dozens of validators and sub-second block times. It would be a real problem at the scale of a public chain with thousands of anonymous participants, which is precisely why QBFT is never proposed as a replacement for proof-of-stake on public Ethereum, and why it is exactly right for a closed consortium.

5. Why Besu, specifically

Choosing QBFT answers *how* the validators agree. It does not answer which piece of software they run to do it, and that is a genuinely separate decision — QBFT is a protocol specification, not a product, and more than one client implements it.

Hyperledger Besu began life as *Pantheon*, an Ethereum client launched in November 2018 by PegaSys, the protocol engineering team inside ConsenSys. PegaSys contributed it to the Hyperledger project in August 2019, at which point it was renamed Besu, with the stated goal of "lowering barriers to entry for enterprises" while remaining a fully compliant client of Ethereum mainnet itself.[6] It is an open-source, Apache 2.0-licensed, Java-based Ethereum client, and it is the same Hyperledger — now reorganised, along with Fabric, Indy and the rest, under the Linux Foundation's broader **LF Decentralized Trust** umbrella — whose foundation runs the wider governance around it.[7]

The property that actually decided this for MedLattice is that Besu is, underneath the permissioning layer, a *real Ethereum client*: full EVM (Ethereum Virtual Machine) execution, the standard JSON-RPC interface, and complete compatibility with Solidity and the entire Ethereum tooling ecosystem — the same compiler, the same testing frameworks, the same contract-interaction libraries a public Ethereum project would use. That matters for a reason that has nothing to do with ideology: it means MedLattice's three contracts are written in the most widely audited, most widely taught smart-contract language that exists, compiled with a mainstream, actively maintained compiler (solc), and can in principle be verified, tested and reasoned about by any Ethereum-literate engineer or auditor without first learning a bespoke platform. A permissioned chain that requires proprietary tooling to inspect is a permissioned chain fewer outside experts can actually be asked to check.

The most direct alternative, **Hyperledger Fabric**, takes a structurally different approach worth naming precisely because it clarifies what Besu is not: Fabric separates transaction execution ("chaincode", written in Go, Java or Node.js) from ordering from validation as three distinct phases, and its consensus is *pluggable* — Raft or a similar crash-fault-tolerant ordering service is typical, rather than a Byzantine-fault-tolerant vote among all peers by default.[8] Fabric's channel model gives strong built-in data-partitioning between subsets of participants, which is a genuinely attractive property for some consortium designs. I did not choose it, for a reason specific to this project rather than a general verdict on Fabric: MedLattice's compartmentalisation already happens at the cryptographic layer — the drawers, the envelopes, the key tree — independently of the ledger, so Fabric's channel-level partitioning would have been

solving a problem I had already solved one layer down, at the cost of leaving Solidity and the Ethereum tooling ecosystem behind for a bespoke chaincode model. R3 Corda, the other name that comes up in this space, makes an even more specific bet on a shared-nothing, pairwise-transaction model aimed squarely at financial contracts between two counterparties — a good fit for its home domain, a poor structural fit for a multi-party shared record every consortium member needs a consistent view of.

Zooming out one level further: Besu is one member of a family, not a solo project. The Linux Foundation launched Hyperledger on 17 December 2015 with 21 founding members, specifically as an umbrella for more than one approach to enterprise distributed ledgers rather than a single blockchain product — a deliberate acknowledgement that "enterprise blockchain" covers genuinely different problems.[9] Alongside Besu and Fabric sit **Sawtooth** (built around parallel transaction execution for high throughput), **Indy** (purpose-built for decentralised digital identity, with correlation-resistant identifiers so two credentials from the same holder can't be linked by a verifier who shouldn't be able to), **Iroha** (a simpler C++ platform aimed at mobile and consumer-facing applications, with common operations built in as first-class commands rather than requiring custom contract code), **FireFly** (a middleware layer — Kaleido calls it a "SuperNode" — that sits in front of a chain and gives application developers token, data and event APIs instead of raw chain access), and **Cacti** (an interoperability framework for linking transactions across otherwise unrelated ledgers). None of that breadth changes the argument this note makes: MedLattice needed EVM/Solidity compatibility and BFT consensus with a known validator set, which is Besu's specific niche within that family, not a property of Hyperledger as a whole.

Property	Hyperledger Besu (QBFT)	Hyperledger Fabric
Execution model	Full EVM, every validator executes every contract call	Execute-order-validate; chaincode runs on an endorsing subset
Contract language	Solidity / Vyper — the Ethereum-standard stack	Chaincode in Go, Java or Node.js
Consensus	QBFT — Byzantine-fault-tolerant vote among all validators	Pluggable ordering service, typically Raft (crash-fault-tolerant only)
Finality	Immediate on 2f+1 commits	Immediate once the ordering service sequences a block
Data partitioning	None built in — handled at the application/crypto layer	Channels give native, ledger-level partitioning between subsets of members
Tooling ecosystem		Fabric-specific SDKs and tooling

The full public-Ethereum toolchain: solc, ethers, standard auditors

6. What actually runs, on MedLattice's own chain

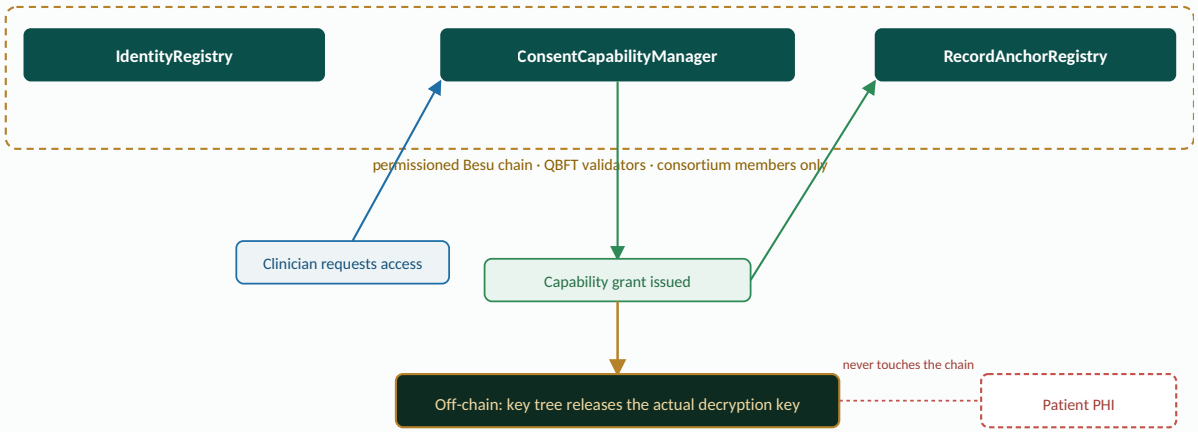
None of the preceding five sections are hypothetical for this project — they describe the network MedLattice's reference implementation already compiles and runs against. The chain is a permissioned Besu network running QBFT, with the validator set restricted to consortium members: participating hospitals, the laboratory network, and the relevant regulatory body, exactly the closed-ring picture in Section 2's diagram rather than the open cloud beside it.

Three Solidity contracts, compiled with `solc` 0.8.28 under `viaIR: true` (the newer IR-based compilation pipeline, needed once the contracts got complex enough that the legacy code generator's stack-depth limits started to bite), constitute the entire on-chain logic:

- **IdentityRegistry** — who exists on the network, in which role, holding which public keys, and until when. No patient identifying information; only participant and role records.
- **ConsentCapabilityManager** — the authorisation kernel. A four-predicate check — who is asking, for what, on what legal or clinical basis, and whether a live relationship or emergency justification actually exists — gates every capability grant, and the grant itself, once issued, is what a reader presents to derive the cryptographic keys described in Not Shown Is Not Locked. This is where the shared logbook and the key tree meet: the ledger decides *whether* a key may be released; it never holds the key itself.
- **RecordAnchorRegistry** — a tamper-evident record of disclosures and refusals, with a daily fingerprint lodged externally so that even a majority collusion among consortium members would still leave detectable evidence of tampering after the fact.

All three run under a full test harness — 77 of 77 tests passing as of the current specification revision — executed against a real EVM rather than a mock. Foundry, the more common modern Solidity toolchain, could not be reached from inside the cloud build environment I was working in (its installer domain is not on the permitted network list there), so the working build instead uses plain npm-distributed `solc` together with an `@ethereumjs/vm` harness — a smaller, more manual toolchain, but one built entirely from the same standard Ethereum tooling Section 5 argued was the whole point of choosing Besu in the first place. One practical scar from that build: ethers' test bindings reject overloaded Solidity function names as ambiguous, which is why the AI-disclosure logging function is named `logModelDisclosure` rather than an overload of the human-disclosure `logDisclosure` — a small naming decision, but the kind of thing

that only shows up once you are actually compiling and testing against real tooling rather than describing the design on paper.



QBFT decides how the validators agree on ledger entries; these three contracts decide what those entries mean. A request checks identity and the four-predicate consent kernel, an approved request becomes a recorded capability grant, and that grant — not the patient data itself — is what a reader takes off-chain to derive a decryption key from the key tree described in *Not Shown Is Not Locked*. The chain anchors who was authorised to do what, and never sees what they actually read.

7. What this does not do

Consistent with how I have written about every other part of this design: a fair account has to include the parts that are not fully resolved.

The gap I have already flagged once, and it belongs here too

Besu's own block-level signatures — the ones validators use to sign their PREPARE and COMMIT votes and, on the underlying account layer, ordinary Ethereum transaction signatures — are standard ECDSA over the secp256k1 curve, exactly as on public Ethereum. That is classical, not quantum-resistant, cryptography, and I said as much in Not Shown Is Not Locked when discussing the ledger rather than the record. It stays a real, named gap rather than a solved one. What limits it: the network is closed to outside participants by the permissioning layer itself, no patient confidentiality depends on this signature scheme at all — that is entirely the job of the ML-KEM/ML-DSA/SLH-DSA stack operating one layer up, at the application and record level — and a QBFT-compatible post-quantum signature replacement is a swap-in-place change to the validator layer that does not require touching the three contracts described above. It is a gap I can see the shape of the fix for. It is a gap today.

A governance question the mathematics cannot answer

QBFT guarantees that *if* fewer than a third of the current validators are faulty, the loyal two-thirds will agree. It says nothing at all about how the validator set itself is decided — which hospitals get a seat, how many seats a regulator holds relative to a hospital consortium, what happens if a validator organisation is acquired, ceases operating, or is later found to have been acting in bad faith the whole time. That is pure governance, decided by whoever controls validator-set changes on the network, and no amount of consensus-protocol rigour substitutes for getting that governance structure right. It is the same category of open question the spec already tracks by number for other parts of the design, and I do not think it has had the same explicit sign-off yet.

The scale this does not, and should not, try to reach

$O(n^2)$ message complexity is a hard ceiling on how many validators a QBFT network can practically run — production networks run comfortably into the dozens, not the thousands. That is not a weakness for a hospital consortium; it is the correct size for one. It would be the wrong protocol entirely for anything meant to onboard an unbounded public of unknown participants, which is one more reason the permissioned/public fork in Section 2 is not merely a preference but a structural precondition for everything downstream of it.

8. Glossary

Term	What it means here
Besu	An open-source, Java-based Ethereum client, originally PegaSys's Pantheon, contributed to Hyperledger in 2019, usable on both public Ethereum and private permissioned networks
Byzantine fault	A failure in which a participant behaves arbitrarily — including actively lying or sending contradictory messages — rather than simply crashing
Consensus	The process by which replicated, mutually distrusting nodes agree on the same next entry in a shared ledger
EVM	Ethereum Virtual Machine — the standard execution environment every Ethereum-compatible node runs smart contracts in, identically
Finality	The point at which a committed block is permanent and cannot be reorganised. QBFT gives immediate finality; proof-of-work chains give only probabilistic finality
IBFT 2.0	The corrected, formally specified 2019 version of Istanbul BFT, fixing liveness bugs in the original protocol
PBFT	Practical Byzantine Fault Tolerance — Castro and Liskov's 1999 protocol that made Byzantine consensus fast enough for real systems
Permissioned chain	A blockchain whose validator set is restricted to vetted, known organisations, as opposed to open to anyone
QBFT	The EEA-specified consensus protocol, based on Istanbul BFT, that Besu now recommends for all new permissioned networks
Smart contract	A program stored on a shared ledger and executed identically by every node, so its logic cannot be quietly altered by any one party
Validator	A node authorised to propose and vote on new blocks under the consensus protocol

9. References

1. **Lamport, L., Shostak, R., Pease, M. — The Byzantine Generals Problem (1982)**
   ACM Transactions on Programming Languages and Systems, Vol. 4, No. 3, pp. 382–401. Author's own copy: https://lamport.azurewebsites.net/pubs/byz.pdf
2. **Castro, M., Liskov, B. — Practical Byzantine Fault Tolerance (1999)**
   Proceedings of OSDI 1999. https://css.csail.mit.edu/6.824/2014/papers/castro-practicalbft.pdf · USENIX record: https://www.usenix.org/conference/osdi-99/practical-byzantine-fault-tolerance
3. **Saltini, R., Hyland-Wood, D. — IBFT 2.0: A Safe and Live Variation of the IBFT Blockchain Consensus Protocol for Eventually Synchronous Networks (2019)**
   arXiv:1909.10194. https://arxiv.org/abs/1909.10194
4. **Besu documentation — Consensus protocols for private networks**
   Confirms QBFT as the current recommended enterprise consensus protocol, IBFT 2.0 kept for backward compatibility, Clique and Ethash no longer supported. https://docs.besu-eth.org/private-networks/how-to/configure/consensus
5. **Enterprise Ethereum Alliance — QBFT Blockchain Consensus Protocol Specification v1 (17 January 2023)**
   https://entethalliance.org/specs/qbft/v1/
6. **Hyperledger Foundation — Announcing Hyperledger Besu (29 August 2019)**
   Origin of Besu as PegaSys's Pantheon (launched November 2018) and its contribution to Hyperledger. https://www.lfdecentralizedtrust.org/blog/2019/08/29/announcing-hyperledger-besu
7. **LF Decentralized Trust — Besu project page**
   https://www.lfdecentralizedtrust.org/projects/besu · general Hyperledger/LF Decentralized Trust background: https://en.wikipedia.org/wiki/Hyperledger
8. **Kaleido — Comparing Hyperledger Fabric and Hyperledger Besu: A Deep Dive**
   Execution model, consensus and tooling comparison referenced in Section 5. https://www.kaleido.io/blockchain-blog/comparing-hyperledger-fabric-and-hyperledger-besu
9. **LF Decentralized Trust — Hyperledger Turns Five (17 December 2020)**
   Confirms the Hyperledger launch date (17 December 2015) and founding membership (21 organisations), cited in Section 5. https://www.lfdecentralizedtrust.org/announcements/2020/12/17/hyperledger-turns-five
10. **Besu documentation — home / getting started**
    https://docs.besu-eth.org/ (besu.hyperledger.org now redirects here)
11. **NIST FIPS 203 — Module-Lattice-Based Key-Encapsulation Mechanism Standard (August 2024)**
    The ML-KEM standard referenced in Section 7 and detailed in full in *Not Shown Is Not Locked*. https://csrc.nist.gov/pubs/fips/203/final